Information Management Technology Library

# EDIT User's Guide and Reference Manual

**Abstract** This manual describes how to use the line editor (EDIT) and screen-mode editor (EDIT VS) provided with the EDIT program. It provides descriptions of the commands and their ranges and information on creating and using files. It also addresses error handling and page-mode editing.

**Part Number** 058061

**Tandem Computers Incorporated**

**Document History**

| Edition | Part Number | Product Version | OS Version | Date |
|---|---|---|---|---|
| First Edition | 84082 | EDIT B30 | GUARDIAN 90 C00 | November 1987 |
| Second Edition | 058061 | EDIT B30 | GUARDIAN 90 C00 | July 1991 |

**Note**

The second edition of this manual was reformatted in July 1991; no changes affecting product functionality were made to the manual content at that time.

New editions incorporate any updates issued since the previous edition.

# Contents

## Section 3 Revising Text in Your File

## Section 4 EDIT Command Summary

## Appendix C Page Mode Editing

## Figures



## Tables

# Welcome to EDIT

## What This Manual Is About

The EDIT program provides two editors—EDIT, a line editor, and EDIT VS, a screen editor. The emphasis of this manual is on EDIT. The manual introduces you to the features and capabilities of EDIT, describes the many EDIT commands and their ranges, and provides information on creating and using EDIT files. For those users who wish to use EDIT VS, a description of page mode editing follows the EDIT material.

## Who Should Use This Manual

This manual assumes that you know how to:

- ☐ Log on to the Tandem system
- ☐ Start EDIT from the application that is running on your terminal

If you need to know how to log on, read the appropriate paragraphs in the *GUARDIAN 90 Operating System Utilities Reference Manual* or ask a knowledgable person to help you.

If there is an application running on your terminal and you do not know how to start the EDIT program from this application, ask a knowledgable person to help you.

The user's guide sections in the beginning portion of the manual are designed to introduce new users to text editing and the more commonly used commands of EDIT. Experienced users can read through the introduction then move directly to the reference material discussing EDIT commands and ranges in Sections 4 and 5.

## How to Use This Manual

Use the Table of Contents and the Index to guide you to the appropriate section or page number.

Section 1, "Introduction to EDIT," contains an overview of the EDIT program including a description of the line editing concept, creating an EDIT file, how to use the line editor, typing EDIT commands, a brief explanation of ranges, and how to exit the EDIT program.

Section 2, "Adding and Displaying Text," describes how to use the ADD and LIST commands to add and display the text in your file.

Section 3, “Revising Text in Your File,” describes how to use eight of the more commonly used EDIT commands for editing and formatting the text in your file.

Section 4, “EDIT Command Summary,” describes each of the EDIT commands, providing the syntax description, a discussion of how the command is used, examples, and tips. In addition, this section explains how to run the EDIT program interactively and noninteractively and provides more information on range parameters.

Section 5, “Range Summary,” details the concept of ranges in an EDIT command, describes each of the nine range parameters, and provides examples of each.

Section 6, “Handling Your EDIT Files,” describes how to create a backup copy of your EDIT file, how files are named, which characters are printable, and how to compress an EDIT file to maximize the space on the disk.

Appendix A, “EDIT Error Messages,” lists the error messages you might receive while using EDIT, describes what they mean, and suggests how to correct the problem that generated the message.

Appendix B, “EDIT Error Recovery Procedure,” guides you through the recovery procedure for EDIT.

Appendix C, “Page Mode Editing,” describes the page mode editing options of the EDIT program—the EDIT VS program and EDIT commands that allow page mode editing from EDIT.

Appendix D, “EDIT VS Error Messages,” lists the error messages you might receive while using EDIT VS, describes what they mean, and suggests how to correct the problem that generated the message.

Appendix E, “EDIT VS Error Recovery Procedures,” guides you through several recovery procedures for EDIT VS.

## Notation Conventions

You must enter commands in a certain form so that EDIT understands them. This form is called syntax—it explains how to enter a command.

These conventions have been established for representing syntax to save space and to avoid having to repeat a lot of information. Uppercase and lowercase letters may have specific meanings, and certain other symbols are also used to explain how to enter the command. Spaces and commas separate the parts of the commands (command name, options, and keywords).

Each command's particular syntax is presented in Section 4, "EDIT Command Summary." The syntax for the different range parameters is presented in Section 5, "Range Summary."

| Notation | Meaning |
|---|---|
| UPPERCASE LETTERS | Uppercase letters represent keywords and reserved words: you can enter these items in any combination of uppercase or lowercase letters. You can also abbreviate a keyword to its first letter. |
| *lowercase italic letters* | Lowercase italic letters represent variable items that you supply. |
| Brackets [ ] | Brackets enclose optional syntax items. A group of vertically aligned items enclosed in brackets represents a list of selections from which you can choose one or none. |
| Braces { } | Braces enclose required syntax items. A group of vertically aligned items enclosed in braces represents a list of selections from which you must choose one. |
| Vertical line \| | A vertical line separating items in a horizontally aligned list of options, enclosed in either braces or brackets, is an alternative to vertical alignment of the selections. |
| Ellipsis ... | An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed syntax items any number of times. |
| Spaces | If two items are separated by a space, that space is required between the items. (But if either of the items is a punctuation symbol, such as a parenthesis or a comma, the space is optional.) |
| "*string*" | A string is one or more characters enclosed in quotation marks. The quotation marks are part of the symtax and must be included in the command line. |
| Punctuation | Parentheses, commas, semicolons, and other symbols not described above must be entered precisely as shown. |

## What's New in This Manual

This manual is an update of the previous *EDIT Manual* and includes information contained in the *PS MAIL Extended-Editor User's Guide for TTY Terminals*. New or changed information includes:

- ☐ TEDIT, the new EDIT command that starts the TEDIT editing program.
- ☐ A new presentation of the range parameter material. The ranges have been reorganized into nine parameters, which are described in detail in Section 5.
- ☐ The various page mode editing options, which have been combined and described in detail in Appendix C. These options include EDIT VS, the screen editing program, as well as three EDIT commands that are useful to users having terminals with full-screen capabilities.
- ☐ A new presentation of the EDIT tutorial material. Sections 2 and 3 present a condensed version that describes some of the more commonly used EDIT commands.
- ☐ The error messages and recovery procedures, which have been rearranged. Appendixes A and B describe error messages and the recovery procedure for EDIT, while those for EDIT VS are presented in Appendixes D and E.

## Where to Go for More Information

The following manuals, also referenced in the text where appropriate, provide supplementary information:

- ☐ The *GUARDIAN 90 Operating System Utilities Reference Manual* describes the system software underlying the EDIT program.
- ☐ The *PS TEXT EDIT and PS TEXT FORMAT User's Guide* and the *PS TEXT EDIT Reference Manual* describe the features and commands of the TEDIT editing program, which you can start from EDIT.

To order copies of the manuals, contact your Tandem sales office.

# 1 Introduction to EDIT

## The EDIT Program

EDIT is a text editing program. A text editor is a tool that lets you use a terminal to write documents and store them on the computer. Depending on your needs, the documents that you write might be business correspondence, a letter, or an entire manual; more advanced uses of EDIT might be creating an OBEY file or the source text of a computer program.

You use a terminal and EDIT to:

- ☐ Create an EDIT file
- ☐ Type a document in an EDIT file
- ☐ Make changes and corrections to the document in the EDIT file
- ☐ Start TEDIT (a more advanced Tandem editor) from a product that starts EDIT by default

The computer system stores the EDIT file on disk. At any time, the writer can use the text editor from any terminal connected to the computer system to make changes and corrections to this EDIT file.

## Writing and Editing With EDIT

Writing a document using EDIT is different from writing a document using a typewriter and paper. With EDIT, editing what you write is easy:

- ☐ You can correct mistakes and make changes as you write.
- ☐ You can return to your document at any time to correct mistakes and make changes.

The editing you do can be as simple as correcting typing mistakes or as sophisticated as moving lines of text from one document into another. For example, you can use EDIT to write the lines:

```
He who whispers down a well
About the goods he has to sell,
Will never reap the golden prize
Like him who learns to advertise.
```

Then, at a later time, you can use EDIT to change these lines to:

```
The codfish lays ten thousand eggs.
The homely hen lays one.
The codfish never cackles
To tell you what she's done.
And so we scorn the codfish,
While the humble hen we prize,
Which only goes to show you
That it pays to advertise.
```

## Capabilities of the EDIT Program

EDIT makes it easy for you to:

- ☐ Create the EDIT file that will contain your document
- ☐ Add text to and delete text from the EDIT file
- ☐ Correct typing errors
- ☐ Read through the text in the EDIT file
- ☐ Modify text in your file
- ☐ Move text from one place to another in your file
- ☐ Break, lengthen, and shorten lines
- ☐ Insert text from another document into your file
- ☐ Copy your text into another EDIT file
- ☐ Renumber part or all of the lines in your file
- ☐ Search for a specific text string
- ☐ Start TEDIT
- ☐ Make global changes to the text
- ☐ View a listing of your files

The EDIT program comprises two editors, a line editor (EDIT) and a screen editor (EDIT VS, for virtual screen). The emphasis of this manual is on EDIT, which is described in Sections 1 through 6. If you want to learn more about EDIT VS, refer to the information in Appendix C.

## Creating an EDIT File

To begin writing a document, you must create an EDIT file to contain the text of that document. You can do that in two steps: start the EDIT program, then create and name your EDIT file.

1. Start the EDIT program at the command interpreter prompt. This prompt is the prompt displayed by the program that communicates between your terminal and your computer system. If TACL (Tandem Advanced Command Language) is running on your system, the standard prompt character is a number, followed by an angle bracket, and then a space (for example, 1 is your first prompt, 2 is the next one, and so on). Less commonly, a system running COMINT displays a colon as the command interpreter prompt.

   For the sake of simplicity, the TACL prompt represents the command interpreter prompt throughout this manual.

   You can type a command interpreter command at a TACL or COMINT prompt character to start any number of programs. Typing EDIT at the command interpreter prompt starts the EDIT program.

2. Create and name the EDIT file when you start the EDIT program or when EDIT prompts you for a file name. For example, start the EDIT program, then create and name an EDIT file at the same time by typing:

```
2> EDIT POEMS
```

   EDIT responds with the question:

```
TEXT EDITOR - T9601B30 - (08MAR87)
$WORK.FICTION.POEMS DOES NOT EXIST.  SHALL I CREATE IT?
```

   Type:

```
YES
```

in answer to the question “SHALL I CREATE IT?”. EDIT prints:

```
CURRENT FILE IS $WORK.FICTION.POEMS
*
```

You have started the EDIT program and created an EDIT file named POEMS, which will contain your document. You are now ready to type commands at the prompt for the EDIT program, which is an asterisk (*).

If you don’t provide a name of a file, EDIT either prompts you for the file name as you begin editing or warns you that the file is undefined.

For example, if you start EDIT and want to begin adding text:

```
TEXT EDITOR - T9601B30 - (08MAR87)
*add
    1     Sing a song of sixpence,
NAME THE NEW FILE:
```

If you choose to provide the name of a file, answer the prompt by typing the file name, press RETURN, and continue adding text to the new file.

If you do not name the file and simply press RETURN at the “NAME THE NEW FILE” prompt, EDIT creates a temporary file and warns you that the file is undefined. You can continue to add text to an undefined file, although EDIT can’t retrieve the text if you leave the file (and you don’t save the text elsewhere). For more information about temporary EDIT files, see “Adding Text to a New EDIT File” in the ADD command description in Section 4.

**Note** $WORK.FICTION.POEMS means there is a file named POEMS, in the subvolume named FICTION, in the volume named $WORK. The file named POEMS that you just created will have the volume name of your current volume and the subvolume name of your current subvolume. For more information about file names, subvolume names, and volume names, see “How Files Are Named” in Section 6.

## Using the Line Editor

You can think of line editing as interactive—or conversational—editing. You and EDIT have a conversation about the lines of text in your file. You communicate with EDIT by typing commands and text, then pressing RETURN. EDIT responds by doing what you tell it to and by displaying prompts at your terminal. Figure 1 illustrates the concept of line editing with the EDIT program.

### The EDIT Prompt

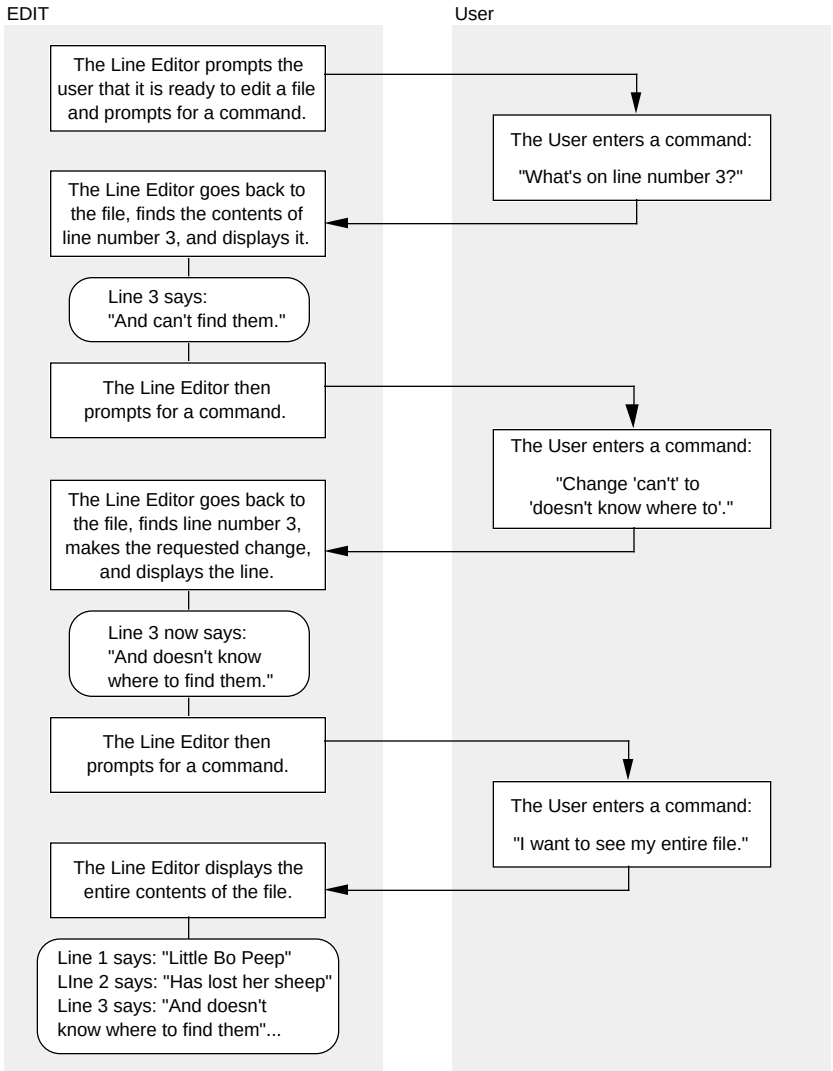
Once you have started the EDIT program from your command interpreter prompt, you'll receive the EDIT prompt (an asterisk). You can type any EDIT command at the asterisk. After typing a command line or a new line of text, press RETURN to tell EDIT to execute what you have typed.

### The RETURN Key

This manual presents examples that show how EDIT works. In many of them, an imaginary user types in commands or text in response to prompts or queries. As mentioned previously, you must press RETURN after you type a command line or line of text. Therefore, in the examples, there is an implicit RETURN at the end of each window of user input. For those cases when the user just needs to press RETURN, the boxed word RETURN is all that appears beside the asterisk prompt.

The RETURN Key

**Figure 1-1. Line Editing With the EDIT Program**



When you simply press RETURN at the asterisk prompt (instead of entering an editor command or text), EDIT responds by listing the next line of text in the EDIT file. For example, if your EDIT file contains the lines:

```
 7     I am only one,
 8     But still I am one.
 9     I cannot do everything,
10     But Still I can do something
11     And Because I cannot do everything
12     I will not refuse to do the something
13     that I can do.
```

then:

Lists the text on line number 7

Lists the text on line number 8

Lists the text on line number 9

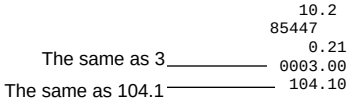
```
*LIST 7
    7     I am only one,
* RETURN
    8     But still I am one.
* RETURN
    9     I cannot do everything,
*
```

If you continue to press RETURN until EDIT returns just the asterisk, you have arrived at the end of the file, and EDIT cannot find any more text lines to return.

## Line Numbers

You use EDIT to display, add text to, and change text in an EDIT file. EDIT assigns a line number to each line of text that you add to an EDIT file and stores each line of text in numerical order within the EDIT file. Although you can see the line numbers along the left margin of the screen (for example, after you request a LIST command), the numbers are not part of your actual document.

A line number can have from one to five digits, followed by a decimal point and from one to three digits. Thus, your file can have line numbers anywhere from 0 to 99999.999. Some examples of line numbers used by EDIT are:




As you use EDIT, you see the line numbers on your screen along with the text they contain.

Line numbers are important to the line editor, primarily because you often need to use specific line numbers when you use editor commands. When you type an editor command, you ask the EDIT program to "operate" on one or more lines of text in your EDIT file. These one or more lines are considered "the range" for that editor command and tell EDIT the lines of text on which you want the EDIT command to operate. In the EDIT program, you see the line numbers assigned to the text in your EDIT file so you can easily use them when you need to include a range with an editor command.

Ranges are discussed in more detail in Section 5.

## Typing Commands

You can type command names and keywords (words that modify the action of a command) in any combination of uppercase and lowercase characters. Using the LIST command and the ALL keyword for an example, all these following commands are equivalent:

```
*list all              *lIsT aLl

*List All              *LIST ALL
```

You can likewise abbreviate any command name and keyword to its first letter. The following commands, for example, are all equivalent:

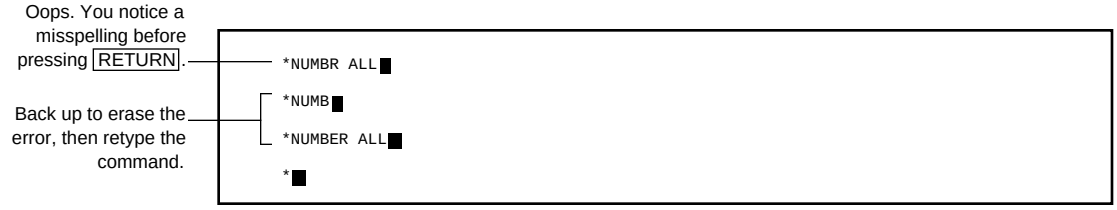
```
*L A                    *La

*l a                    *LIST ALL
```

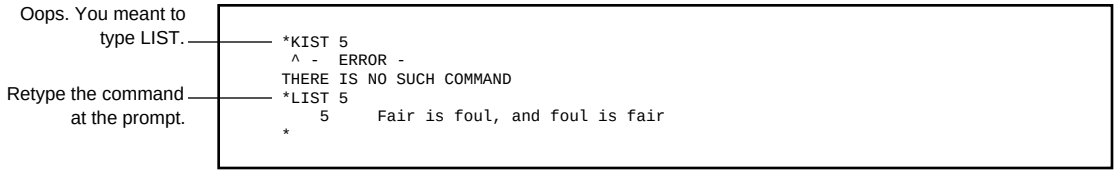
You can also combine more than one command on a single command line. See Section 4, "Typing Several Commands on One Command Line," for more information.

## Correcting Mistyped Editor Commands

Commands can be short, like ADD 1, or quite long, like CHANGE ALL "OLD"NEW" NUM 2/25. If you notice a mistake before you press RETURN, simply use the BACKSPACE key to back up to your mistake and retype the remainder of the line. For example:




If you discover an error in a short command after pressing RETURN, you may find retyping the command is easiest:




If, however, your command line is longer, you can use FC (FIX COMMAND) to alter the last command line if it needs a change or a correction. Using FC for this kind of error saves you from retyping the whole command line. (See Section 4 for a detailed description of the FIX COMMAND.)

For example, assume you have a file in your default subvolume called MYPOEM. You type the command:

```
*GET MPOEM 1/10 TO 40;LIST ALL
```

and EDIT returns the error message:

```
??:011
FILE  $SYSTEM.USER.MPOEM  CANNOT BE OPENED
```

because you don't have a file named MPOEM.

Type FC to tell EDIT to display the most recent command line for editing:

```
*FC
(COMMAND) GET MPOEM 1/10 TO 40;LIST ALL
..........
```

The R, I, and D subcommands of FC work exactly the same when you're editing a command line as they do when you're editing a text line. For example:

```
(COMMAND) GET MPOEM 1/10 TO 40;LIST ALL
..........     iy
(COMMAND) GET MYPOEM 1/10 TO 40;LIST ALL
..........RETURN
```

Pressing RETURN when you've finished editing causes EDIT to execute the corrected command.

**Note** EDIT does not execute the command line after FC if you type two right slants (//) in columns 1 and 2 after the FIX prompt (the 10 periods) and press RETURN

### Ranges

For many editor commands, EDIT operates on one or more lines of text in your file. So, when you type many of the commands, you are actually specifying both a command name and a range of text to be operated on. A range is the one or more lines of text on which you want the command to operate.

You specify ranges in a variety of ways. The simplest ways are with a line number (for a single line) or two line numbers separated by a slash (for two or more consecutive lines). You can specify a range of one or more columns within your file. You can also specify a range of lines that contains a particular character string (a character string is a series of characters such as a word, a phrase, or a number enclosed in quotation marks).

Ranges are an integral part of many editor commands. For more information about ranges, see “Introduction to Ranges” in Section 4. For a thorough discussion as well as detailed descriptions of each type of range parameter, see “Range Summary” in Section 5.

### Printable Characters

When typing text in your file, use only printable characters. Typing nonprintable characters in your file (such as any character that you would type while holding down CONTROL has unpredictable results and may adversely affect the text in your file. The printable characters are:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
{ } ! @ # $ % ^ & * ( ) - _ +
= [ ] ~ " ` : ; ? / > . < ,
blank space

## Conversation With EDIT

In the following sequence, you can step through a conversation with EDIT. Notice how the terminal displays your conversation. When you want to see lines of text, you must tell EDIT to list them on the screen. EDIT lists the line number and the text. Then, you use commands to tell EDIT to operate on specific lines of text. EDIT operates on the text, then prompts you for another command.

1. Start the EDIT program to edit an existing file named SHAKE. EDIT prompts you for an editor command with an asterisk. When you see an asterisk on the terminal, you know EDIT is waiting for you to type an editor command.

```
1> EDIT SHAKE
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.FICTION.SHAKE
*
```

2. You type a command to list all the text lines in your EDIT file. EDIT reads your EDIT file and lists all its text lines, in numerical order, on the screen. Then EDIT prompts you for another command. (If you misspell an editor command name, EDIT has no idea what you want it to do, so it returns an error message.)

```
1> EDIT SHAKE
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.FICTION.SHAKE
*LIST ALL
    1     The world's a stage, where God’s
    2     omnipotence His justice, knowledge,
    3     love, and providence Do act the parts.
*
```

3. You type an editor command to fix the text on line number 1. EDIT displays the line to be fixed on the screen. When you have finished fixing line 1, press RETURN at the FC prompt dots and EDIT prompts you for another editor command.

```
*LIST ALL
    1     The world's a stage, where God's
    2     omnipotence His justice, knowledge,love,
    3     and providence Do act the parts.
*FIX 1
    1     The world's a stage, where God's
.........omnipotence diAll t//       ddddddddddd
    1     All the world's a stage,
.........RETURN
*
```

4. You type an editor command to delete the text on line number 2. EDIT displays the deleted line on the screen, then prompts you for another command.

```
*FIX 1
    1     The world's a stage, where God's
.........omnipotence diAll t//       ddddddddddd
    1     All the world's a stage,
.........RETURN
*DELETE 2
    2     His justice, knowledge, love, and
          providence
*
```

5. You type an editor command to add text beginning at line number 2. EDIT prompts you with the number 2. You type a line of text. EDIT prompts you with the number 2.1. You type the next line of text. EDIT prompts you with the number 2.2. You type two right slants (/ /) to tell EDIT that you're finished adding text now. Then EDIT prompts you for another editor command.

```
*DELETE 2
    2     His justice, knowledge, love, and
          providence
*ADD 2
    2     And all the men and women merely
    2.1   players, They have their exits and their
          entrances,
    2.2   //
*
```

6. You type an editor command to change the text on line number 3. EDIT displays the changed line on the screen and prompts you for another editor command.

```
*ADD 2
    2     And all the men and women merely
          players,
    2.1   They have their exits and their
          entrances,
    2.2   //
*CHANGE "Do act the"And one man in his time
plays many" 3
    3      And one man in his time plays many
           parts.
*
```

7. You type an editor command to list all the text lines in your EDIT file. EDIT reads your EDIT file and lists all its text lines, in numerical order, on the screen. Then EDIT prompts you for another editor command.

```
*CHANGE "Do act the"And one man in his time plays
          many" 3
    3     And one man in his time plays many
          parts.
*LIST ALL
    1     All the world's a stage,
    2     And all the men and women merely
          players,
    2.1   They have their exits and their
          entrances,
    3     And one man in his time plays many
          parts.
*
```

## The EXIT Command: Leaving Your File

To exit from the EDIT program, simply type:

```
*EXIT
```

The EXIT command closes your current EDIT file, terminates the editing session, and returns you to the command interpreter or the program that was running at your terminal before you called EDIT.

If you need more information, see the discussion of the EXIT command in Section 4.

# 2 Adding and Displaying Text

## The ADD Command: Adding Text to Your File

Once you have created an EDIT file, you are ready to use EDIT to write a document in your file. Or, you might already have a file that you want to continue working on. To add text to a file, use the ADD command. EDIT provides you with several ways to add text. You can:

- ☐ Add text to the end of your file
- ☐ Add text between existing lines by using decimal line numbers from .001 to 10
- ☐ Add lines of text before the first line of your file

The simplest way to add text is to ask EDIT to add it at the next available line. You do this by typing the ADD command at the prompt (*). EDIT responds by adding text to the end of your file, or, if you've been previously adding text, to the place where you last added text.

For example, if you have no previous text, the command:

```
*ADD
   1
```

starts you on line 1. If, however, you already have 35 lines of text, the command:

```
*ADD
  36
```

starts you on the line number (36) after the last line of text in your file. But suppose you last added lines 35 through 40 to a file with 200 lines. The command:

```
*ADD
  41
```

starts you on the line number (41) after the line you most recently added.

To insert text into an existing file, you can ask EDIT to start adding text at a particular line number. For example, assume you want to insert lines into your file and your current file is:

```
1     Peter, Peter
2     Pumpkin eater
3     Put her in a pumpkin shell
4     And there he kept her very well.
```

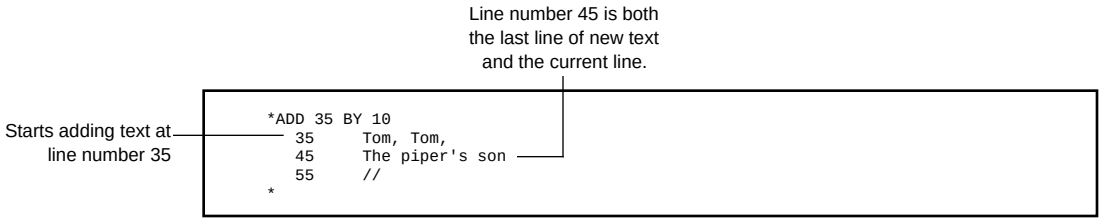
The command:

```
*ADD 2
    2.1    Had a wife
    2.2    But couldn't keep her
    2.3    //
*
```

causes EDIT to begin inserting text after line 2. EDIT begins adding new text at line number 2.1, using a decimal numbering of .1, and continues to use this numbering increment with each new line you add.

When you finish adding text, tell EDIT you are done by pressing RETURN, typing a double slash (/ /) in the first two columns, then pressing RETURN again. EDIT returns a prompt and awaits further commands.

EDIT uses numbering increments from .001 to 10 to add lines. Study the following examples.



The numbering of the previous ADD command uses whole numbers, and the last line of new text is line number 45. Therefore, the command:

```
*ADD
   55     Stole a pig
   65     And away he run.
   75     Tom, Tom,
   85     And far away."
   95     //
*
```

Line number 85 is both the last line of new text and the current line.

starts adding text at line 55.

You can use a decimal line number to add lines between existing lines of text. The command:

```
*ADD 75.1
   75.1   The piper's son
   75.2   Are "Over the hills
   75.3   //
*
```

causes EDIT to use a decimal numbering system based on .1.

If, again, you want to insert text lines into the file at line 75.1, EDIT notes that text already exists there by returning that line number and its text. EDIT then provides you with the next available line:

```
*ADD 75.1
   75.1   The piper's son
   75.11  He learned to play
   75.12  When he was young;
   75.13  //
*
```

If you type another ADD without requesting a whole or fractional line number, EDIT begins adding text at the next available line number, using the previously established numbering method:

```
*ADD
   75.13  But all the tunes
   75.14  That he could play
   75.15  //
*
```

The current file is now:

```
35     Tom, Tom,
45     The piper's son
55     Stole a pig
65     And away he run.
75     Tom, Tom,
75.1   The piper's son
75.11  He learned to play
75.12  When he was young;
75.13  But all the tunes
75.14  That he could play
75.2   Are "Over the hills
85     And far away."
```

Suppose that your file begins on line 1, and you want to add text before line 1. You can use ADD 0 to insert text before the first line of your file:

```
*ADD 0
    0     Tom, Tom, The Piper's Son
    0.1   RETURN
    0.2   A Favorite Nursery Rhyme
    0.3   From the Mother Goose Collection
    0.4   RETURN
    0.5   RETURN
    0.6   //
*
```

## The LIST Command: Listing Lines in Your File

To list the lines in your file, use the LIST command. The LIST command allows you to:

- ☐ List a single line
- ☐ List a range of lines
- ☐ List lines containing a specific character string
- ☐ List all lines in a file

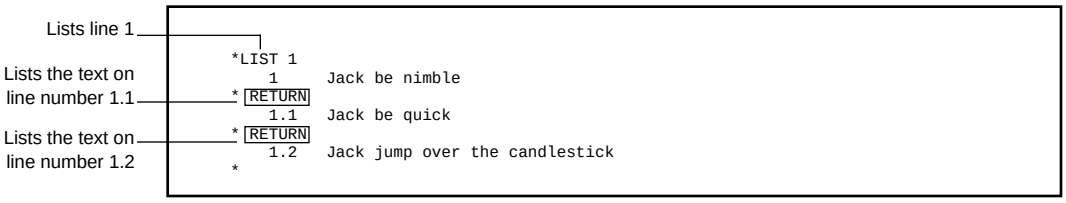
Suppose that your file contains these lines:

```
1     Jack be nimble
1.1   Jack be quick
1.2   Jack jump over the candlestick.
2     Jack and Jill
2.1   Went up a hill
3     To fetch a pail of water.
4     Jack fell down
5     And broke his crown
6     And Jill came tumbling after.
```

You can list a single line of your file by typing:

```
*LIST 1
   1     Jack be nimble
*
```

If you then press RETURN without typing a command at the prompt (*), EDIT lists the next line of text in your file. For example:



You can list a range of lines by line number. For example:

```
*LIST 1/3
    1     Jack be nimble
    1.1   Jack be quick
    1.2   Jack jump over the candlestick.
    2     Jack and Jill
    2.1   Went up a hill
    3     To fetch a pail of water.
*
```

You can list lines that contain a specific character string. For example:

```
*LIST "Jack"
    1     Jack be nimble
    1.1   Jack be quick
    1.2   Jack jump over the candlestick.
    2     Jack and Jill
    4     Jack fell down
*
```

You can list all the lines in the file. To do this, you must use the keyword ALL with the command so that EDIT knows to list your entire file. You type:

```
*LIST ALL
    1     Jack be nimble
    1.1   Jack be quick
    1.2   Jack jump over the candlestick.
    2     Jack and Jill
    2.1   Went up a hill
    3     To fetch a pail of water.
    4     Jack fell down
    5     And broke his crown
    6     And Jill came tumbling after.
*
```

## Specifying Ranges

When you specify a range of lines with a command, you must always give the range as lower/higher. EDIT searches the line numbers sequentially from the lowest line number to the highest; it isn't able to move randomly in the file. For example:

```
*LIST 17/13
          -- ERROR --
ALL RANGES MUST BE GIVEN AS LOWER/HIGHER
```

Simply retype the command, listing the range of lines in the correct order.

# 3 Revising Text in Your File

## Editing and Moving Your Text

You can use the commands described in this section to help you accomplish the following editing and formatting tasks:

- ☐ Deleting lines from your file
- ☐ Replacing lines in your file
- ☐ Fixing characters in one or more lines
- ☐ Changing characters in one or more lines
- ☐ Lengthening or shortening lines
- ☐ Breaking lines
- ☐ Moving one or more lines within your file
- ☐ Renumbering lines in your file
- ☐ Copying text from another file into your file or copying your files into a new file

Each of the following commands are also described in more detail in Section 4, “EDIT Command Summary.”

## The DELETE Command: Deleting Lines

You can use the DELETE command to:

- ☐ Delete a single line
- ☐ Delete consecutive lines
- ☐ Delete several lines
- ☐ Delete all lines

Suppose your file contains the following lines:

```
1     Jack be nimble
1.1   Jack be quick
1.2   Jack jump over the candlestick.
2     Jack and Jill
2.1   Went up a hill
3     To fetch a pail of water.
4     Jack fell down
5     And broke his crown
6     And Jill came tumbling after.
```

You can delete one line of text by typing:

```
*DELETE 3
    3     To fetch a pail of water.
*
```

EDIT displays the line that it deletes. Then, if you list your file, you can see that line 3 is gone.

You can delete several consecutive lines of text. For example, to delete the text starting at line number 2 and ending with line number 5, type:

```
*DELETE 2/5
    2     Jack and Jill
    2.1   Went up a hill
    4     Jack fell down
    5     And broke his crown
*
```

Lines 2 through 5 are removed from the file. You can review the remaining lines in the file by listing them:

```
*LIST ALL
    1     Jack be nimble
    1.1   Jack be quick
    1.2   Jack jump over the candlestick.
    6     And Jill came tumbling after.
*
```

You can delete several separate lines by listing them after the DELETE command. Separate them with a single space:

```
*DELETE 1 1.1 1.2
    1     Jack be nimble
    1.1   Jack be quick
    1.2   Jack jump over the candlestick.
*
```

You can delete all the lines in a file by using the ALL range parameter with the DELETE command. For example:

```
*DELETE ALL
    6     And Jill came tumbling after.
*
```

If you have 10 lines or less in your file, as in the previous example, EDIT will simply list the deleted lines. However, if there are over 10 lines, EDIT responds with a prompt, asking you permission to delete the lines. Suppose you have 25 lines in your file, and you want to delete 12 lines:

```
*DELETE 1/12
RANGE 1/12 CONTAINS 12 LINES.   SHALL I DELETE THEM?
```

If you type YES (or simply Y), EDIT deletes lines 1 through 12 and lists these lines as it deletes them (but you can use the QUIET keyword to tell EDIT not to list the deleted lines). If there are any remaining lines in your file—in this case, lines 13 through 25—the next line, line 13 here, becomes the current line. Any reply at the prompt other than YES (or yes, Y, or y) is taken by EDIT to mean NO. In that case, EDIT does not delete the lines, and the current line is the same as it was after the command previous to this DELETE command.

## The REPLACE Command: Replacing Lines of Text

The REPLACE command allows you to replace the text in a specified location of your file. You can:

- ☐ Replace a single line
- ☐ Replace consecutive lines

Suppose your current file contains these lines:

```
1     Simple Simon
2     She made some tarts
3     All on a summer's day;
4     Says Simple Simon to the pieman:
5     "Pray let me taste your ware."
6     Said the pieman to Simple Simon:
7     "Show me first your penny."
8     Says Simple Simon to the pieman:
9     "Indeed I have not any."
```

You can replace a single line of text using REPLACE. To replace line 1:

```
*REPLACE 1
    1     Simple Simon
    1     The Queen of Hearts
*
```

EDIT displays existing text on line 1, then prompts for new text with that line number. Type in the new text.

You can also replace consecutive lines of text by providing EDIT with a range of lines. EDIT displays the first line in the range and prompts you with that line number. Type in the new text and press RETURN ; EDIT will prompt you with the next line. Continue until you have replaced all the lines in the range.

If you type a double slant (/ /) in response to the line number prompt, EDIT asks the following question:

```
SHALL I DELETE THE REMAINING LINES?
```

If you type YES (or simply Y), EDIT deletes the current line and any remaining lines in the range. Any other response leaves the current and remaining lines unchanged. In either case, the REPLACE command terminates. For example:

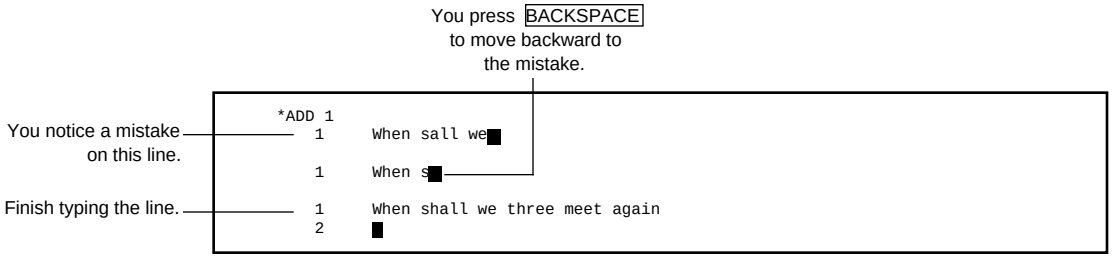
Using the LIST command to view the results of your text replacement

```
*REPLACE 4/9
    4     Says Simple Simon to the pieman:
    4     The Knave of Hearts
    5     "Pray let me taste your ware."
    5     He stole those tarts
    6     Says the pieman to Simple Simon:
    6     And took them clean away.
    7     "Show me first your penny."
    7     //
SHALL I DELETE THE REMAINING LINES?  y
    8     Says Simple Simon to the pieman:
    9     "Indeed I have not any."
*LIST ALL
    1     The Queen of Hearts
    2     She made some tarts
    3     All on a summer's day;
    4     The Knave of Hearts
    5     He stole those tarts
    6     And took them clean away.
*
```

## The FIX Command: Modifying Text Lines

As you add text, you may also add typing errors. You can correct mistyped text with or without using a command, depending on when you notice the error.

When you notice a mistyped word of text before you press RETURN, you can press BACKSPACE to back up to the mistake. Then, retype the line from there. For example:




When you mistype a word in a line of text and you do not notice the mistake until after you press RETURN, you must use a command (usually FIX) to correct the mistake.

### Features of the FIX Command

The FIX command, used to modify one or more text lines in your file, uses the three subcommands D, I, and R and recognizes the double slash (//) as a special character that stops a command.

Use the FIX command to:

- ☐ Delete characters in a line of text
- ☐ Insert a character string in a line of text
- ☐ Replace a character string in a line of text

### How to Edit With the FIX Command

When you give the FIX command with a line or a range of lines, EDIT displays the single line or the first line of text in the range. EDIT then prompts you with the FIX prompt (10 periods) and waits for you to edit. You do your editing after the prompt on the editing line—the line below the text line. Then, after you type your correction, press RETURN . EDIT displays the revised line of text and again prompts you to edit it. EDIT continues to display the revised line of text until you press RETURN alone at the FIX prompt. Then, if the range you specified has more than one line in it, EDIT displays the next line of text. When you finish correcting the last line of the specified range, EDIT returns you to the EDIT prompt.

### Using the Three FIX Subcommands

The FIX command uses these three subcommands when editing a text line.

- D (for delete) deletes the character in the text line that is above the D subcommand on the editing line. To correct a misspelling, for example, type:

```
*FIX 1
    1     Old King Cole was a merrry old soul
..........                          d
    1     Old King Cole was a merry old soul
..........(RETURN)
*
```

- I (for insert), followed by an insertion string, inserts the string into the text line. Insertion begins at the character above the I subcommand. To insert a word, type:

```
*FIX 2
    2     And a merry soul was he;
..........            iold
    2     And a merry old soul was he;
..........(RETURN)
*
```

- R (for replace), followed by a replacement string, replaces characters in the text line on a one-for-one basis with the characters in the replacement string on the editing line. Replacement begins with the character above the R subcommand. To replace a word, type:

```
*FIX 3
    3     He yelled for his pipe, and he called for his bowl,
.........   rcalled
    3     He called for his pipe, and he called for his bowl,
.........(RETURN)
*
```

EDIT considers any string of characters on the editing line that does not begin with the subcommand D, I, or R to be a replacement string. EDIT replaces the characters in the line of text (excepting the double slash) on a one-for-one basis. For example:

```
*FIX 4
    4     And he shouted at his jugglers seven.
.........       called for his fiddlers three.
    4     And he called for his fiddlers three.
.........(RETURN)
*
```

If you want to insert or replace a string of characters that begins with D, I, or R, you must use the relevant subcommand. Otherwise, EDIT mistakes the D, I, or R as a subcommand and acts accordingly on the rest of the string.

### Typing Several Subcommands on an Editing Line

If you have more than one correction to make to a text line, you can put more than one subcommand on an editing line. EDIT considers the first nonblank character on the editing line to be the beginning of a subcommand (if the character is an R, I, or D) or the beginning of a replacement string. EDIT also considers the first nonblank character that follows a double slash to be the beginning of a subcommand. Therefore, you can type several subcommands on one editing line by:

- Terminating an R or I subcommand with the double slash and then giving another subcommand or replacement string.
- Terminating a replacement string with the double slash and then giving another replacement string or a subcommand.
- Following the D subcommand with a replacement string or with another subcommand. You do not have to terminate the D subcommand with the double slash. EDIT treats the first character after a D subcommand as the beginning of another subcommand.

For more information and examples of typing several subcommands on an editing line, see Section 4, “FIX Command.”

### Terminating the FIX Command

You can terminate a FIX command before you’ve edited all the lines in a range by typing the double slash in columns 1 and 2 of the editing line and pressing RETURN The double slash causes EDIT to ignore any corrections you made to the current line you are editing and returns you to the EDIT prompt (*).

## The CHANGE Command: Modifying Text Lines

The CHANGE command is an editing command that allows you to make changes to specified character strings in your file. The following examples demonstrate how you can:

- ☐ Correct misspellings in one line or a range of lines
- ☐ Modify character strings in one line or a range of lines
- ☐ Combine CHANGE with the keywords ALL, BOTH, and WORD to further modify the changes you want to make

When you list the text in a file, sometimes you find misspelled words. To correct spelling and typing errors, you can use the CHANGE command.

Suppose you are working on the following file:

```
1      Jack be nimble
1.1    Jack be quick
1.2    Jack jumb over the candlestick.
2      Went up a hill
4      Jack fell down
5      And broke his crown
6      And Jill came tumbling after.
```

You mistyped the word *jump* when you added text at line number 1.2. To correct this mistake, type:

```
*CHANGE "jumb"jump" 1.2
    1.2   Jack jump over the candlestick.
*
```

Notice that the CHANGE command displays the changed line. If, for some reason, the CHANGE command cannot make the change you request (for instance, a syntactically incorrect CHANGE command or a change to text or file lines that do not exist), it returns the * prompt.

You can use the CHANGE command to modify text in one line or a range of lines. For example, to change the first occurrence of *be* to *be agile and* in line numbers 1 and 1.1:

```
*CHANGE "be"be agile and" 1 1.1
    1     Jack be agile and nimble
    1.1   Jack be agile and quick
*
```

To change *Jack* to *Peter* in every line in the file, use ALL as your range of lines with the CHANGE command line:

```
*CHANGE "Jack"Peter" ALL
    1     Peter be agile and nimble
    1.1   Peter be agile and quick
    1.2   Peter jump over the candlestick.
*
```

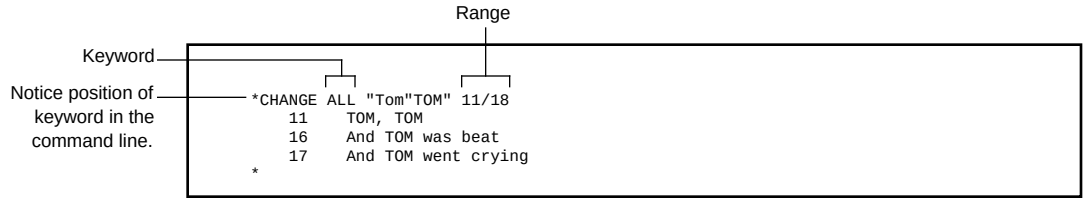
### Using Keywords With the CHANGE Command

You can use several keywords with the CHANGE command. One of these keywords is ALL (not to be confused with the range ALL). The keyword ALL tells EDIT to change all occurrences of a specified character string to a new string. Without ALL, EDIT changes just the first occurrence of a character string it finds in a line.

Your file contains these lines:

```
11    Tom, Tom
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tom was beat,
17    And Tom went crying
18    Down the street.
19    Tom, Tom
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

Use the keyword ALL here to change all occurrences of *Tom* to *TOM* in the specified line range:




You can use the keyword BOTH if you want EDIT to change both uppercase and lowercase (or mixed) occurrences of an existing character string to a new string. If you omit BOTH, EDIT only changes the character string in the text that matches exactly what you type on the command line.

You can list BOTH with ALL to change all uppercase and lowercase forms of the existing character string on one or more lines in the file to a new character string.

If the current file is:

```
11    tom, TOM
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And tom was beat
17    And tom went crying
18    Down the street.
19    tom, TOM
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

```
*CHANGE ALL BOTH "tom"Thomas" 11 16 17 19
    11    Thomas, Thomas
    16    And Thomas was beat
    17    And Thomas went crying
    19    Thomas, Thomas
*
```

EDIT searches lines 11, 16, 17, and 19 and changes every uppercase and lowercase *tom* it finds to *Thomas.*

When you use the keyword WORD, EDIT changes only those occurrences of a character string that constitute a word. To EDIT, a word is (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line.

If you omit WORD, EDIT changes all occurrences of a character string to a new string. WORD is useful when the string to be changed might also be imbedded in other words— for example, *and* appears inside *candlestick* and *command.* If the current file is:

```
11    Thomas, Tom
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tomwas beat,
17    And Tomwent crying
18    Down the street.
19    Thomas, Tom
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then with the following command:

```
*CHANGE WORD BOTH ALL "tom"TOM" ALL
    11    Thomas, TOM
    19    Thomas, TOM
*
```

EDIT searches the entire file and changes every uppercase and lowercase word *tom* it finds to *TOM*.

Omitting the keyword WORD in this case produces the following results:

Every uppercase and lowercase combination of the string *tom* is changed.

```
*CHANGE BOTH ALL "tom"TOM" ALL
    11    Thomas, TOM
    16    And TOMwas beat,
    17    And TOMwent crying
    19    Thomas, TOM
*
```

## The JOIN Command: Lengthening and Shortening Lines

EDIT enables you to lengthen or shorten lines of text in your file. The JOIN command lengthens lines to a certain line width by adding one or more words from the next line to your current line. To shorten lines to a certain line width, JOIN breaks off one or more words and adds them to the next line.

You can instruct EDIT to join your text to your exact specifications. Use the JOIN command to:

- ☐ Adjust the width of two or more lines of text to an automatic width of 70 characters
- ☐ Adjust the width of two or more lines of text to a width that you specify between 1 and 255 characters

First, review your file by listing all the lines:

```
*LIST ALL
    1     Old King Cole
    1.01  was a merry old soul
    1.1   And a merry old soul was he;
    1.2   He called for his pipe, and he called for his bowl,
    2     And he called for his fiddlers three.
    3     Every fiddler, he had a fine fiddle
    4     And a very fine fiddle had he;
    5     Oh there's none so rare as can compare
    6     With King Cole and his fiddlers three.
*
```

You then use the JOIN command to lengthen or shorten the lines in your file. If you don't specify a join width, EDIT uses 70 as the width for joining the lines. (See the description of the SET command in Section 4 for more information on JOIN and line widths.) For example:

Joining didn't exceed 70.

```
*JOIN 1/1.01
    1     Old King Cole was a merry old soul
*
```

Or, to control the maximum width of your lines, you can specify the line width when you use the JOIN command. If you specify a line width, you must also use the keyword WIDTH in the command line.

To lengthen the first three lines to a maximum width of 55, use the following command line (note the keyword WIDTH):

```
*JOIN 1/1.2 WIDTH 55
    1     Old King Cole was a merry old soul And a merry old soul
    1.1   was he;  He called for his pipe, and he called for his
    1.2   bowl,
*
```

To shorten lines 2 through 6 to a maximum line width of 25, type:

```
*JOIN 2/6 WIDTH 25
    2     And he called for his
    2.1   fiddlers three.  Every
    3     fiddler, he had a fine
    3.1   fiddle  And a very fine
    4     fiddle had he;  Oh
    5     there's none so rare as
    5.1   can compare  With King
    6     Cole and his fiddlers
    7     three.
*
```

However, you may not know the exact column number that you want to use as your new column width. In this case, use LIST with the keyword COL to create a column number template for a line of your file:

```
*LIST COL 1.2
          ....+....1....+....2....+....3....+....4....+....5.
    1.2    He called for his pipe, and he called for his bowl,
*
```

The numerals 1, 2, 3, 4, and 5 indicate columns 10, 20, 30, 40, and 50, respectively. The plus (+) signs occur at columns 5, 15, 25, and so on, and the periods occur at the other column positions.

## The BREAK Command: Breaking Lines

When you need to break a line of text in a particular place, use the BREAK command. The BREAK command enables you to:

- ☐ Break one line of text into two or more lines by marking the breaks with a character
- ☐ Break one line of text into two or more lines by specifying the character strings where you want breaks to occur
- ☐ Break one line of text into two or more lines by specifying column numbers where you want breaks to occur

Suppose your file now contains these lines:

```
1     Old King Cole was a merry old soul And a merry old soul
1.1   was he;  He called for his pipe, and he called for his
1.2   bowl,
2     And he called for his
2.1   fiddlers three.  Every
3     fiddler, he had a fine
3.1   fiddle  And a very fine
4     fiddle had he;  Oh
5     there's none so rare as
5.1   can compare  With King
6     Cole and his fiddlers
7     three.
```

When you give the BREAK command, you must specify the line you want broken. Suppose you want to break line 1:

```
*BREAK 1
```

EDIT displays line number 1 and waits for you to mark the breaks:

```
Old King Cole was a merry old soul And a merry old soul
```

You can use any nonblank character to mark breaks. Place the character under each letter where you want to start a new line. For example, notice how typing a “z” in three places in this line causes the line to break into three lines:

```
Old King Cole was a merry old soul And a merry old soul
z             z                   z
    1     Old King Cole
    1.01  was a merry old soul
    1.02  And a merry old soul
*
```

You can break a line by using the keyword AT and specifying character strings in the command line. When you supply EDIT with this kind of BREAK command, you can skip the step of marking the breaks under a displayed line. EDIT goes ahead and breaks the line where you’ve indicated. For example:

```
*BREAK 2 AT "he" "for"
    2     And
    2.01  he called
    2.02  for his
*
```

You can also break a line by specifying a column number with the keyword AT. Use LIST COL to obtain a column number template, then use the BREAK command to complete your change:

```
*LIST COL 1.1
          ....+....1....+....2....+....3....+....4....+....5....
    1.1   was he;  He called for his pipe, and he called for his
*BREAK 1.1 AT 10 34
  ┌ 1.1   was he;
  │ 1.11  He called for his pipe,
  └ 1.12  and he called for his
*
```

Breaks line 1.1 into three lines

You can specify the column numbers in any order:

```
*BREAK 1.1 AT 34 10
    1.1   was he;
    1.11  He called for his pipe,
    1.12  and he called for his
*
```

Use the LIST command to view the new shape of your file:

```
*LIST ALL
    1     Old King Cole
    1.01  was a merry old soul
    1.02  And a merry old soul
    1.1   was he;
    1.11  He called for his pipe,
    1.12  and he called for his
    1.2   bowl,
    2     And
    2.01  he called
    2.02  for his
    2.1   fiddlers three.  Every
    3     fiddler, he had a fine
    3.1   fiddle  And a very fine
    4     fiddle had he;  Oh
    5     there's none so rare as
    5.1   can compare  With King
    6     Cole and his fiddlers
    7     three.
*
```

## The MOVE Command: Moving Lines of Text in Your File

The MOVE command allows you to rearrange the text in your file. Use the MOVE command to:

- ☐ Reorganize text by moving one or more lines to one or more new locations within the existing text in a file
- ☐ Move lines out beyond the last line of text in a file
- ☐ Copy one or more lines from one location to one or more other locations in a file

When you move lines, EDIT deletes the lines from their original location in the file. (If you want to move lines elsewhere and still retain them in their original location in the file, see the MOVE COPY discussion later in this section.)

Suppose your file contains these garbled lines:

```
*LIST ALL
    1     Jack and Jill
    2     To fetch a pail
    3     of water.
    4     went up a hill
    5     One shoe on and
    6     one shoe off,
    7     And Jill came
    8     tumbling after.
    9     Jack fell down and
   10     broke his crown
*
```

Now you want to make your rhyme read properly. First, you can reorganize it by moving line 4 to its traditional position, which follows line 1. You cannot move text to a line number that already contains text. Since there is text on line 2, tell EDIT to add line 4 to an available line number between lines 1 and 2:

```
*MOVE 4 TO 1.1
    1.1   went up a hill
*=
```

You can then move lines 7 and 8 to the end of the file. You can use the particular line number (in this case, line 11) or, if you don't know what it is, you can use the range LAST:

```
*MOVE 7/8 TO LAST
   11     And Jill came
   12     tumbling after.
*
```

Lines 5 and 6 belong in a different poem. You can move them out of your text to a new location by telling EDIT to locate them elsewhere in the file:

```
*MOVE 5/6 TO 50
   50     One shoe on and
   51     one shoe off,
*
```

If you don't want to keep the lines at all, you can simply use the DELETE command.

Perhaps you want some text to appear in several places of your file. You can use MOVE with the keyword COPY to copy lines of text from one location in the file to other locations in the same file. (When you copy lines, the lines you are copying remain in their original location in the file.)

To copy line number 1 to line numbers 15, 20, 25, and 30, type:

```
*MOVE COPY 1 TO 15, 20, 25, 30
   15     Jack and Jill
   20     Jack and Jill
   25     Jack and Jill
   30     Jack and Jill
*
```

You can also copy several consecutive lines with the MOVE COPY command:

```
*MOVE COPY 1.1/3 to 16
   16     went up a hill
   17     To fetch a pail
   18     of water.
*
```

Use the LIST command to review the file:

```
*LIST ALL
    1     Jack and Jill
    1.1   went up a hill
    2     To fetch a pail
    3     of water.
    9     Jack fell down and
   10     broke his crown
   11     and Jill came
   12     tumbling after.
   15     Jack and Jill
   16     went up a hill
   17     To fetch a pail
   18     of water.
   20     Jack and Jill
   25     Jack and Jill
   30     Jack and Jill
   50     One shoe on and
   51     one shoe off,
*
```

## The NUMBER Command: Renumbering Lines

EDIT does not automatically renumber lines as you work in your EDIT file. EDIT silently assigns sequential line numbers to the lines in your file as you edit, using decimal line numbers when whole numbers are not available.

You can use the NUMBER command to:

- ☐ Renumber your entire file
- ☐ Renumber a portion of your file

You might want to renumber an entire file when you've done a great deal of editing and the line numbers become difficult to follow. Or you might need to renumber a portion of a file to make room for new text.

Suppose your file contains these lines:

```
 1     Jack and Jill
 1.1   went up a hill
 2     To fetch a pail
 3     of water.
 9     Jack fell down and
10     broke his crown
11     And Jill came
12     tumbling after.
15     Jack and Jill
16     went up a hill
17     To fetch a pail
18     of water.
20     Jack and Jill
25     Jack and Jill
30     Jack and Jill
50     One shoe on and
51     one shoe off,
```

Renumber the entire file by using the range ALL with the command:

```
*NUMBER ALL
*
```

The NUMBER command renumbers the lines of your file, then returns the asterisk prompt. It does not automatically list the renumbered text each time you give a NUMBER command. If you want to list the text, you must use the LIST command:

```
*LIST ALL
    1     Jack and Jill
    2     went up a hill
    3     To fetch a pail
    4     of water.
    5     Jack fell down and
    6     broke his crown
    7     And Jill came
    8     tumbling after.
    9     Jack and Jill
   10     went up a hill
   11     To fetch a pail
   12     of water.
   13     Jack and Jill
   14     Jack and Jill
   15     Jack and Jill
   16     One shoe on and
   17     one shoe off,
*
```

Notice that the NUMBER command reassigned line numbers. It did not change the order in which the lines occur.

When you are adding a block of text into an existing file, you might need to create a gap in the numbering sequence to accommodate adding the new text. To renumber lines 8 through 17 and start the new line numbers at line number 50, type:

```
*NUMBER 8/17 TO 50
*LIST ALL
    1     Jack and Jill
    2     went up a hill
    3     To fetch a pail
    4     of water.
    5     Jack fell down and
    6     broke his crown
    7     And Jill came
   50     tumbling after.
   51     Jack and Jill
   52     went up a hill
   53     To fetch a pail
   54     of water.
   55     Jack and Jill
   56     Jack and Jill
   57     Jack and Jill
   58     One shoe on and
   59     one shoe off,
*
```

## Renumbering to Add More Lines

Line numbers in your file range from 0 to 99999.999. When you do a great deal of editing on a file, you may find yourself with a series of line numbers like the following:

```
24.005 I wish the bald eagle had not been chosen as the
24.006 representative of our country;
24.007 like those among men who live by sharping
24.008 and robbing, he is generally poor, and
24.009 often very lousy.
24.010 The turkey is a much more respectable bird,
24.011 and withal a true original native of America.
```

If you want to add text between line numbers 24.006 and 24.007, you cannot do it because you've exhausted EDIT's decimal numbering capability. There are no line numbers between 24.006 and 24.007. You simply renumber the file to solve this problem. You can renumber the entire file, or you can start renumbering from line number 24.007. If you renumber this file from 24.007 and start the new line numbers at 50 (the command is NUMBER 24.007 / 24.011 TO 50), the file looks like the following:

```
24.005 I wish the bald eagle had not been chosen as the
24.006 representative of our country;
50     like those among men who live by sharping
51     and robbing, he is generally poor, and
52     often very lousy.
53     The turkey is a much more respectable bird,
54     and withal a true original native of America.
```

You now have plenty of line numbers for new text between line number 24.006 and line number 50.

## The GET Command: Copying Another File to Your File

When you use the GET command, you have access to EDIT files other than the one you are in currently. Use the GET command to:

- ☐ Copy a portion of another file into your current file
- ☐ Copy another entire file into your current file

### Copying Part of a File Into Your File

Suppose that you have an EDIT file named CLIENTS which contains a list of names you need to add to a memo you are writing. To copy the text on line numbers 3 and 4 of CLIENTS into your current file (in this example, BIZMEMO) at line number 20, type:

```
*GET CLIENTS 3/4 TO 20
LAST NEW LINE IS 21 <- 4
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

**Note** At this point, if you forget to include TO and a line location, EDIT gets the specified lines of CLIENTS, closes the current file, and places the lines in a new file it creates. EDIT then prompts you to name the new file. You're no longer in BIZMEMO.

When you use the GET command to copy text into your file, it does not list the lines after it copies them. It does, however, tell you that:

- ☐ Line number 4 from CLIENTS is the last line added to BIZMEMO.
- ☐ Line number 4 from CLIENTS is now line number 21 in BIZMEMO.
- ☐ The current file is still $WORK.TUTOR.BIZMEMO.

List your current file from line number 12 to the end to see the results of your GET command:

```
*LIST 12/LAST
   12     Old King Cole
   13     The Queen of Hearts
   14     Old Mother Hubbard
   20     Little Miss Muffett
   21     Simple Simon
*
```

### Copying All of a File Into Your File

You can also copy the entire file of CLIENTS into your current file. To copy all six lines of CLIENTS into your current file at line 4, type:

```
*GET CLIENTS TO 4
LAST NEW LINE IS 4.6 <- 6
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

Remember that the keyword TO is important here also. Omitting TO and a line location in this case will cause EDIT to close BIZMEMO and make CLIENTS your current file.

Use the LIST command to list the lines you just copied. Note that line number 6 from CLIENTS is now line number 4.6 in BIZMEMO, and BIZMEMO is still the current file.

```
*LIST 4/5
    4     The Knave of Hearts
    4.1   Humpty Dumpty
    4.2   Three Men in a Tub
    4.3   Little Miss Muffett
    4.4   Simple Simon
    4.5   Wee Willie Winkie
    4.6   Little Boy Blue
    5     Peter, Peter, pumpkin eater
*
```

If you do forget the keyword TO and find yourself in a different file, simply use the GET command with the name of the file you were editing to make that file the current one again.

For example, suppose you are in the file named BIZMEMO, and you give the following command, which omits the TO keyword:

```
*GET CLIENTS
CURRENT FILE IS $WORK.TUTOR.CLIENTS
*
```

Instead of successfully adding the file CLIENTS to your current file (BIZMEMO), you've made CLIENTS your current file. Use the GET command to regain BIZMEMO as your current file:

```
*GET BIZMEMO
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

### Copying Text to the Beginning or End of Your File

You can use the GET command to add text to the very beginning of your file. Tell EDIT to renumber your current file (to ensure clear line numbering). Then tell EDIT to copy the text into your file at line .001.

If your current file is MOBY:

```
*NUMBER ALL
*GET ISHMAEL 55/66 TO .001
LAST NEW LINE IS .012 <- 66
CURRENT FILE IS $WORK.FICTION.MOBY
*
```

EDIT renumbers lines in MOBY, making the first line in the file line number 1, then copies lines 55 through 66 of ISHMAEL into MOBY, starting at line number .001.

You can use the keyword LAST to copy lines to the end of your file. Suppose you want to add lines from CLIENTS to the end of your current file, but you don't know the last line number. Simply use the keyword LAST:

Copies lines 5 and 6 of CLIENTS to the end of BIZMEMO

```
*GET CLIENTS 5/6 TO LAST
LAST NEW LINE IS 23 <- 6
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

### Renumbering to Accommodate Added Lines

Sometimes when you add lines from another file to your current file, you may not have enough available lines in your file for the text you want to add. The following example illustrates such a situation and the way to handle it.

If your current file named MARKET contains the lines:

```
10.991 To market, to market to buy a fat pig,
10.992 Home again, home again, jiggety-jig;
10.993
11     To market, to market to buy a fat hog,
11.1   Home again, home again, jiggety-jog;
11.2
11.3   To market, to market to buy a plum bun,
11.4   Home again, home again, market is done.
```

and the file named COLE contains the lines:

```
21     Old King Cole was a merry old soul,
22     And a merry old soul was he;
23     He called for his pipe, and he called for his bowl,
24     And he called for his fiddlers three.
25     Every fiddler, he had a fine fiddle,
26     And a very fine fiddle had he;
27     Oh, there's none so rare as can compare
28     With King Cole and his fiddlers three.
```

then the following command attempts to copy the entire contents of COLE into MARKET, starting at line number 10.994:

```
*GET COLE TO 10.994
                 ^ -- ERROR --
AN INCREMENT SMALLER THAN .001 WOULD BE NEEDED
ATTEMPTING TO ADD LINE 11
NEXT LINE TO BE ADDED FROM SOURCE FILE IS 27
*
```

There are not enough line numbers available in MARKET to copy all of COLE. So, EDIT copies as many lines from COLE as it can into MARKET and then displays a message telling you where it stopped copying. The message:

```
ATTEMPTING TO ADD LINE 11
NEXT LINE TO BE ADDED FROM SOURCE FILE IS 27
```

means EDIT tried to put the text from line number 27 of COLE on line number 11 of MARKET but found that line number 11 already had text on it. To fix this problem and complete the copy of COLE into MARKET, simply renumber MARKET from line number 11 to the end of the file. Then issue another GET command to copy lines 27 through the end of COLE into MARKET. For example:

Use LIST to see the results.

```
*NUMBER 11/LAST TO 100
*GET COLE 27/LAST TO 11
LAST NEW LINE IS 13 <- 28
CURRENT FILE IS $WORK.FICTION.MARKET
*LIST ALL
   10.991 To market, to market to buy a fat pig,
   10.992 Home again, home again, jiggety-jig;
   10.993
   10.994 Old King Cole was a merry old soul,
   10.995 And a merry old soul was he;
   10.996 He called for his pipe, and he called for his
   10.997 And he called for his fiddlers three.
   10.998 Every fiddler, he had a fine fiddle,
   10.999 And a very fine fiddle had he;
   11     Oh, there's none so rare as can compare
   12     With King Cole and his fiddlers three.
   100    To market, to market to buy a fat hog,
   101    Home again, home again, jiggety-jog;
   102
   103    To market, to market to buy a plum bun,
   104    Home again, home again, market is done.
*
```

Then, if you want, you can use the NUMBER command to renumber the line numbers of the file.

### Listing Your Files: The ?FILES Command

When using file names and GET commands, you might need to check the spelling of a file name, for example, or search for a particular file. Without leaving the EDIT program, you can use the ?FILES command to display all the files on your current subvolume (or another if specified). (You can also use the FILES command at the command interpreter prompt to do the same thing.)

Suppose you are currently in the file $WORK.FICTION.MOBY. You can type ?FILES at the asterisk prompt to view all the files on your FICTION subvolume:

The file names are listed alphabetically.

```
*?FILES
ARTHUR    ISHMAEL   MALAPROP  MOBY      OLIVER    SHERLOCK
*
```

For more information on subvolume names and file names, see Section 6, "How EDIT Files Are Named."

### The QUERY Command

If you are still in your file, yet you need to know the name of the file (for locating yourself, for example), you can use the QUERY command. QUERY followed by the keyword NAME tells EDIT to display the full name of the current file:

```
*QUERY NAME
FILE $WORK.FICTION.MOBY
```

Using the QUERY command with no keyword provides the full name of the current file as well as additional information about the set options, size, and disk space being used by that file.

For more information regarding ?FILES, QUERY, and other related commands, refer to Section 4.

## The PUT Command: Copying Your File Into a New File

You can use the PUT command to copy part or all of your current file into another, new file. The PUT command creates the new file with a name you specify. The name of a file can be from one to eight alphanumeric characters, the first of which must be alphabetic, and the file name can be partially or fully qualified. (For more information on file names, refer to Section 6.)

### Copying Part of Your File

Suppose your current file, BIZMEMO, has 14 lines. Use the PUT command to copy lines 9 through 14 of BIZMEMO into a new file you have named as MYDATA:

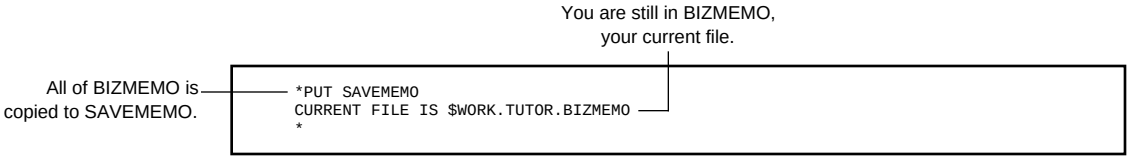
```
*PUT MYDATA 9/14
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

In the new file, the copied lines retain their original line numbers.

The PUT command only copies lines to another file. It does not delete the lines from your current file. If you then want to delete the copied lines from the current file, use the DELETE command.

### Copying All of Your File

You can use the PUT command to copy your entire file into a new EDIT file. When you are copying your entire file, you don't need to give a range of lines. Simply tell EDIT where you want a copy of your current file to go:



If you are using the PUT command to copy your current file into an already existing EDIT file, EDIT displays an error message:

Trying to put the current file into SAVEMEMO again

```
*PUT SAVEMEMO
          ^ -- ERROR --
THE NAMED FILE ALREADY EXISTS
*
```

You can either specify a different file name or add an exclamation point to your command line. By adding the ! character, you are telling EDIT to delete the existing file (if a file by that name already exists), to create a new file with the same name, and to copy the text into this new file:

```
*PUT SAVEMEMO !
CURRENT FILE IS $WORK.TUTOR.BIZMEMO
*
```

You might find the ! character useful when you want to replace an existing file with an updated or edited version of that file, instead of keeping both versions.

# 4 EDIT Command Summary

## Running the EDIT Program

To run the EDIT program, you type EDIT at the command interpreter prompt character. You can run the EDIT program in either interactive or noninteractive mode. Interactive mode requires you to respond at the EDIT prompt after EDIT executes each command. Noninteractive mode means that EDIT executes commands from a command file while allowing you to use the terminal in the meanwhile. The EDIT program can read from and write to disk files, nondisk devices, and processes (as illustrated in Figure 2).

### Running the EDIT Program Interactively

To run the EDIT program interactively, use the following EDIT command syntax:

```
EDIT [ filename ] [ ! ] [ [ ;editorcommand ] ... ]
```

`filename`

is the name of an existing EDIT file or is the name of a new EDIT file that you want to create. You can give a full file name or a partial file name. (See "How EDIT Files Are Named" in Section 6 for details.) If `filename` names a new file, the EDIT program asks you if it should create the file. If you do not give a `filename`, the EDIT program prompts you for one. (See "Adding Text to a New EDIT File" of the ADD command, later in this section.)

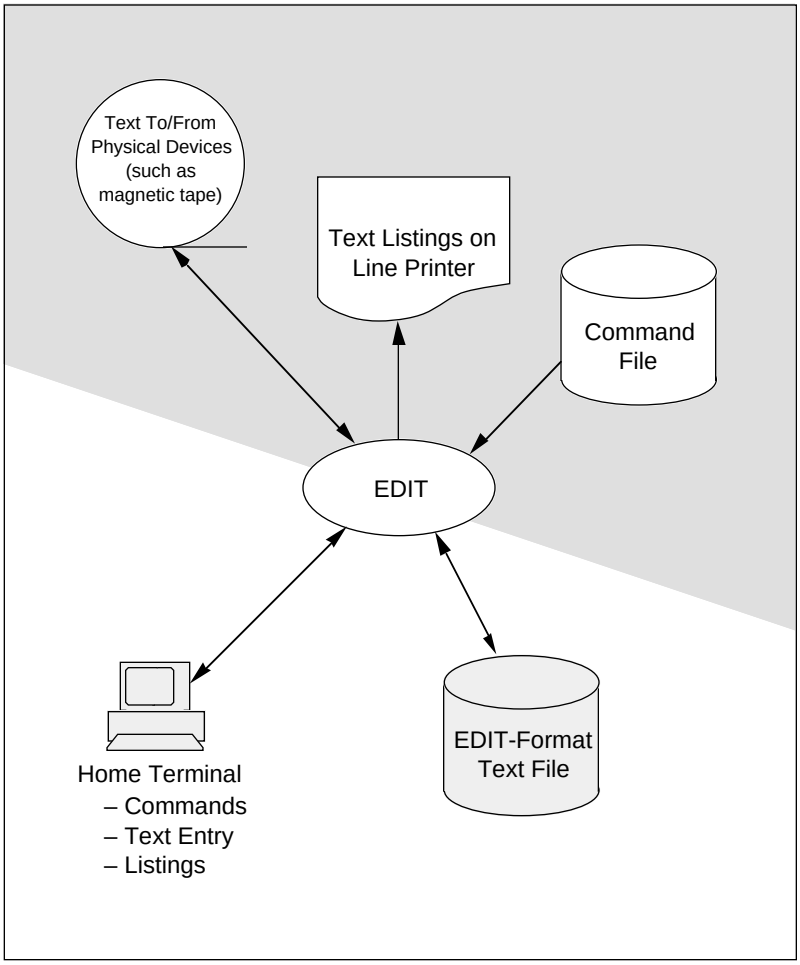
`!`

tells the EDIT program to create an EDIT file named `filename`, if `filename` does not already exist.

`editorcommand`

is an editor command.

**Figure 4-1. Running the EDIT Program**



### Example

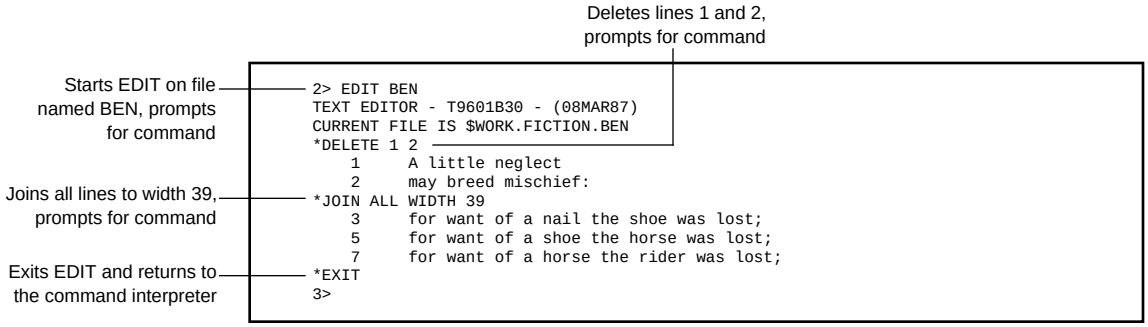
Suppose an EDIT file named BEN contains the following lines:

```
1     A little neglect
2     may breed mischief:
3     for want of a nail
4     the shoe was lost;
5     for want of a shoe
6     the horse was lost;
7     for want of a horse
8     the rider was lost.
```

You can edit this file by going through the EDIT program command execution cycle:

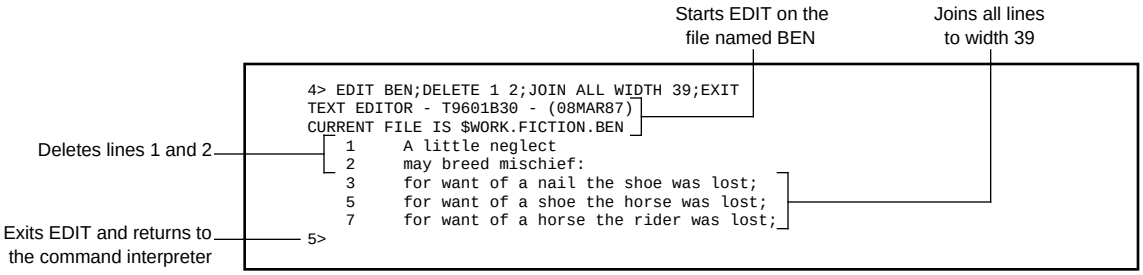
- ☐ Start the EDIT program
- ☐ Enter an editor command at the EDIT prompt
- ☐ Wait for the command to execute
- ☐ Continue to enter an editor command at the EDIT prompt until you exit from the EDIT program

For example:



### Typing Several Commands on One Line

As you become familiar with the individual commands of the EDIT program, you can learn to edit more efficiently by typing several commands in the same command line. You supply the commands all at once, and EDIT completes them in sequential order. When you do this, you must separate the individual commands with semicolons. For example:




**Note** When you include an OBEY command in an EDIT command line, the EDIT program executes all the other commands in the command line before executing the OBEY command. See the OBEY command description, later in this section, for details.

### Running the EDIT Program Noninteractively

When you run the EDIT program noninteractively, you free your terminal for other operations. Before you can use the EDIT program noninteractively, however, you must create a command file. (See Example 1, following.) This command file communicates with the EDIT program for you.

To run the EDIT program noninteractively, use the following EDIT command syntax:

```
EDIT / [IN filename1] [, OUT filename2] [, NOWAIT] /
```

`IN`

reads editor commands.

`filename1`

If you do not specify `filename1`, the EDIT program reads from the command file of the command interpreter (usually the home terminal).

`OUT`

specifies the file to which the EDIT program writes its output.

`filename2`

is the name of a nondisk device, process, or existing disk file. If you do not specify `filename2`, the EDIT program writes to the output file of the command interpreter (usually the home terminal).

`NOWAIT`

tells the command interpreter to prompt you for another command as soon as the EDIT program starts to run. If you omit NOWAIT, the command interpreter waits for the EDIT program to execute all EDIT commands in the command file before prompting you for another command.

For more information about NOWAIT and other command interpreter or TACL options, see the *GUARDIAN 90 Operating System Utilities Reference Manual*.

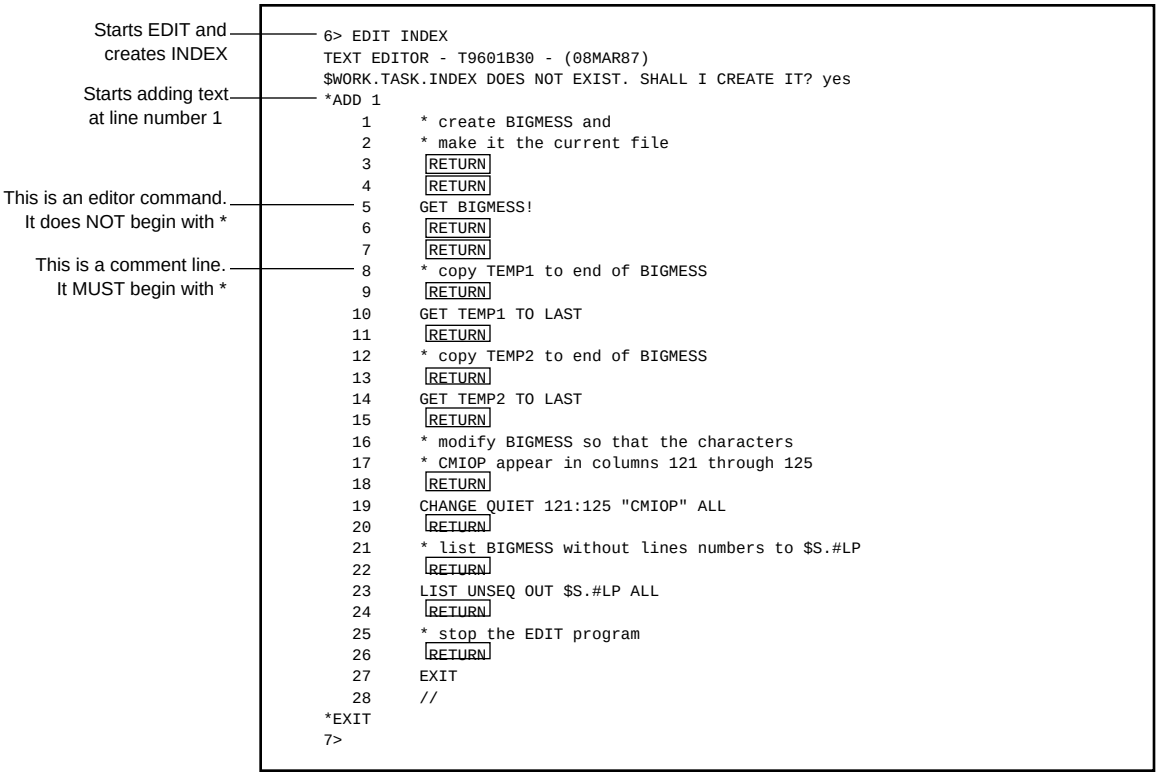
When you run the EDIT program noninteractively:

- The IN and OUT keywords accept a full file name or a partial file name. (See "How EDIT Files Are Named" in Section 6 for information on file names.)
- The EDIT program reads 132-byte records from the command file until the end-of-file is encountered. You can give multiple commands per record by separating the commands with a semicolon (;).
- When the output file is an unstructured disk file, each record is 132 characters (partial lines are blank-filled through column 132).

### Examples

1. A command file contains editor commands that will be executed by the EDIT program. A command file can also contain comment lines (any line that begins with an asterisk in the first column). When the EDIT program executes a command file, it ignores any comment line. (See the OBEY command description, later in this section, for more information on comment lines.)

You can use the EDIT program to create a command file as follows:




```
6> EDIT INDEX
TEXT EDITOR - T9601B30 - (08MAR87)
$WORK.TASK.INDEX DOES NOT EXIST. SHALL I CREATE IT? yes
*ADD 1
    1     * create BIGMESS and
    2     * make it the current file
    3      RETURN
    4      RETURN
    5     GET BIGMESS!
    6      RETURN
    7      RETURN
    8     * copy TEMP1 to end of BIGMESS
    9      RETURN
   10     GET TEMP1 TO LAST
   11      RETURN
   12     * copy TEMP2 to end of BIGMESS
   13      RETURN
   14     GET TEMP2 TO LAST
   15      RETURN
   16     * modify BIGMESS so that the characters
   17     * CMIOP appear in columns 121 through 125
   18      RETURN
   19     CHANGE QUIET 121:125 "CMIOP" ALL
   20      RETURN
   21     * list BIGMESS without lines numbers to $S.#LP
   22      RETURN
   23     LIST UNSEQ OUT $S.#LP ALL
   24      RETURN
   25     * stop the EDIT program
   26      RETURN
   27     EXIT
   28     //
*EXIT
7>
```

Now that you have created the command file named INDEX, run EDIT noninteractively by typing this command:

```
8 EDIT/IN INDEX, OUT $S.#LP, NOWAIT/
9
```

This command starts the EDIT program at the command interpreter and provides the name of the command file (in this case, INDEX). The prompt then reappears, at which you can type another command. Because EDIT returns the prompt, you can use your terminal while the EDIT program runs INDEX, which instructs EDIT to:

- ☐ Create BIGMESS and make it the current EDIT file
- ☐ Copy the contents of TEMP1 to the end of BIGMESS
- ☐ Copy the contents of TEMP2 to the end of BIGMESS
- ☐ Modify every line in BIGMESS so that it contains the characters CMIOP in columns 121 through 125
- ☐ List the entire contents of BIGMESS without line numbers to the process named $S.#LP (here, to a printer)
- ☐ Exit EDIT

2. The following command, using the noninteractive command syntax, tells EDIT to read editor commands and text from the device named $TAPE and write any output on the device named $PRINTER:

```
10 EDIT /IN $TAPE, OUT $PRINTER/
```

## The Keyword QUIET

The keyword QUIET combines with many of the commands available in EDIT and tells EDIT not to prompt you or list lines (as the case may be) as a command is executed.

The following summary illustrates the effect of QUIET on commonly used EDIT commands.

- ADD QUIET allows you to type new text lines without a line number prompt from EDIT. Remember, EDIT displays a line number prompt for each new line of text. An ADD command with QUIET might be:

```
*ADD QUIET 5 BY .1
```

- DELETE QUIET tells EDIT not to list lines as they are deleted. A DELETE command with QUIET might be:

```
*DELETE QUIET ALL
```

- CHANGE QUIET tells EDIT not to list lines as they are changed. A CHANGE command with QUIET might be:

```
*CHANGE QUIET WORD "Sam"SAM" ALL
```

- JOIN QUIET tells EDIT not to list the lines as they are joined. A JOIN command with QUIET might be:

```
*JOIN QUIET 1/5 WIDTH 55
```

- MOVE QUIET tells EDIT not to list lines as they are moved. MOVE and MOVE COPY commands with QUIET might be:

```
*MOVE QUIET 2/4 TO LAST
*MOVE QUIET COPY 2 TO 5, 10, 15
```

- REPLACE QUIET tells EDIT not to list the existing line of text prior to deleting it. A REPLACE command with QUIET might be:

```
*REPLACE QUIET 12/15
```

## Introduction to Ranges

When you type a command, you ask EDIT to "operate" on one or more lines of text in your file. So, typing a command actually involves typing both a command name and a range of text. A range is the line or lines of text on which you want the command to operate.

One of the most powerful features of EDIT is the variety of ways in which it allows you to specify ranges of lines or columns in your text. The simplest ways to indicate line ranges are with a line number (for a single line) or two line numbers separated by a slash (for two or more consecutive lines). You can also specify a range of lines that contains a particular character string (a character string is a series of characters such as a word, a phrase, or a number enclosed in quotation marks). The way to specify a column range is by indicating a single column or two column numbers separated by a colon (for two or more columns in the same line).

If you don't specify a range when you type a command that optionally takes one, EDIT assumes that you want to operate on the current line (that is, the last line displayed). The exception to this general rule concerns the ADD command: When you use an ADD command with no range, EDIT simply gives you the next available line. Table 1 summarizes the most common ways of indicating ranges. For a complete description of ranges, refer to Section 5.

**Table 4-1. Common Ranges**

| Range | How You Specify the Range | Command Example |
|---|---|---|
| A single line | Line number | LIST 10 |
| The first line | Keyword FIRST (F) | DELETE F |
| The last line | Keyword LAST (L) | FIX L |
| The current line | * | LIST * |
| Several separate lines | Line numbers separated by spaces | REPLACE 1 5 10 15 |
| | Beginning line and ending line | LIST 12.3/36 |
| Consecutive lines | Keyword FIRST and a line number | LIST F/5 10/22 |
| | Keyword LAST and a line number | MOVE 2/L to 50 |
| | * and a line number | JOIN 3/* WIDTH 7 |
| One or more lines containing a character string | Character string | BREAK AT “Jack” CHANGE WORD “is”was” ALL |
| One column | Column number | CHANGE 1 “ ” ALL |
| A group of columns | Rightmost and leftmost columns separated by : | LIST “he” COL 3:30 |
| Several separatecolumns | Column numbers separated by spaces | BREAK AT 5 20 35 |
| All lines | Keyword ALL (A) | LIST A |

## Editor Command Summary

The EDIT program recognizes 25 editor commands. These editor commands are summarized in Table 4-2.

**Table 4-2. Editor Command Summary** (Page 1 of 2)

| Command | Function |
|---|---|
| ADD | Enters text into an EDIT file |
| ADD BLOCK | Enters blocks of text into an EDIT file |
| BREAK | Separates a text line into two or more lines |
| CHANGE | Changes a string or a column to a new string |
| DELETE | Removes lines from an EDIT file |
| EXIT | Closes the current EDIT file and stops EDIT |
| FIX | Modifies text lines or editor commands |
| GET | Makes an EDIT file the current EDIT file |
| IMAGE | Modifies a string to a new string |
| JOIN | Formats lines to a specified width |
| LIST | Displays lines to the terminal or other location |
| MOVE | Moves lines from one location to another |
| NUMBER | Renumbers lines |
| OBEY | Executes a file of editor commands |
| PUT | Puts lines into a new text file |
| QUERY | Displays the current settings of EDIT control options |
| REPLACE | Replaces a line of a file with a new line |
| REPLACE BLOCK | Replaces blocks of text with new text |
| SET | Sets EDIT control options |
| TEDIT | Starts the TEDIT editing program |

**Table 4-2. Editor Command Summary** (Page 2 of 2)

| Command | Function |
|---|---|
| XEQ | Starts the EDIT VS editing program |
| ?ENV | Lists the current system name and volume name |
| ?FILES | Lists the names of the files in the default or a specified volume |
| ?SYSTEM | Sets the current system |
| ?VOLUME | Sets the current volume |

## ADD Command

The ADD command allows you to enter text from a terminal or from an OBEY file into a new or existing EDIT file. (See the OBEY editor command later in this section.)

### What to Enter

```
ADD [ QUIET ] [ line [ BY incr ] ]
```

`QUIET`

allows you to type new text lines without a line number prompt from the EDIT program. By default, the EDIT program displays a line number prompt for each new line of text.

`line`

is all the characters that have the same line number. EDIT uses this line number as the first line number at which to begin adding text.

`BY`

specifies the numbering increment for new line numbers.

`incr`

is a number from .001 through 10.

**How to Use ADD**

- When you issue an ADD command with no line number and no increment parameter and there is no previous ADD command (see Example 1, "Adding Text to an Existing File," and Examples 1 and 2, "Adding Text to a New EDIT File"), by default EDIT:
  - Starts adding text at a line number following the last line number in the EDIT file.
  - Numbers each new line of text by an increment of 1.
- When you issue an ADD command with a line number and an increment parameter (see Example 2, "Adding Text to an Existing File"), EDIT:
  - Starts adding text to the EDIT file at the line number you specify. If this line number already exists, EDIT starts adding text at the next available line number.
  - Numbers each new line of text according to the increment you specify.
- When you issue an ADD command with no line number and no increment parameter (see Example 3, "Adding Text to an Existing File"), by default EDIT:
  - Starts adding text at a line number following the last line number of the previous ADD command.
  - Numbers each new line of text according to the increment you specified in the previous ADD command.

- When you type an ADD command with a line number and no increment, EDIT chooses an increment (for example, 1, .1, .01, or .001) in order to insert text into your file. The increment is either of the following:
  - The same number of digits as the fraction portion of the line number you specified for a range parameter
  - Less than the difference between the line number to be added and the next existing line number in the file. See this point illustrated in Example 4, "Adding Text to an Existing File."
- At the line number prompt, you can terminate an ADD command in three different ways:
  - Type the / / character sequence (followed by a carriage return) in columns 1 and 2.
  - Press BREAK (or equivalent).
  - Type CTRL-Y (press CTRL and Y at the same time).

**Examples** **Adding Text to an Existing File**

1. If line number 1.6 is the last line of the EDIT file named JACK, the commands:

```
11 EDIT JACK
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.FICTION.JACK
*ADD
    2.6   Jack be nimble,
    3.6   Jack be quick,
    4.6   Jack jump
    5.6   Over the candlestick.
    6     //
*
```

starts the EDIT program. EDIT then starts adding text to the next available line after the last line in the file and numbers each new line by an increment of 1.

2. Suppose you have a file named TOM. The command:

The numbering increment is 1 by default; typing BY 1 here is optional.

```
*ADD 35 BY 1
   35     Tom, Tom,
   36     The piper's son
   37     Stole a pig
   38     //
*
```

starts adding text at line number 35. Line number 37 is the last line of new text as well as the current line.

3. The increment parameter of the previous ADD command is 1, and the last line of new text is line number 37. Therefore, the command:

```
*ADD
   38     And away he run.
   39     Tom, Tom,
   40     //
*
```

starts adding text at line number 38. Line number 39 is the last line of new text as well as the current line.

4. The command:

```
*ADD 40.01
   40.01  The piper's son
   40.02  He learned to play
   40.03  When he was young;
   40.04  //
*
```

causes the EDIT program to use a numbering increment of .01. Line number 40.03 is the last line of new text as well as the current line.

The current file named TOM now looks like this:

```
35     Tom, Tom,
36     The piper's son
37     Stole a pig
38     And away he run.
39     Tom, Tom,
40.01  The piper's son
40.02  He learned to play
40.03  When he was young;
```

The command:

```
*ADD 38
   38     And away he run.
   38.1   The pig was eat
   38.2   And Tom was beat,
   38.3   //
*
```

lists line number 38. Since line number 39 already exists, the EDIT program starts adding new text at line number 38.1 and uses a numbering increment of .1. Line number 38.2 is the last line of new text as well as the current line. 5.

The command:

```
*LIST LAST
   40.03  When he was young;
*
```

lists the last line in the file and makes line number 40.03 the current line. Line number 38.2 remains the last line of new text. (See the LIST command, later in this section, for more information about LIST.)

The command:

```
*ADD *
   41     When he was young;
   42     But all the tunes
   43     That he could play
   44     Was “Over the hills
   45     And far away."
   46     //
*
```

starts adding text after the current line (40.03).

6. The command:

```
*ADD QUIET 38.2
   38.2  And Tom was beat
   38.3  And Tom went crying
   38.4  Down the street.
   38.5  //
*LIST FIRST/40
   35     Tom,Tom
   36     The piper's son
   37     Stole a pig
   38     And away he run.
   38.1   The pig was eat
   38.2   And Tom was beat,
   38.3   And Tom went crying
   38.4   Down the street.
   39     Tom,Tom
*
```

lets you add new text following line number 38.2 and does not prompt you with line numbers. The EDIT program is numbering each new line of text, but it is not displaying the line numbers on the screen.

### Adding Text to a New EDIT File

1. To create and add text to a new EDIT file named POEMS, type:

Starts the EDIT program

Begins adding text at line number 1 and numbers subsequent lines by an increment of 1

```
12> EDIT POEMS
TEXT EDITOR - T9601B30 - (08MAR87)
$WORK.FICTION.POEMS DOES NOT EXIST. SHALL I CREATE IT? y
CURRENT FILE IS $WORK.FICTION.POEMS
*ADD
    1     Hey diddle diddle
    2     The cat and the fiddle
    3     The cow jumped
    4     Over the moon.
    5     //
*
```

2. If you invoke the EDIT program without an EDIT file name and issue an ADD command, the EDIT program prompts you to name the file. If the file name you type names a new EDIT file, the EDIT program creates an EDIT file with that name, begins adding new text to that file starting at line 1, and numbers each new line by an increment of 1. If the file name you type names an existing EDIT file, EDIT asks you if you want to purge it:

```
13 EDIT
TEXT EDITOR - T9601B30 - (08MAR87)
*ADD
     1     Hickory, dickory, dock
NAME THE NEW FILE: poems
SHALL I PURGE THE OLD FILE NAMED $WORK.FICTION.POEMS? y
     2     The mouse ran up the clock;
     3     //
*
```

If you invoke the EDIT program without an EDIT file name, issue an ADD command, and then do not name an EDIT file when the EDIT program prompts you for one, EDIT creates a temporary EDIT file, adds text to this temporary file starting at line 1, and numbers each new line by an increment of 1:

```
14 EDIT
TEXT EDITOR - T9601B30 - (08MAR87)
*ADD
    1     To market, to market
NAME THE NEW FILE:
*ADD
    ^ - WARNING
THE CURRENT FILE IS TEMPORARY
    2     to buy a fat pig,
    3     //
*
```

**Note** The GUARDIAN 90 operating system purges temporary EDIT files when the temporary file is closed. You close a file when the file is no longer the current file or when you exit from EDIT. If you decide you want to save the text in a temporary EDIT file, use the PUT editor command to put the text from the temporary file into a new EDIT file. For more information, see the discussion of the PUT editor command, later in this section.

## ADD BLOCK Command

The ADD BLOCK command allows you to mimic page mode editing from the EDIT program, which is designed primarily as a line editor. This command is useful only if you have a terminal with full-screen capabilities; see Appendix C, “Page Mode Editing.”

## BREAK Command

The BREAK command separates a text line into two or more text lines.

### What to Enter

```
BREAK line [ AT { column-num  ...   } ]
           [    { [ WORD ]          } ]
           [    { [ BOTH ] "string" } ]
```

`line`

is all the characters that have the same line number.

`AT`

specifies the columns or strings where you want to split the line.

`column-num`

is one or more numbers from 1 to 239.

`BOTH`

tells the EDIT program to break the specified line at the occurrence of `string`, regardless of whether `string` is uppercase or lowercase in the file. If you omit BOTH, the EDIT program breaks the specified line at the first occurrence of `string` that matches exactly what you type.

`WORD`

tells the EDIT program to break the specified line at only the occurrences of `string` that constitute a word. A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line. If you omit WORD, the EDIT program breaks the specified line at the first occurrence of `string`.

`string`

is one or more character strings enclosed in quotes. You can also enclose `string` within a pair of right slants (/) or single apostrophes (').

**How to Use BREAK**

- When you issue the BREAK command with no AT keyword, the EDIT program lists the line you specify in the range parameter then waits for you to mark where you want to break the line. To mark a break, type any nonblank character at the position where you want the break to occur. (See "BREAK Command With No AT Parameter," following.)
- When you use the AT keyword, you can specify more than one column or string variable. If you specify more than one column number, separate them with at least one blank space. (See "The AT Keyword and Column Numbers," following.) If you specify more than one character string, enclose each string in quotes and separate the strings with at least one blank space. EDIT breaks the line at the first character in the string. (See "The AT Keyword and Character Strings," following.)
- When specifying a string on the command line, you can qualify that string with the keywords BOTH or WORD; you can use one or both on one line. (See "The BOTH or WORD Keyword and Character Strings," following).
- In addition to quotation marks, you can enclose a string within a pair of right slants (/) or single apostrophes ('). Be certain to use the same enclosing characters throughout a command.
- When you use the AT keyword and several column numbers in the BREAK command line as locations for EDIT to break a line, the column numbers can be listed in any order. However, if you use the AT keyword and one or more strings, the order of the strings must reflect the order in which they appear in the text line.

**Examples** For the following examples, lines 45 and 46 of the current EDIT file are:

```
45     Jack be nimble, Jack be quick, Jack jump over the candlestick.
46     Jack and Jill went up the hill to fetch a pail of water,
```

### BREAK Command With No AT Parameter

1. The command:

```
*BREAK 45
```

lists line number 45:

```
Jack be nimble, Jack be quick, Jack jump over the candlestick.
```

and waits for you to mark the breaks. Type any nonblank character under each J to turn one line into three lines, each of which starts with the word Jack:

```
Jack be nimble, Jack be quick, Jack jump over the candlestick.
z               z              z
```

EDIT lists the three lines:

```
   45     Jack be nimble,
   45.1   Jack be quick,
   45.2   Jack jump over the candlestick.
*
```

2. Another example of the BREAK command with no AT parameter is:

```
*BREAK 46
Jack and Jill went up the hill to fetch a pail of water,
k             k               k
   46     Jack and Jill
   46.1   went up the hill
   46.2   to fetch a pail of water,
*
```

### The AT Keyword and Column Numbers

1. The commands:

```
*LIST COL 45
          ....+....1....+....2....+....3....+....4....+....5....+....6..
   45     Jack be nimble, Jack be quick, Jack jump over the candlestick.
*BREAK 45 AT 17 32
   45     Jack be nimble,
   45.1   Jack be quick,
   45.2   Jack jump over the candlestick.
*
```

lists line 45 with a column template above it and breaks line 45 into three lines.

2. Another example of the BREAK command with the AT keyword and column numbers is:

Lists line 46 with a column template above it

Breaks line 46 into 3 lines

```
*LIST COL 46
          ....+....1....+....2....+....3....+....4....+....5....+.
   46     Jack and Jill went up the hill to fetch a pail of water,
*BREAK 46 AT 32 15
   46     Jack and Jill
   46.1   went up the hill
   46.2   to fetch a pail of water,
*
```

### The AT Keyword and Character Strings

1. The command:

```
*BREAK 45 AT "Jack" "Jack" "Jack"
   45     Jack be nimble,
   45.1   Jack be quick,
   45.2   Jack jump over the candlestick.
*
```

   breaks line 45 into 3 lines.

2. Another example of the BREAK command with the AT keyword and character strings is:

Breaks line 46 into 3 lines

```
*BREAK 46 AT "to" "went"
   46     Jack and Jill
   46.1   went up the hill
   46.2   to fetch a pail of water,
*
```

### The BOTH or WORD Keywords and Character Strings

1. The command:

```
*BREAK 45 AT BOTH "O"
   45      Jack be nimble, Jack be quick, Jack jump
   45.1    over the candlestick.
*
```

   breaks line 45 at the first occurrence of string “O” (the “o” in “over”). The BOTH keyword tells EDIT that “O” can be either in uppercase or lowercase.

2. The command:

```
*BREAK 46 AT WORD "ill"
   46     Jack and Jill went up the hill to fetch a pail of water,
*
```

asks EDIT to break the line at the string “ill” only if “ill” is a word. Line 46 contains “ill” twice, but the string only occurs inside other words. EDIT, therefore, can’t find a string that matches exactly the string on the command line, so it returns line 46 unbroken.

## CHANGE Command

The CHANGE command has both of these capabilities:

- ☐ Change occurrences of an existing character string to a new character string in a specified range of lines
- ☐ Change designated columns to a character string in a specified range of lines.

### What to Enter

☐ The syntax of the CHANGE command to change an existing string to a new string is:

```
                 {[WORD]
CHANGE [ QUIET ] {[BOTH] "oldstring"[ newstring ]" }
                                      [range-specifier]
                 {[ALL ]                           }
```

☐ The syntax of the CHANGE command to change columns to a string is:

```
CHANGE [ QUIET ] { column-range-list "newstring" }
                                     [ range-specifier ]
```

`QUIET`

tells the EDIT program not to list lines as they are changed. If you omit QUIET, the EDIT program lists each line as it is changed.

`WORD`

tells the EDIT program to change all the occurrences of `oldstring` that constitute a word. A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line. If you omit WORD, the EDIT program changes all occurrences of `oldstring` to `newstring`. (See Example 5, following.)

BOTH

tells the EDIT program to change both uppercase and lowercase occurrences of `oldstring` to `newstring`. If you omit BOTH, the EDIT program changes only the occurrences of `oldstring` that exactly match what you type. (See Examples 3, 5, and 6, following.)

ALL

tells the EDIT program to change all occurrences of `oldstring` to `newstring` in a given line. If you omit ALL, the EDIT program changes only the first occurrence of `oldstring` in a line. (See Examples 1, 3, 4, 5, and 6, following.)

`oldstring`

is a character string that exists in your EDIT file. You can enclose `oldstring` within two right slants (/) or single apostrophes (') as well as within quotation marks.

`newstring`

is the character string that replaces `oldstring` or is the character string that replaces the existing text in the specified columns. You can enclose `newstring` within two right slants (/) or single apostrophes (') as well as within quotation marks.

`column-range-list`

references a single column or one or more groups of columns, delimited by pairs of column numbers, in one line of an EDIT file. Turn to "Column-Range-List Parameter" in Section 5 for a full explanation of this range.

`range-specifier`

indicates combinations of line-range and string-range parameters. Turn to "Range-Specifier Parameter" in Section 5 for a full explanation of this range.

**How to Use CHANGE**

- The quotes (“) surrounding `oldstring` and `newstring` are string-field separators. EDIT also accepts an apostrophe (’) and a slash (/) as a string-field separator. You must, however, use the same separator character throughout a CHANGE command.
- When you give a left column number and a right column number in a column-range-list parameter, the right column number must be equal to or greater than the left column number (for example, 5:5 is valid, FIRST:LAST is valid, 15:7 is not valid). The `newstring` variable is a character string. (See “Changing Columns to a String,” Examples 1 and 2, following.)
- You can specify columns 1 through 239.

**Examples** **Changing an Existing String to a New String**

1. If your current EDIT file is:

```
11    Tom, Tom
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tom was beat,
17    And Tom went crying
18    Down the street.
19    Tom, Tom
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

Line-range element of the range-specifier parameter

```
*CHANGE ALL "Tom"TOM" ALL
   11    TOM, TOM
   16    And TOM was beat
   17    And TOM went crying
   19    TOM, TOM
*
```

EDIT searches all lines and changes all occurrences of *Tom* that it finds on a line to *TOM*.

2. If line number 19 is the current line and the current file is:

```
11    TOM, TOM
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And TOM was beat
17    And TOM went crying
18    Down the street.
19    TOM, TOM
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

Line-range element of the range-specifier parameter

```
*CHANGE "TOM"tom" 11/*
   11    tom, TOM
   16    And tom was beat
   17    And tom went crying
   19    tom, TOM
*
```

Line-range element of the range-specifier parameter EDIT searches lines 11 through 19 and changes the first occurrence of *TOM* that it finds on a line to *tom*.

3. If line number 19 is the current line and the current file is:

```
11    tom, TOM
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And tom was beat
17    And tom went crying
18    Down the street.
19    tom, TOM
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

Line-range-list element of the range-specifier parameter

```
*CHANGE BOTH ALL "tom"Thomas" 11 16 17 19
   11    Thomas, Thomas
   16    And Thomas was beat
   17    And Thomas went crying
   19    Thomas, Thomas
*
```

EDIT searches lines 11, 16, 17, and 19 and changes every uppercase and lowercase *tom* that it finds on a line to *Thomas*.

4. If line number 19 is the current line and the current file is:

```
11    Thomas, Thomas
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Thomas was beat
17    And Thomas went crying
18    Down the street.
19    Thomas, Thomas
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

String-range element of the range-specifier parameter

```
                                  ┌──────────┐
*CHANGE "Thomas "Tom" "Thomas" NUM 1/9
   11    Thomas, Tom
   16    And Tomwas beat,
   17    And Tomwent crying
   19    Thomas, Tom
*
```

EDIT searches for the first through the ninth lines (regardless of actual line numbering) that contain the string *Thomas*, then changes every *Thomas(space)* that it finds on these lines to *Tom*. In this case, there were only four lines that contained the specified string, so all were changed. Also note that the string *Thomas,* (with the comma) did not change; EDIT changes only the strings that match exactly the string typed in the CHANGE command line.

5. If 19 is the current line number and the current file is:

```
11    Thomas, Tom
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tomwas beat,
17    And Tomwent crying
18    Down the street.
19    Thomas, Tom
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

Line-range element of the range-specifier parameter

```
*CHANGE WORD BOTH ALL "tom"TOM" ALL
   11    Thomas, TOM
   19    Thomas, TOM
*
```

EDIT searches the entire file and changes every uppercase and lowercase *tom* that it finds on a line to *TOM*.

6. If 19 is the current line and the current file is:

```
11    Thomas, TOM
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tomwas beat,
17    And Tomwent crying
18    Down the street.
19    Thomas, TOM
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

then:

String-range element of the range-specifier parameter

```
*CHANGE BOTH ALL "tom"TOMMY " WORD BOTH "and"
   16     And TOMMY was beat,
   17     And TOMMY went crying
*
```

EDIT searches the entire file for all the lines that contain uppercase and lowercase occurrences of the word *and*, then changes every *tom* that it finds on these lines to *TOMMY(space)*.

### Changing Columns to a String

1. If 19 is the current line number and current file is:

```
11    Thomas, TOM
12     The piper's son
13     Stole a pig
14     And away he run.
15     The pig was eat
16     And TOMMY was beat,
17     And TOMMY went crying
18     Down the street.
19     Thomas, TOM
20     The piper's son
21     He learned to play
22     When he was young;
23     But all the tunes
24     That he could play
25     Was "Over the hills
26     And far away."
```

then:

Two separate string-range elements of the range-specifier parameter (one does not modify the other)

```
*CHANGE 1:20 "Tom, Tom" "Thomas" NUM F, "Thomas" NUM F
   11     Tom, Tom
   19     Tom, Tom
*
```

EDIT searches the file for the first line that contains *Thomas* and changes columns 1 through 20 of that line to *Tom, Tom*. EDIT then searches the file again for the first line that contains *Thomas* and changes columns 1 through 20 of that line to *Tom, Tom*.

2. If 19 is the current line number and the current file is:

```
11     Tom, Tom
12     The piper's son
13     Stole a pig
14     And away he run.
15     The pig was eat
16     And TOMMY was beat,
17     And TOMMY went crying
18     Down the street.
19     Tom, Tom
20     The piper's son
21     He learned to play
22     When he was young;
23     But all the tunes
24     That he could play
25     Was "Over the hills
26     And far away."
```

then:

Line-range element of the range-specifier parameter

```
*CHANGE 1 "       " ALL
   11           Tom, Tom
   12           The piper's son
   13           Stole a pig
   14           And away he run.
   15           The pig was eat
   16           And TOMMY was beat,
   17           And TOMMY went crying
   18           Down the street.
   19           Tom, Tom
   20           The piper's son
   21           He learned to play
   22           When he was young;
   23           But all the tunes
   24           That he could play
   25           Was "Over the hills
   26           And far away."
*
```

EDIT adds seven blank spaces to the beginning of every line in the file.

## DELETE Command

The DELETE command removes a range of text lines from the current EDIT file.

### What to Enter

```
DELETE [ QUIET ] { range-specifier }
       [   !   ]
```

`QUIET`

tells the EDIT program not to list lines as they are deleted.

`!`

tells the EDIT program not to prompt you for permission to delete more than 10 lines. If you omit the !, for each block of more than 10 lines, the EDIT program prompts you for permission to delete the lines.

`range-specifier`

indicates combinations of line-range and string-range parameters. Turn to "Range-Specifier Parameter" in Section 5 for a full explanation of this range.

**Examples** 1. If the current file is:

```
11    Tom, Tom
12    The piper's son
13    Stole a pig
14    And away he run.
15    The pig was eat
16    And Tom was beat,
17    And Tom went crying
18    Down the street.
19    Tom, Tom
20    The piper's son
21    He learned to play
22    When he was young;
23    But all the tunes
24    That he could play
25    Was "Over the hills
26    And far away."
```

and you issue the command:


```
*DELETE 15/LAST
```

EDIT responds with:

```
RANGE 15/26 CONTAINS 12 LINES.  SHALL I DELETE THEM?
```

If you type Y or y or YES or yes, EDIT deletes lines 15 through 26, prints these lines as it deletes them, and makes line number 14 the current line. EDIT takes any other reply to mean NO, does not delete lines 15 through 26, and keeps the current line number the same as it was after the command previous to this DELETE command.

2. If the current EDIT file contains 2000 lines numbered from 1 to 2000 and you issue the command:


EDIT deletes lines 50 through 700 without asking you if it's okay to do so, prints these lines as it deletes them, and makes line number 701 the current line.

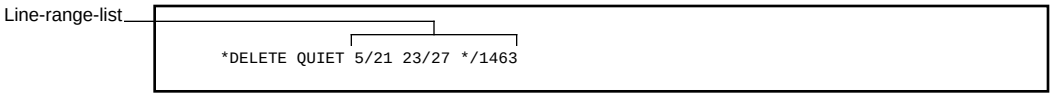
3. If the current file is:

```
1.5   Mary had a little lamb
1.8   Whose fleece was white as snow
2.1   And everywhere that Mary went
2.4   The lamb was sure to go.
2.7   He followed her to school one day
4     It was against the rule
4.3   It made the children laugh and play
4.6   To see a lamb in school.
```

and you issue the command:


EDIT deletes lines 2.1, 2.4, and 2.7, does not print the lines it deletes, and makes line number 4 the current line.

4. If the current EDIT file has 2000 lines numbered from 1 to 2000 and if 101 is the current line number, then the command:




prompts:

```
RANGE 5/21 CONTAINS 17 LINES.  SHALL I DELETE THEM?
```

If you type Y or y or YES or yes, EDIT deletes lines 5 through 21. (Line number 101 is the current line.) EDIT takes any other reply to mean NO and does not delete lines 5 through 21. (Line number 101 is the current line.) Then, EDIT deletes lines 23 through 27. (Line number 101 is the current line.) Finally, EDIT prompts:

```
RANGE 101/1463 CONTAINS 1242 LINES.  SHALL I DELETE THEM?
```

Once again, if you type Y or y or YES or yes, EDIT deletes lines 101 through 1463 and makes line number 1464 the current line. EDIT takes any other reply to mean NO, does not delete lines 101 through 1463, and makes line number 28 the current line.

## EXIT Command

The EXIT command terminates an editing session. Entering the EXIT command closes the current file (if any), deletes the process associated with the editing session, and returns control to the command interpreter.

### What to Enter

```
EXIT
```

### How to Use EXIT

Besides using the EXIT command, you can exit from the EDIT program by typing CTRL-Y (press (CTRL) and (Y) at the same time) at the asterisk prompt.

### Examples

1. The command:

```
*EXIT
18
```

   terminates the EDIT program and returns you to the command interpreter.

2. The character:

```
*CTRL-Y
*EOF!
19
```

   terminates the EDIT program and returns you to the command interpreter.

## FIX Command

The FIX command allows you to modify text lines interactively. The FIX command with the COMMAND keyword allows you to modify editor command lines interactively.

### What to Enter

```
FIX [ range-specifier ]
    [ COMMAND         ]
```

`range-specifier`

indicates combinations of line-range and string-range parameters. Turn to "Range-Specifier Parameter" in Section 5 for a full explanation of this range.

`COMMAND`

tells EDIT you want to modify the last editor command line. If you omit COMMAND, the FIX command operates on a line of text. (See "Fixing an Editor Command," following.)

### How to Use FIX

- If you do not give a range, the FIX command operates on the current line of text.
- When you issue a FIX command, you must interactively edit each line in the range. EDIT prints the first line of text in the range and waits for you to edit it. You do your editing on the editing line--the line below the text line:

```
    Jack be nimbble, Jack be quite
              d
```

EDIT prints the new text line and again waits for you to edit it:

```
Jack be nimble, Jack be quite
                           ck
```

EDIT continues to print the new text line until you press RETURN alone on a line:

```
Jack be nimble, Jack be quick
RETURN
```

Then EDIT prints the next text line in the range.

- The FIX command has three subcommands that you can use when editing a text line. They are:
  - D or d (for delete) deletes the character in the text line that is above the D subcommand on the editing line. See Example 1, following.
  - I or i (for insert) followed by an insertion string inserts the string following the I subcommand on the editing line into the text line. Insertion begins at the character that precedes the character above the I subcommand. See Example 2, following.
  - R or r (for replace) followed by a replacement string replaces characters in the text line with the characters in the replacement string on the editing line on a one-for-one basis. Replacement begins with the character above the R subcommand. See Example 3, following.
- EDIT treats as replacement strings any strings on the editing line that do not begin with the subcommand R, I, or D. Characters in replacement strings replace the characters in the text line on a one-for-one basis. See Example 4, following.

- EDIT treats the first nonblank character in the editing line as the beginning of a subcommand (if the character is an R, I, or D) or the beginning of a replacement string. EDIT also treats the first nonblank character that follows the / / character sequence as the beginning of a subcommand. You can put more than one subcommand on an editing line in three ways:
  - Terminating an R or I subcommand with the / / character sequence and then giving another subcommand or replacement string.
  - Terminating a replacement string with the / / character sequence and then giving another replacement string or a subcommand.
  - Following the D subcommand with a replacement string or with another subcommand. (You do not have to terminate the D subcommand with the / / character sequence. EDIT treats the first character after the D subcommand as the beginning of another subcommand.) See Example 5, following.
- You can terminate a FIX command before you have edited all the lines in the range in three ways:
  - Type the / / character sequence (followed by a carriage return) in columns 1 and 2 following the FIX command prompt
  - Press BREAK (or equivalent)
  - Type CTRL-Y (press CTRL and Y at the same time)

When you terminate a FIX command, the current line displayed by the FIX command is restored to its pre-FIX command state (that is, any FIX operations are ignored). See Example 6, following.

**Examples: Fixing a Text Line**

1. Deleting Characters

   The D subcommand deletes any character above it. If line 5.3 of the current file is:

```
Little Jack Horner sat in the corner
```

   and you want to change that line to:

```
Jack sat in the corner
```

   then use the command:

```
*FIX 5.3
    5.3   Little Jack Horner sat in the corner
.........ddddddd     ddddddd
    5.3   Jack sat in the corner
.........(RETURN)
*
```

   The D subcommand can be followed by another subcommand. See Example 5, following.

2. Inserting Characters

   To use the I subcommand, you type the letter I on the editing line directly under the character that the insertion is to precede. Immediately follow the I subcommand with the insertion string.

   If lines 3 and 4 of your current file are:

```
3    Mary Mary contrary
4    How do you grow?
```

you can change them to:

```
3    Mary Mary quite contrary
4    How does your garden grow?
```

with the command:

```
*FIX 3/4
    3     Mary Mary contrary
.........         i quite
    3     Mary Mary quite contrary
.........RETURN
    4     How do you grow?
.........      ies
    4     How does you grow?
.........            ir garden
    4     How does your garden grow?
.........RETURN
*
```

3. Replacing Characters

   To use the R subcommand, you type the letter R on the editing line underneath the first character in the text line to be replaced. Immediately follow the R subcommand with the replacement string.

   If the current file is:

```
1     Wee Willie Winkie runs through the town,
2     Upstairs and downstairs, in his nightgown;
3     Singing at the window, crying through the lock,
4     "Are the children in their beds?
5     Now it's eight o'clock."
```

   you can change line 3 to:

```
3     Rapping at the window, crying through the lock,
```

with the command:

```
*FIX 3
    3     Singing at the window, crying through the lock,
.........RRapping
    3     Rapping at the window, crying through the lock,
.......... RETURN
*
```

4. Replacing Characters

   If a string on the editing line does not begin with the R, I, or D subcommand, EDIT treats the string as a replacement string. The characters in the replacement string replace the characters in the text line on a one-for-one basis. If line 1 of the current file is:

```
1    Humpty Dumpty was a merry old soul,
```

you can change it to:

```
1    Old King Cole was a merry old soul,
```

with the command:

```
*FIX 1
    1     Humpty Dumpty was a merry old soul,
.........Old King Cole
    1     Old King Cole was a merry old soul,
.......... RETURN
*
```

However, to change the line:

```
3     Singing at the window, crying through the lock,
```

to:

```
3     Rapping at the window, crying through the lock,
```

you must use the R subcommand since the replacement string *Rapping* begins with an *R*. If you do not precede this replacement string with the R subcommand, the result is:

```
*FIX 3
    3     Singing at the window, crying through the lock,
.........Rapping
    3     appingg at the window, crying through the lock,
.........RETURN
*
```

Also, to change the line:

```
5     Who sat here beside her,
```

to:

```
5     Who sat down beside her,
```

you must use the R subcommand since the replacement string *down* begins with a *d*. If you do not precede this replacement string with the R subcommand, the result is:

```
*FIX 5
    5     Who sat here beside her,
.........        down
    5     Who sat own beside her,
.........RETURN
*
```

5. Typing Two or More Subcommands on an Editing Line

   When you want to give another subcommand on the same editing line as the I or R subcommand, use the character sequence / / to terminate the R or I subcommand. Then give the next subcommand. For example, if the current file is:

```
1     Georgie pudding and pastry pie,
2     Tickled the girls and made them shy;
3     When the bullies came out to play,
4     Georgie Porgie ran away.
```

   To change the line:

```
1     Georgie pudding and pastry pie,
```

   to:

```
1     Georgie Porgie, pudding and pie,
```

use the command:

```
*FIX 1
    1     Georgie pudding and pastry pie,
.........        iPorgie, // ddddddd
    1     Georgie Porgie, pudding and pie,
.........RETURN
*
```

When you want to give another subcommand on the same editing line as a replacement string, use the character sequence / / to terminate the replacement string. Then give the next subcommand. For example, to change the line:

```
2     Tickled the girls and made them shy,
```

to:

```
2     Pinched the girls and made them cry,
```

use the command:

```
*FIX 2
    2     Tickled the girls and made them shy,
.........Pinched//                       cry
    2     Pinched the girls and made them cry,
.........RETURN
*
```

When you want to give another subcommand on the same editing line as the D subcommand, you do not need to use the character sequence // to terminate the D subcommand. For example, to change the string:

```
3     When the bullies came out to play,
```

to:

```
3     When the boys came out to play,
```

use the command:

```
*FIX 3
    3     When the bullies came out to play,
..........          ddddddiboys
    3     When the boys came out to play,
..........RETURN
*
```

6. Terminating the FIX Command

If the file is:

```
1     Georgie pudding and pastry pie,
2     Tickled the girls and made them shy;
3     When the bullies came out to play,
4     Georgie Porgie ran away.
```

and you issue the following command and perform the following sequence of fixes:

```
*FIX 1/3
    1     Georgie pudding and pastry pie,
.........       i Porgie,//  ddddddd
    1     Georgie Porgie, pudding and pie,
.........[RETURN]
    2     Tickled the girls and made them shy;
.........Pinched
    2     Pinched the girls and made them shy;
.........                                 cry
    2     Pinched the girls and made them cry;
.........//
*
```

the / / character sequence in columns 1 and 2 terminates the FIX command and negates any fixes you made to the line. The current file is:

```
1     Georgie Porgie, pudding and pie,
2     Tickled the girls and made them shy;
3     When the bullies came out to play,
4     Georgie Porgie ran away.
```

### Fixing an Editor Command

You use the FIX COMMAND command to alter the last editor command line. You will find the FIX COMMAND form useful when you enter a long string of commands and one of them fails due to a typographical error. For example, the commands:

```
20 FILES
$SYSTEM.USER

MYPOEM     OLDPOEM

21 EDIT OLDPOEM
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $SYSTEM.USER.OLDPOEM
*GET MYPOM 1/10 TO LAST;LIST LAST-10/LAST
```

returns the error message:

```
??:011
FILE  $SYSTEM.USER.MYPOM  CANNOT BE OPENED
```

because EDIT cannot find a file named MYPOM.

The following command prints the most recent editor command line and waits for you to edit it:

```
*FIX COMMAND
(COMMAND) GET MYPOM 1/10 TO LAST;LIST LAST-10/LAST
..........
```

The R, I, and D subcommands work exactly the same when you're editing a command line as they do when you're editing a text line. When you're through editing, EDIT re-executes the command line. For example:

```
(COMMAND  GET MYPOM 1/10 TO LAST;LIST LAST-10/LAST
.........        ie
(COMMAND) GET MYPOEM 1/10 TO LAST;LIST LAST-10/LAST
.........RETURN
LAST NEW LINE IS 1457 <- 10
CURRENT FILE IS $SYSTEM.USER.OLDPOEM
 1454     Little Boy Blue,
 1455     Come blow your horn!
 1456     The sheep's in the meadow,
 1457     The cow's in the corn.
 1458     Where's the boy
 1459     That looks after the sheep?
 1460     He's under the haystack,
 1461     Fast asleep.
 1462     Will you wake him? No, not I;
 1463     For if I do, he'll be sure to cry.
*
```

EDIT does not execute the command line after a FIX COMMAND:

- ☐ If you type two right slants (followed by a carriage return) in columns 1 and 2 after the FIX COMMAND prompt
- ☐ If you press BREAK (or equivalent)
- ☐ If you type CTRL-Y (press CTRL and Y at the same time)

## GET Command

The GET command is a versatile command that can do all of the following:

- ☐ Specify an EDIT file and make it the current file for editing. (See "Making an EDIT File the Current File for Editing," following.)
- ☐ Select all or part of an existing EDIT file, create a new file containing the selected part, and make the new file the current file for editing. (See "Using an Existing File to Create a New Current File," following.)
- ☐ Specify all or part of an existing EDIT file for addition to the current file—with or without renumbering and with or without replacing duplicate lines (as indicated by their line numbers). (See "Adding All or Part of an EDIT File to the Current File," following.)
- ☐ Specify all or part of a text file on a nondisk device or in a non-EDIT-format disk file, create a new EDIT file containing the selected part, and make that the current file for editing. (See "Using GET With a Nondisk Device or Non-EDIT-Format File," following.)
- ☐ Specify all or part of a text file on a nondisk device or in a non-EDIT-format disk file for addition to the current file. (See "Adding Text From a Nondisk Device or Non-EDIT-Format Disk File," following.)

### What to Enter

☐ The syntax of the GET command to specify an EDIT file and make it the current file for editing is:

```
GET filename1 [ { READ } ]
              [ {  !   } ]
```

☐ The syntax of the GET command to accomplish the second through fifth items mentioned above is:

```
GET filename1 [ { line-range-list        } ]
              [ { NUM ordinal-range-list } ]

              [ { PUT filename2 [ ! ]      } ]
              [ { TO { line [ BY incr  ] } } ]
              [ {    { SAME [ KEEP ]     } } ]
```

`filename1`

is the name of an EDIT file, the name of a nondisk device, or the name of a non-EDIT-format disk file. The GET command accepts a full or a partial file name. (See "How EDIT Files Are Named" in Section 6 for information on file names.)

`READ`

tells the EDIT program to open `filename1` with read-only access. If you specify READ, the EDIT program rejects any command that attempts to write to `filename1` (for example, ADD or FIX). You cannot specify READ if you specify the ! character.

`!`

tells the EDIT program to get `filename1` and to create it if it doesn't exist. EDIT doesn't prompt you with a confirmation message. If you specify !, you cannot specify any of the other optional parameters.

`line-range-list`

references one or more contiguous lines or blocks of text in an EDIT file. Turn to "Line-Range-List Parameter" in Section 5 for a full explanation of this range.

NUM

tells the EDIT program to add the lines specified in the ordinal-range-list parameter from `filename1` to the current file. The NUM keyword always precedes the ordinal-range-list value.

`ordinal-range-list`

references the literal position of one or more lines in your file. Turn to "Ordinal-Range-List Parameter" in Section 5 for a full explanation of this range.

PUT

tells the EDIT program to copy text from `filename1` into a new file named filename2 and make `filename2` the current file. (See "Using an Existing File to Create a New Current File" and "Using GET With a Nondisk Device or Non-EDIT-Format File," following.)

`filename2`

is the name of an EDIT file.

TO

tells the EDIT program to add text from `filename1` to the current file. (See "Adding All or Part of an EDIT File to the Current File" and "Adding Text From a Nondisk Device or Non-EDIT-Format Disk File," following.)

`line`

is all the characters that have the same line number.

BY

specifies the numbering increment for lines added to the current file.

`incr`

is a number from .001 through 10.

SAME

tells the EDIT program not to renumber lines from `filename1` when they are added to the current file. (See "Adding All or Part of an EDIT File to the Current File," following.)

KEEP

tells the EDIT program to retain the lines in the current file that have the same line numbers as those in `filename1`. If you omit KEEP, the EDIT program replaces lines in the current file. (See "Adding All or Part of an EDIT File to the Current File," following.)

**Examples** **Making an EDIT File the Current File for Editing**

1. The command:

```
*GET POEMS
CURRENT FILE IS $WORK.FICTION.POEMS
*
```

closes the current file, if any, and makes the existing EDIT file named POEMS the current file.

2. The following command closes the current EDIT file, creates a new EDIT file named PARTY, and makes PARTY the current file:

```
*GET PARTY
$WORK.FICTION.PARTY DOES NOT EXIST. SHALL I CREATE IT? y
CURRENT FILE IS $WORK.FICTION.PARTY
*
```

### Using an Existing File to Create a New Current File

1. The command:

```
*GET AESOP PUT FABLE
CURRENT FILE IS $WORK.FICTION.FABLE
*
```

   copies the entire contents of the file AESOP into the file FABLE and makes FABLE the current EDIT file. 2.

   The command:

```
*GET SHAKE 10/250 PUT ROMEO
CURRENT FILE IS $WORK.FICTION.ROMEO
*
```

   copies lines 10 through 250 of the file SHAKE into the file ROMEO and makes ROMEO the current EDIT file.

3. The following GET command copies the first line, the tenth through the twentieth lines, and the last line of SHAKE into HAMLET and makes HAMLET the current EDIT file:

```
*GET SHAKE NUM FIRST 10/20 LAST PUT HAMLET
CURRENT FILE IS $WORK.FICTION.HAMLET
*
```

4. The command:

```
*GET SHAKE 20/100
NAME THE NEW FILE: LYRIC
CURRENT FILE IS $WORK.FICTION.LYRIC
*
```

omits the PUT keyword, and EDIT prompts for a file name. You name the new file LYRIC. LYRIC becomes the current file. Lines 20 through 100 are copied into LYRIC.

If you do not name the new file (if you press BREAK , RETURN , or type CTRL-Y at the prompt), EDIT creates a temporary current file. EDIT deletes the temporary file when you exit EDIT or when you make another EDIT file the current file.

If the file name that you specify with the PUT keyword already exists, EDIT asks if it should purge the existing file. If you reply “yes” or “y”, EDIT purges the existing file, creates a new file, and copies text into the new file. For example, if the file named FABLE already exists, then the GET command prompts a response from EDIT:

EDIT is doublechecking your PUT command when it returns this message.

```
*GET AESOP 10/20 PUT FABLE
SHALL I PURGE THE OLD $SYSTEM.USER.FABLE? yes
CURRENT FILE IS $SYSTEM.USER.FABLE
*
```

Your “yes” reply causes EDIT to purge the existing file named FABLE and create a new file named FABLE. EDIT copies lines 10 through 20 of AESOP into FABLE and makes FABLE the current file.

### Adding All or Part of an EDIT File to the Current File

1. If the current file is RHYME, the command:

```
*GET JACK 10/20 TO LAST
LAST NEW LINE IS 916 <- 20
CURRENT FILE IS $WORK.FICTION.RHYME
*
```

   copies lines 10 through 20 of JACK to the last line of RHYME.

   The message:

```
LAST NEW LINE IS 916 <- 20
```

   tells you that:

   - ☐ Line number 20 from JACK is the last line added to RHYME
   - ☐ Line number 20 from JACK is now line number 916 in RHYME

2. If the current file is RHYME, the command:

```
*GET JILL 50.1/60 TO 1000.1
LAST NEW LINE IS 1021.1 <- 60
CURRENT FILE IS $WORK.FICTION.RHYME
*
```

   copies lines 50.1 through 60 of JILL into RHYME starting at line 1000.1. The message:

```
LAST NEW LINE IS 1021.1 <- 60
```

tells you that:

- ☐ Line number 60 from JILL is the last line added to RHYME
- ☐ Line number 60 from JILL is now line number 1021.1 in RHYME

3. If the current file is RHYME, the command:

```
*GET PEEP 75/85 TO SAME
CURRENT FILE IS $WORK.FICTION.RHYME
*
```

copies lines 75 through 85 of PEEP into RHYME without renumbering. If a duplicate line number exists, the lines from PEEP replace the corresponding lines in RHYME.

4. If the current file is RHYME, the command:

```
*GET PEEP 75/85 TO SAME KEEP
CURRENT FILE IS $WORK.FICTION.RHYME
*
```

copies lines 75 through 85 of PEEP into RHYME without renumbering. If a duplicate line number exists, the line from RHYME is retained.

5. If the current file is MAX, the commands NUMBER and GET:

```
*NUMBER ALL
*GET SMITH 55/66 TO 0
LAST NEW LINE IS .92 <- 66
CURRENT FILE IS $WORK.FICTION.MAX
*
```

   renumbers lines in MAX making the first line in the file line number 1, then copies lines 55 through 66 of SMITH into MAX starting at line number 0 (that is, copies SMITH to the beginning of MAX).

   For a specific example of renumbering the lines of a file when adding a large number of lines with a GET command, see Section 3, "Renumbering to Accommodate Added Lines."

#### Using GET With a Nondisk Device or Non-EDIT-Format File

EDIT assumes that `filename1` contains sequential records; the number of characters in each record is set by the INLEN option of the SET command. (The default setting of INLEN is 132. If the setting of INLEN is not sufficient to read an entire physical record, a file management error 21—illegal count specified—occurs and the EDIT program terminates abnormally.) EDIT reads records sequentially from `filename1` until it encounters the physical end-of-file.

1. This example illustrates using a magnetic tape unit to create a new current file.

```
*SET INLEN 100
*GET $TAPE PUT BOOK
*
```

   EDIT reads 100-byte records from the nondisk device named $TAPE (a magnetic tape unit) until an end-of-file mark is encountered. Each record is assigned a line number (starting with 1, in increments of 1) and put into an EDIT- format disk file called BOOK.

2. This second example illustrates using a non-EDIT-format disk file with the GET command.

Omits the PUT keyword

```
*SET INLEN 128
*GET TEST
NAME THE NEW FILE: NEWTEST
*
```

EDIT reads 128-byte records from the disk file named TEST until the end-of-file. The data is put into the EDIT file named NEWTEST.

### Adding Text From a Nondisk Device or Non-EDIT-Format Disk File

The commands:

```
*SET INLEN 100
*GET $TAPE NUM 30/50 TO 995.01
*
```

causes EDIT to read 100-byte physical records from $TAPE and add the thirtieth through fiftieth records to the current file beginning at line number 995.01.

**Tip** When you issue a GET command and `filename1` is an EDIT-format file, EDIT checks the validity of the file format. If EDIT detects any validity errors, it prints the message:

```
THIS FILE IS INVALID.  DO YOU WANT TO RECOVER?
```

If this happens to you, see the recovery procedure in Appendix B, “EDIT Error Recovery Procedure.”

## IMAGE Command

The IMAGE command replaces a portion of a line of text (line segment) with new text. You reference the line segment that you want to replace by specifying the character strings that surround it.

### What to Enter

```
      [ QUIET ]
IMAGE [ WORD  ] {"[lstring]"[newstring]"[rstring]" }
      [ BOTH  ] {END "[lstring]"[newstring]"       }
      [ ALL   ]

      [ { string-range-list }
                    [ RANGE string-range-list ] ... ]
```

`QUIET`

tells the EDIT program not to list lines as they are changed. If you omit QUIET, the EDIT program lists each line as it is changed.

`WORD`

tells the EDIT program to search for `lstring` and `rstring` as words. (A word is an alphanumeric character string that is preceded and followed by a space or any character other than a number or a letter.)

`BOTH`

tells the EDIT program to search for both uppercase and lowercase occurrences of `lstring` and `rstring`. If you omit BOTH, the EDIT program only searches for `lstring` and `rstring` that exactly match what you type.

ALL

tells the EDIT program to change all occurrences of the line segment that it finds in a line. If you omit ALL, the EDIT program changes only the first occurrence of the line segment that it finds in a line.

lstring

is a character string immediately preceding (to the left of) the line segment that you want to replace with new text. You can also enclose lstring within a pair of right slants (/) or single apostrophes (') as well as in quotes.

newstring

is the new text to be placed in the line segment found between lstring and rstring. You can also enclose newstring within a pair of right slants (/) or single apostrophes (') as well as in quotes.

rstring

is a character string immediately following (to the right of) the line segment that you want to replace with new text. You can also enclose rstring within a pair of right slants (/) or single apostrophes (') as well as in quotes.

END

tells the EDIT program that you are not specifying rstring. When you specify END, it means the line segment that you want to replace with new text extends from lstring through the last character in the line.

`string-range-list`

references one or more character strings in one or more lines of an EDIT file. Turn to "String-Range-List Parameter" in Section 5 for a full explanation of this range.

RANGE

is a keyword that tells EDIT to look for the "string" in the indicated line-range or string-range.

**How to Use IMAGE**

The IMAGE command works as follows:

- If you use the "lstring"newstring"rstring" form of the command, for each line in the range, the EDIT program searches for the left string (`lstring`). If it finds `lstring`, then the EDIT program searches for the right string (rstring). (The right string must occur to the right of the left string.) If the EDIT program finds both `lstring` and `rstring`, all the characters in the line segment bounded by these two strings are replaced by new text (`newstring`).
- If you use the END "lstring" newstring" form of the command, for each line in range, the EDIT program searches for `lstring`. If the EDIT program finds `lstring`, all the characters in the line segment from `lstring` to the end of the line are replaced by `newstring`. (See Example 3, following.)

**Examples**

1. The command:

```
*LIST 100
  100    What's going on here?
*IMAGE "s "new"?" 100
  100     What's new?
*
```

replaces the line segment bound by the string `s(space)` on the left and `?` on the right with the string `new`.

2. Any of the strings `lstring`, `newstring`, or `rstring` may be null. If `lstring` is null, `newstring` is inserted before `rstring` and begins in column 1. (All the original text that preceded `rstring` is replaced with `newstring`.) For example:

```
*LIST 3
    3     And the rain was upon the earth forty days and
*IMAGE ""It rained "forty" 3
    3     It rained forty days and forty
*
```

   If `newstring` is null, the EDIT program deletes all the characters between `lstring` and `rstring`. For example:

```
*IMAGE "forty ""and" 3
    3     It rained forty and forty
*
```

   If `rstring` is null, `newstring` is inserted immediately following `lstring`. For example:

```
*IMAGE "forty "nights "" 3
    3     It rained forty nights and forty
*
```

3. The following command replaces the line segment bound by `s` on the left through the end of the line with the string *a nice girl like you doing here?*:

```
*LIST 10
   10      What's new?
*IMAGE END "s "a nice girl like you doing here?"
   10      What's a nice girl like you doing here?
*
```

## JOIN Command

The JOIN command treats a contiguous block of nonblank lines as a paragraph. Within a paragraph, the JOIN command moves `words` between lines so that each line contains as many words as possible within a specified width. A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) any character string that occurs at the end of a line.

### What to Enter

```
JOIN [ QUIET ] { line-range } [ WIDTH rightcolumn ]
```

`QUIET`

tells the EDIT program not to list the lines as they are joined.

`line-range`

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

`WIDTH`

specifies the maximum number of characters to be placed on each line in the range during execution of the command.

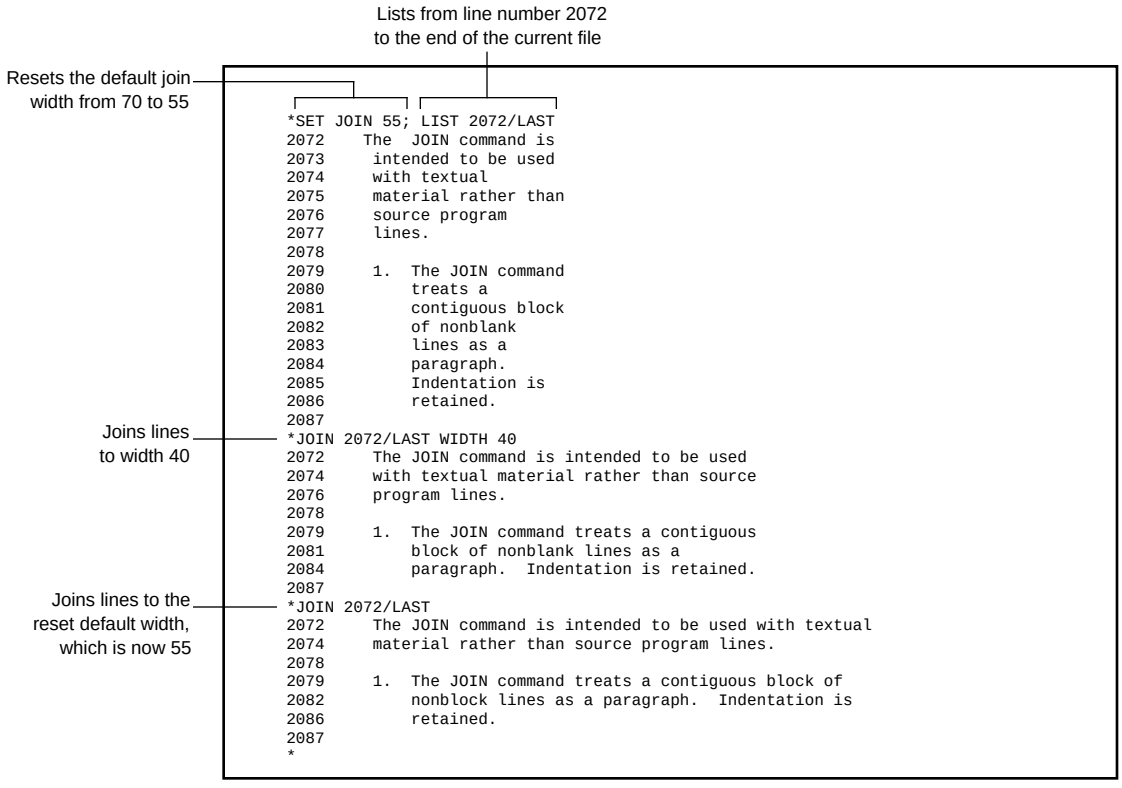
`rightcolumn`

is a number from 1 to 255. If you omit WIDTH, EDIT uses the `rightcolumn` value specified by the JOIN parameter of the SET command. (The default for `rightcolumn` is 70).

**How to Use JOIN**

- ☐ The JOIN command is intended to be used with textual material rather than source program lines.
- ☐ JOIN retains any leading (that is, left-side) indentation.
- ☐ JOIN stops filling lines when it encounters a blank line. However, if a blank line is contained within a specified range of lines, JOIN resumes filling lines at the next nonblank line. JOIN continues to fill lines until it reaches the last line in the range.

**Example** Study the following example:




```
*SET JOIN 55; LIST 2072/LAST
2072    The  JOIN command is
2073     intended to be used
2074     with textual
2075     material rather than
2076     source program
2077     lines.
2078
2079     1.  The JOIN command
2080         treats a
2081         contiguous block
2082         of nonblank
2083         lines as a
2084         paragraph.
2085         Indentation is
2086         retained.
2087
*JOIN 2072/LAST WIDTH 40
2072     The JOIN command is intended to be used
2074     with textual material rather than source
2076     program lines.
2078
2079     1.  The JOIN command treats a contiguous
2081         block of nonblank lines as a
2084         paragraph.  Indentation is retained.
2087
*JOIN 2072/LAST
2072     The JOIN command is intended to be used with textual
2074     material rather than source program lines.
2078
2079     1.  The JOIN command treats a contiguous block of
2082         nonblock lines as a paragraph.  Indentation is
2086         retained.
2087
*
```

The commands in the preceding example use both the SET JOIN and the JOIN commands to illustrate how to reset the default join width, how to join lines to a specified width, and how to join lines to a new default join width (reset with the SET command). See the SET command description later in this section for more information about SET JOIN.

**Tips**

- SET JOIN *width* changes the setting of the join width from 70 (the default join width) to the value you specify for *width*.
- To find out the value of your current join width, type the command QUERY JOIN at the EDIT prompt. EDIT then displays the current join width setting.
- When EDIT joins lines of text, it leaves two blank spaces after a period (.), a question mark (?), an exclamation point (!), or a colon (:) at the end of a text line.
- EDIT resets the default join width to 70 when you exit the program.

## LIST Command

The LIST command:

- ☐ Lists text lines in the current file on the home terminal, on a line printer, or to a process. (See Example 1, following.)
- ☐ Transfers text to non-Tandem systems by means of a non-disk device such as a magnetic tape unit. (See Example 2, following.)
- ☐ Writes fixed-length text records in non-EDIT-format disk files. (See Example 3, following.)

### What to Enter

```
     [ COL   ]
LIST [ SEQ   ] [ OUT listfile ] [ range-specifier ]
     [ UNSEQ ]
```

`COL`

tells the EDIT program to list a column number template above each line in the range.

`SEQ`

tells the EDIT program to list only the line numbers and not the text. If you omit SEQ, the EDIT program lists text lines with their line numbers.

`UNSEQ`

tells the EDIT program to list text lines without preceding them with line numbers. If you omit UNSEQ, the EDIT program lists text lines with their line numbers.

`OUT`

specifies a list device.

`listfile`

is `$device name`, `$logical device number`, or the name of a non-EDIT-format disk file.

*`range-specifier`*

indicates combinations of line-range and string-range parameters. Turn to "Range-Specifier Parameter" in Section 5 for a full explanation of this range.

**How to Use LIST**

- If you omit the OUT parameter, listing occurs on the device specified in the OUT filename parameter of the command to run the EDIT program.

  If `listfile` is a line printer or a process, the file name of the current file is printed at the top of each page; 56 text lines are listed per page.

  If `listfile` is an unstructured disk file or a magnetic tape, text lines are padded with blanks so that physical records are the length specified by the OUTLEN parameter of the SET command; characters to the right of the length specified in the OUTLEN parameter are listed on subsequent line(s) and padded with blanks up to the length. (See "Listing Text Onto Magnetic Tape" and "Listing Text Into a Non-EDIT-Format Disk File," following.)

- If you omit a range parameter, EDIT lists the current line.
- You can specify COL, UNSEQ or SEQ, and OUT in any order (but see "Tip" at the end of the command description).

### Examples Displaying Text Lines

1. The following command lists the block of text beginning at line number 35 and ending at line number 36 and puts a column number template above each line listed:

```
*LIST COL 35/36
          ....+....1....+....2....+....3....+....4....+....5..
   35     Text is added beginning at the line specified in the
          ....+....1....+....2....+....3....+....4..
   36     line parameter.  EDIT prompts for input by
*
```

   The numerals 1, 2, 3, 4, and 5 indicate columns 10, 20, 30, 40, and 50; the + signs occur at columns 5, 15, 25, and so on; the periods occur at intermediate column positions.

2. The command:

```
*LIST UNSEQ 1/2
A
Tandem
editor
text
*
```

   lists the text on the block of lines that begins at line number 1 and ends at line number 2.

3. The following command lists the line numbers of all lines in which the string *The editor* occurs in the range from line number 1000 to the end of the file:

```
*LIST SEQ "The editor" RANGE 1000/LAST
 1008      1010      1012      1025      1030      1032
 1050      1055      1071      1072      1078      1096
 1138      1139      1152      1164      1166      1178
 1192      1202
*
```

4. The following command lists the entire current file to the process named $S. Text lines are preceded by line numbers:

```
*LIST OUT $S ALL
*
```

### Listing Text Onto Magnetic Tape

You can write text to a nondisk device, such as magnetic tape, by specifying a magnetic tape unit for the listfile. When you do this, you should also specify the UNSEQ parameter. EDIT writes each line as one physical record on tape; the length of the record is set by the OUTLEN parameter of the SET command (the default length is 132 bytes). When all lines in the range are written, EDIT writes an EOF mark on tape.

The following commands write the entire text file, without line numbers, to a magnetic tape unit. Each line is written as one physical record on tape containing 130 bytes. Lines containing fewer than 130 characters are padded with trailing blanks. Characters to the right of column 130 are written in a subsequent line(s) of 130 bytes (also padded with trailing blanks if necessary).

```
*SET OUTLEN 130
*LIST UNSEQ OUT $TAPE ALL
*
```

### Listing Text Into a Non-EDIT-Format Disk File

When you list text into a non-EDIT-format disk file, EDIT writes fixed-length records, padded with trailing blanks if necessary, of the length specified by the OUTLEN parameter of the SET command.

The LIST command does not create a disk file. Therefore, you must use the CREATE command at the command interpreter to create the disk file before you issue the LIST command. Study the following example.

```
22 CREATE SLUG,1
23 EDIT WORM
CURRENT FILE IS $WORK.FICTION.WORM
*SET OUTLEN 128
*LIST UNSEQ OUT SLUG ALL
*
```

The first command in the sequence above creates a disk file named SLUG in your default subvolume on your default volume and assigns SLUG an extent size of 2048 bytes. The LIST command, which comes last in this sequence, writes the entire text named WORM, without line numbers, to the non-EDIT-format file named SLUG. Each line is written as one record containing 128 bytes. Lines containing less than 128 characters are padded with trailing blanks. Characters to the right of column 128 are written in subsequent line(s) of 128 bytes (padded with trailing blanks if necessary).

**Tip** Depending on how you enter them on the command line, the LIST command options COL and SEQ override each other when used together. EDIT executes a LIST COL SEQ command as LIST SEQ and a LIST SEQ COL command as LIST COL. Therefore, you can't combine these two options on the same command line.

## MOVE Command

The MOVE command moves or copies text lines from one location to another location in the current file.

### What to Enter

```
MOVE [ QUIET ] { line-range TO line }
                             [ BY line ] [, { line }
                             [ BY line ] ] ...
     [ COPY  ]
```

`QUIET`

tells the EDIT program not to list lines as they are

`COPY`

tells the EDIT program to move a copy of the original text one or more destinations. (See Example 2, following.) If you omit COPY, EDIT deletes the original text when it moves it to its new destination.

`line-range`

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

`TO`

specifies the destination location for the move operation.

*line*

is all the characters that have the same line number.

BY

specifies the numbering increment of the line numbers assigned to the moved text. If you omit BY, EDIT chooses the increment (either 1, .1, .01, or .001). EDIT chooses the increment that has the same number of digits as the fractional part in the *line* variable or is less than the difference between the destination line number and the next existing line number.

*incr*

is a number from .001 through 10.

**Examples** 1. The command:

```
*MOVE 20/50.1 TO LAST
*
```

moves the text on line numbers 20 through 50.1 to the end of the file. Line numbers 20 through 50.1 disappear.

2. The following command copies the text on line numbers 100 through 110, then moves a copy of the text starting at line number 20, line number 200, and line number 310. Line numbers 100 through 110 remain in the file.

```
*MOVE COPY 100/110 TO 20, 200, 310
*
```

3. The following command moves the text on line numbers 500 through 505 to the end of the file. The numbering increment between the new line numbers is 10. The lines of text are not listed as they are moved. Line numbers 500 through 505 disappear.

```
*MOVE QUIET 500/505 TO LAST BY 10
*
```

## NUMBER Command

The NUMBER command renumbers lines in the current file. The NUMBER command does not alter the order in which the lines of text occur.

### What to Enter

```
NUMBER { line-range } [ TO line-num ] [ BY incr ]
```

`line-range`

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

`TO`

specifies the first line number to be assigned to the renumbered lines. If you omit TO and if you specify a line-range parameter that indicates more than one line of text, the first line number assigned is that of the first line in the range; if you omit TO and if you specify the ALL keyword, the first line number assigned is line number 1.

*`line-num`*

specifies any EDIT file line. *line-num* is a number from .001 to 99999.999 or the keyword FIRST or LAST.

`BY`

specifies the numbering increment of the renumbered lines. If you omit the BY parameter, EDIT chooses the increment (either 1, .1, .01, or .001). EDIT chooses the increment that has the same number of digits as the fractional part of *line-num* or is less than the difference between the renumbered lines and the next existing line number.

*incr*

is a number from .001 through 10. If you specify an increment that requires line numbers to exceed 99999.999, EDIT ignores the increment you specify and renumbers the entire file starting at 1 by an increment of 1.

**How to Use NUMBER**

You can use the NUMBER command when you want to reassign the line numbers in your file or to create room within your file to insert text. You cannot, however, reassign line numbers with the NUMBER command in order to move lines in the file. For example, if the current file is:

```
 3      Humpty Dumpty
 7      sat on a wall,
 9      Humpty Dumpty
 9.1    had a great fall;
23      All the King's horses
30      And all the King's men
32      Couldn't put Humpty Dumpty
46      Together again.
```

the following command is invalid because line numbers exist between 6 and 9.1. When you issue an invalid NUMBER command, EDIT prints an error message and does not execute the NUMBER command:

```
*NUMBER 9.1/LAST TO 6
                    ^ -- ERROR --
THE RENUMBERING WOULD OVERLAP EXISTING LINES
*
```

**Examples** 1. The command:

```
*NUMBER ALL
*
```

renumbers the entire current text file. The first line is assigned line number 1 and the increment between lines is 1.

2. The command:

```
*NUMBER 20/30
*
```

renumbers lines 20 through 30 in place. The line-to-line numbering assigns even increments to the new line numbers even if there are more than 11 lines in the range.

3. The command:

```
*NUMBER ALL TO 1 BY 2
*
```

renumbers the entire current text file. The first line is given line number 1, the second line is given line number 3, the third is given line number 5 (that is, odd-numbered line numbers).

4. If you specify TO *line-num* but do not specify a BY parameter, EDIT chooses an increment (that is, 1, .1, .01, or .001) that either has the same number of digits as the fractional part in the line number or is less than the difference between the TO *line-num* and the next existing line number. For example, entering the command:

```
*NUMBER 100/110 TO 100.00
*
```

   renumbers lines 100 through 110; the first line is given line number 100, the second line is given line number 100.01, the third is given 100.02, and so on.

5. To create room in the file, you can renumber the file from a particular text line to the end of the file to create room in your file to insert text. For example, if your file is numbered sequentially in whole numbers from 1 to 100, the command:

```
*NUMBER 50/LAST TO 100
```

   renumbers lines 50/100 to 100/150, thus creating a gap in the middle of your file. See also “Renumbering to Add More Lines” in Section 3.

## OBEY Command

The OBEY command reads editor commands and text from a file.

### What to Enter

```
OBEY [ filename [ QUIET ] ]
```

*filename*

is the name of a command file that contains editor commands and, possibly, text. The file can be an unstructured, EDIT format, or structured disk file, a nondisk device, or a process. EDIT reads records from the command file until it detects the end-of-file, it reads an OBEY command with no *filename*, or an error occurs.

QUIET

tells the EDIT program not to echo the commands and text it reads from the command file.

### How to Use OBEY

- If your command file includes an OBEY command with a command *filename*, EDIT completes the current command line, closes the current command file, and reads commands from the newly specified command file.
- If your command file includes an OBEY command with no command *filename*, EDIT closes the current command file and waits for commands from the terminal. This means that any commands following the OBEY command are not performed.
- In a command file, precede each comment line with an asterisk (*). Any editor commands preceded by an asterisk are not performed; lines of text preceded by an asterisk are ignored. (See Example 2, following.)
- EDIT reads records from the command file of the length specified with the INLEN parameter of the SET command. The default record length is 132 bytes. (See the SET command, later in this section, for details.)

- If your command file includes an ADD command, EDIT treats every line after the ADD command as text until it reads a line that contains two consecutive right slants ( / / ) in columns 1 and 2. For example:

```
ADD 100
This is text being added
to the current file.  Text is added until EDIT
encounters only two consecutive right slants
in columns 1
and 2
//
```

  Placing the double slant directly in your text in columns 1 and 2 can be tricky. If you type two right slants after an ADD command, EDIT simply thinks you're done adding text and returns you to the EDIT prompt. So, if you use the ADD command in EDIT to create your command file, type two characters in columns 1 and 2 other than the right slants, then replace them by using either the CHANGE or FIX command (these commands are described previously). You can also compose your command file in EDIT VS (see Appendix C) or in TEDIT (see the TEDIT command, following).

- If your command file includes a REPLACE command, EDIT treats every line after the REPLACE command as replacement text until it exhausts the range or until it reads a line that contains two consecutive right slants in columns 1 and 2. If EDIT encounters two right slants, it deletes the remaining text lines in the REPLACE command range (if any). (See the REPLACE command later in this section.)

- If you include an OBEY command in an EDIT command line, EDIT executes all the other commands in the command line before executing the OBEY file. For example, the following command line:

```
24> EDIT EXAMPLE; OBEY TABS; SET JOIN 65; NUMBER ALL
```

  - Starts EDIT on the file named EXAMPLE
  - Sets the default join width to 65 characters
  - Renumbers all line numbers in the file EXAMPLE starting at 1

  EDIT executes these three commands first without obeying the file named TABS. Then, when you press \(key(RETURN)\), EDIT executes the OBEY TABS command. (Compare this with Example 1, following, which illustrates each command entered separately.)

- You should only include one OBEY command in an EDIT command line. If you do include more than one, EDIT will only execute the last OBEY command in the command line. (See Example 3, following.)

**Examples** 1. If the command file named TABS contains the line:

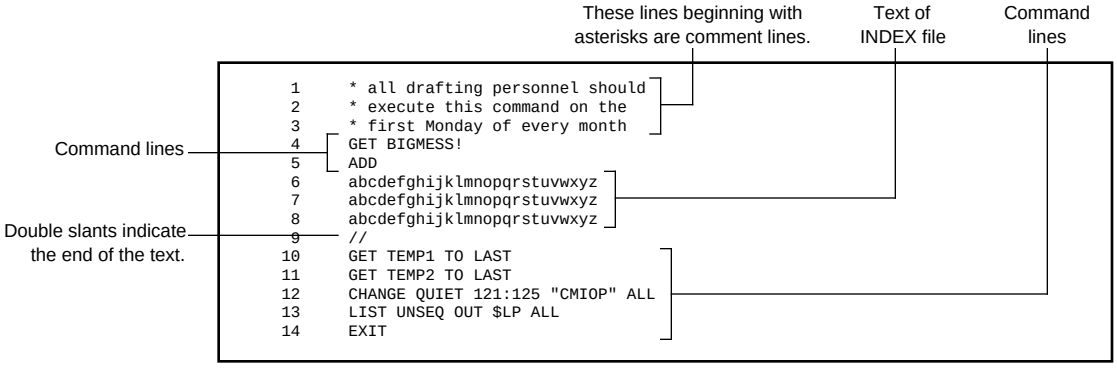
```
1    SET TABS 15 30 55
```

these three commands, typed on separate lines,

```
25> EDIT EXAMPLE
*OBEY TABS
*SET JOIN 65
```

start the EDIT program on the file named EXAMPLE, executes the commands in the file named TABS, and sets the join width to 65, respectively.

2. If the command file named INDEX contains the following lines:




the command line:

```
26> EDIT; OBEY INDEX
```

- ☐ Creates BIGMESS and makes it the current EDIT file
- ☐ Adds three lines of text to line number 0 of BIGMESS
- ☐ Copies the contents of TEMP1 to the end of BIGMESS
- ☐ Copies the contents of TEMP2 to the end of BIGMESS
- ☐ Modifies every line in BIGMESS so that it contains the characters CMIOP in columns 121 through 125
- ☐ Lists the entire contents of BIGMESS without line numbers on the printer named $LP
- ☐ Exits EDIT

3. If your command line includes more than one OBEY command:

```
1> EDIT MYFILE; OBEY AFILE; OBEY BFILE; OBEY CFILE
```

EDIT opens MYFILE for editing, then skips to the last OBEY command on the command line and obeys the command file CFILE. EDIT ignores the other OBEY commands on the command line.

## PUT Command

You can use the PUT command for two main purposes:

- ☐ Making the current file more compact.
- ☐ Copying all or a portion of the current file into a new EDIT-format file (keeping the same current file)

### What to Enter

The syntax of the PUT command to make the current file more compact is:

```
PUT !
```

The syntax of the PUT command for copying all or a portion of the current file into a new EDIT-format file is:

```
PUT filename [!] [ line-range-list | string-range ]
   [ RANGE { line-range-list | string-range } ]
```

*filename*

is the name of a fully or partially qualified file. If you are copying lines and *filename* names an existing file, you must specify the ! character. Specifying *filename* is required if you do not use the ! character. You may omit *filename* only when you are compressing the current file; otherwise, EDIT needs to know which *filename* to compress. (For more information on file names, see "How EDIT Files Are Named" in Section 6.)

!

tells the EDIT program to continue with the PUT operation even though you have specified the file name of an existing file. If you omit *filename* or if you specify the file name of the current file, the ! character tells the EDIT program to compress the current file to make it more compact.

`line-range-list`

references one or more contiguous lines or blocks of text in an EDIT file. Turn to "Line-Range-List Parameter" in Section 5 for a full explanation of this range.

`string-range`

references a character string in one or more lines of an EDIT file. Turn to "String-Range Parameter" in Section 5 for a full explanation of this range.

RANGE

is a keyword that tells EDIT to look for the "string" in the indicated line-range or string-range.

**How to Use PUT**

If you are copying lines and *filename* names a new file, EDIT creates an EDIT file with the new file name and copies the text into the new EDIT file. If you are copying lines and *filename* names an existing file, you must specify the ! char-acter. Then, EDIT purges the existing file, creates a new EDIT file with this file name, and copies text into this EDIT file.

The PUT command always creates an EDIT-format file and therefore cannot be used to transfer data to a nondisk device or to non-Tandem systems.

The lines copied to *filename* retain their original line numbers.

## Examples

### Copying All or Part of the Current File Into a New File

1. The command:

```
*PUT PUPPY !
*
```

   copies all of the current file into a new file named PUPPY. If a file named PUPPY already exists, the existing file is purged and a new file is created.

2. The following command copies lines containing the string JACK within the range of lines 100 through 1000 into a new file named HORNER. The lines copied to HORNER retain their original line numbers:

```
*PUT HORNER "JACK" RANGE 100/1000
*
```

### Creating a New, More Compact Current File

1. As you make modifications to an EDIT file, EDIT copies the modified lines of text into a new area in the disk file. EDIT can reuse portions of the disk file that have been obsoleted as a result of editing. However, portions of the file that are obsolete often remain and take up space on the disk.

   You use the QUERY command to determine the percentage of an EDIT file that is unused. If the unused portion (SLACK) is greater than 40%, you should use two PUT commands to compress the file. The first PUT command compresses the file. The second PUT command maximizes the compression. (See “Compressing Space in an EDIT File” in Section 6 for an example of the QUERY and PUT commands.)

2. Type this command at the command interpreter (rather than at the EDIT prompt)

```
FUP INFO filename, DETAIL
```

to see the number of extents (physical space) allocated to an EDIT file before and after you use the PUT ! command to compress the file. For example, the FUP INFO command returns information about any specified file. You want to look at the values displayed for EXT, BUFFERSIZE, EOF, and EXTENTS ALLOCATED. Study the following example.

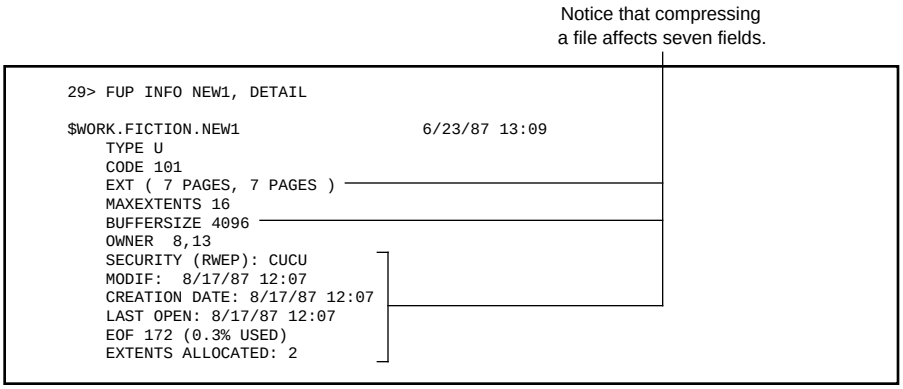
```
27> FUP INFO NEW1, DETAIL

$WORK.FICTION.NEW1                          6/23/87 13:09
    TYPE U
    CODE 101
    EXT (12 PAGES, 12 PAGES)
    MAXEXTENTS 16
    BUFFERSIZE 2048
    OWNER  8,13
    SECURITY (RWEP): CUCU
    MODIF:  3/04/87 14:35
    CREATION DATE: 3/04/87 14:35
    LAST OPEN: 8/17/87 08:46
    EOF 2220 (3.4% USED)
    EXTENTS ALLOCATED:  9
```

The command:

```
28> EDIT NEW1; PUT !; EXIT
```

compresses NEW1. The results of one compression are:




You can continue to use the PUT ! command; EDIT will compress the file and make the additional space on the disk available for use. Unless the file is especially large and heavily edited, two file compressions compress a file sufficiently.

## QUERY Command

The QUERY command displays the name of the current file, the current settings of EDIT control options, and the size and amount of disk space in use by the current file.

### What to Enter

```
      [ setoption [, setoption ... ] ]
QUERY [ NAME                         ]
```

*setoption*

is one or more of the following keywords:

| | | | | |
|---|---|---|---|---|
| BLOCK | DITTO | INLEN | OUTLEN | SHIFT |
| CONTROL | FREQ | JOIN | QUIET | TABS |

Each of these keywords is an option that you can control using the SET command. (See the SET command, later in this section, for details.)

NAME

tells the EDIT program to display the name of the current text file.

### How to Use QUERY

- If you do not specify a set option or NAME, EDIT displays the name of the current file, lists all of the set options, and displays the size and amount of disk space used by the current file.
- EDIT displays the default values for each of the set options, unless you have changed any. You can use the SET command to alter a set option. (See the SET command, which is described later in this section.)

**Examples** 1. The command:

```
*QUERY
```

returns information about the current file in the following format:

```
FILE $volname.subvolname.filename, setoption,
  setoption,
SPACE USED  end-of-file / filesize SLACK percentage
```

where:

*$volname.subvolname.filename*

is the full name of the current EDIT file. See "How EDIT Files Are Named" in Section 6 for information on EDIT file names.

*setoption, setoption,*

is information on each of the possible set options.

*end-of-file*

is the relative byte address of the current end-of-file location.

*filesize*

is the maximum size of the file in bytes.

*percentage*

indicates the unused portion of the file between file byte zero and the end-of-file location.

(See "Compressing Space in an EDIT File" in Section 6 for an example of why and how to use the QUERY command to determine file size.)

2. If this command:

```
*QUERY CONTROL, TABS, INLEN
```

returns the following information:

```
NOCONTROL, TABS  5  7  9, INLEN  132
```

it means that the NOCONTROL option is in effect; that TABS are set to columns 5, 7, and 9; and that INLEN is set to 132 bytes.

## REPLACE Command

The REPLACE command replaces all the text on a line with new text. There are two forms of the REPLACE command: the explicit form and the implicit form (see the examples that follow).

### What to Enter

☐ The syntax of the explicit REPLACE command is:

```
REPLACE [ QUIET ] { range-specifier }
```

`QUIET`

tells the EDIT program not to list the existing line of text prior to deleting it.

`range-specifier`

indicates combinations of line-range and string-range parameters. Turn to "Range-Specifier Parameter" in Section 5 for a full explanation of this range.

☐ The syntax for replacing lines implicitly with the REPLACE command is:

```
lnum "[ replacementtext ]"
```

*`lnum`*

is an EDIT-file line number that references an existing line whose text you want to replace or a nonexisting line where you want to add text.

*replacementtext*

is text. If you omit *replacementtext* and *lnum* references an existing line, EDIT deletes the current text on *lnum* and leaves a blank line. If you omit *replacementtext* and *lnum* references a line that does not exist, EDIT adds a blank line. The quote character (") is a string-field separator. You can also use a right slash (/) or an apostrophe (') for a string field separator. You must, however, use the same string-field separator at the beginning and end of *replacementtext*.

**How to Use REPLACE**

- The REPLACE command, when used explicitly, prompts for new text with an EDIT file line number for each line in the specified range. When you type the new text and press RETURN, EDIT deletes the old text on the line and replaces it with the new text. See Examples 1 and 2, following.
- If you type // in response to the line number prompt, EDIT asks the following question:

```
SHALL I DELETE THE REMAINING LINES?
```

  If you type "yes" or "y", EDIT deletes the current line and any remaining lines in the range. Any other response leaves the current and remaining lines unchanged. In either case, the REPLACE command terminates.
- If you press BREAK or type CTRL-Y in response to the line number prompt, the REPLACE command terminates and does nothing to any remaining lines in the range.
- If you want to replace lines implicitly, you simply type—at the EDIT prompt—the line number of a file and the new text you want placed on that line. You do not type the command REPLACE.

**Examples: Using the REPLACE Command Explicitly**

1. Suppose the current file contains these lines:

```
1     Little Jack Horner
2     Sat in a corner
3     Eating dumplings.
```

With the following series of commands, you replace the text on line number 3 and then list the result:

Displays the current text on line 3, then prompts for new text with that line number.

```
*REPLACE 3
    3     Eating dumplings.
    3     Eating a Christmas pie.
*LIST 3
    3     Eating a Christmas pie.
*
```

2. Suppose the current file contains these lines:

```
100     What a piece of work is a man!
101     how noble in reason!
102     how infinite in faculty!
103     in form and moving how express and admirable!
104     in action how like an angel!
105     in apprehension how like a god!
```

In the following series of commands, you tell EDIT to replace the contents of lines 100 through 102, to delete lines 103 through 105, and to list the results:

```
*REPLACE 100/105
  100     What a piece of work is a man!
  100     Unless above himself he can
  101     how noble in reason!
  101     Erect himself,
  102     how infinite in faculty!
  102     how poor a thing is man!
  103     in form and moving how express and admirable!
  103     //
SHALL I DELETE THE REMAINING LINES? y
  104     in action how like an angel!
  105     in apprehension how like a god!
*LIST ALL
  100     Unless above himself he can
  101     Erect himself,
  102     how poor a thing is man
*
```

### Examples: Using the REPLACE Command Implicitly

Suppose the current file contains this text:

```
3     Persons attempting to find a motive in this narrative
4     will be prosecuted;
5     persons attempting to find a moral in it
6     will be banished;
7     persons attempting to find a plot in it
8     will be shot.
```

With the following series of commands, you can implicitly replace lines 3 through 8 with new text, then list the results:

Note that you do not need to type REPLACE at each asterisk prompt when using REPLACE implicitly.

```
*3 "We should be careful to get out of an experience"
*4 "only the wisdom that is in it — and stop there;"
*5 "lest we be like the cat that sits on a hot stove."
*6 "She will never sit down on a hot stove again —"
*7 "and that is well; but also she will never sit"
*8 "on a cold one anymore."
*LIST ALL
    3     We should be careful to get out of an experience
    4     only the wisdom that is in it — and stop there;
    5     lest we be like the cat that sits on a hot stove.
    6     She will never sit down on a hot stove again —
    7     and that is well; but also she will never sit
    8     on a cold one anymore.
*
```

## REPLACE BLOCK Command

The REPLACE BLOCK command lets you edit text, a block at a time, with the EDIT program, which is primarily designed as a line editor. This command is useful only if you have a terminal with full-screen capabilities; see Appendix C, “Page Mode Editing.”

## SET Command

The SET command sets various internal parameters that control the EDIT program and provides the user with greater flexibility when creating and working in a file.

### What to Enter

```
SET setoption [, setoption ... ]
```

*setoption*

is one or more of the following keywords:

| | |
|---|---|
| BLOCK | JOIN |
| CONTROL/NOCONTROL | OUTLEN |
| DITTO/NODITTO | QUIET/NOQUIET |
| FREQ | SHIFT/NOSHIFT |
| INLEN | TABS/NOTABS |

BLOCK *numberoflines*

You use BLOCK if you have a full-screen terminal and can use the page mode editing capability (see the note following). BLOCK sets the maximum number of lines presented on the screen at any one time during a REPLACE BLOCK command. You can set *numberoflines* from 1 to 20. The default number of lines is 16. When *numberoflines* is set to 16, you can add up to eight lines of text to the lines displayed on the screen. (Most screens hold 24 lines.) If you expect that you might add more than eight lines of text, you can change *numberoflines* to a lower number (less than 16). If you expect to add less than eight lines of text, you can change *numberoflines* to a higher number (higher than 16). (See Example 6, following.)

**Note** Page mode editing commands do not work with TTY terminals, which display one line at a time and do not have full-screen capabilities. Therefore, while using EDIT, which is mainly a line editing program, you might never use REPLACE BLOCK or assign a value to the BLOCK setoption of the SET command.

```
CONTROL
NOCONTROL
```

CONTROL tells EDIT to process control characters that you type as significant characters. NOCONTROL tells EDIT to delete control characters that you type before it processes them. NOCONTROL is the default. (See Example 2, following.)

Different terminals respond differently to the same control character. If you set CONTROL, be very careful. Make sure you do not insert nonprintable characters into an EDIT file. (See "Text Lines and Printable Characters" in Section 6 for information on printable characters.)

```
DITTO character
NODITTO
```

DITTO sets a character to be used as a ditto. You can use any character for a ditto except the comma (,) or null. NODITTO disables the DITTO feature. NODITTO is the default. (See Example 7, following.)

FREQ *frequency*

You use FREQ to balance the two conflicting needs of good resource utilization and failure recovery. FREQ sets the maximum number of lines that can be altered by commands without the altered line(s) being written out to the disk file (alterations and additions to text are made in memory and buffered for later output to disk). You can set *frequency* from 1 to any number you feel is practical. Ten lines is the default frequency. (See Example 4, following.)

EDIT writes any buffered text out to disk just before the command input prompt is given. Therefore, the text file on disk is valid at each prompt.

INLEN *inrecordlength*

INLEN sets the read count used by EDIT when it reads from a command file and when the GET command reads a non-EDIT file. You can set *inrecordlength* from 1 to 255 bytes. The default in-record-length is 132 bytes. (See the GET command, earlier in this section, for examples.)

JOIN *rightcolumn*

JOIN sets the line width used by a JOIN command with no WIDTH specification. You can set *rightcolumn* from 10 to 255. The default right column is 70. (See the JOIN command, earlier in this section, for examples.)

OUTLEN *outrecordlength*

OUTLEN sets the number of characters written in each record when you direct LIST command output to a nondisk file or magnetic tape. Text lines with less than *outrecordlength* bytes are padded with trailing blanks when written. You can set *outrecordlength* from 1 to 255 bytes. The default out-record-length is 132 bytes. (See the LIST command, earlier in this section, for examples.)

```
QUIET
NOQUIET
```

A number of commands (such as REPLACE and CHANGE) have a QUIET keyword that suppresses the listing of altered lines. QUIET used with the SET command suppresses the listing of altered or deleted lines as the default for all the other editor commands that accept QUIET. NOQUIET reenables the listing of altered or deleted lines in all the other editor commands that accept QUIET. NOQUIET is the default. (See Example 5, following.)

```
SHIFT
NOSHIFT
```

SHIFT tells EDIT to upshift all alphabetical lowercase characters to uppercase characters. NOSHIFT tells EDIT to leave all alphabetical characters in the case in which you type them. NOSHIFT is the default. (See Example 1, following.)

```
TABS [ tabposition ] ...
NOTABS
```

TABS tells EDIT to simulate tabbing (by inserting blanks) when you type a *horizontal tab*. NOTABS tells EDIT not to simulate tabbing. NOTABS is the default.

When you use the EDIT program (when you are in line- editing mode), a horizontal tab is a CTRL-I character, NOT the key. To type a CTRL-I, press CTRL and hold it down while you press I .

*Tabposition* is an EDIT-file column number from 1 to 255. You can specify up to twenty tab positions. Separate each tab position with a space.

If you specify TABS, EDIT processes horizontal tab characters even though the NOCONTROL set option may be in effect. (See Example 3, following.)

**Examples** 1. The command:

```
*SET SHIFT
*ADD 1
    1     Lump the whole thing!
    2     Say that the Creator made Italy from designs
    3     by Michael Angelo!
*LIST 1/3
    1     LUMP THE WHOLE THING!
    2     SAY THAT THE CREATOR MADE ITALY FROM DESIGNS
    3     BY MICHAEL ANGELO!
*
```

sets the setoption SHIFT so that EDIT changes all alphabetical characters to uppercase, no matter how you entered them.

2. When NOCONTROL is in effect, the line:

CTRL-G is CTRL and G pressed simultaneously

```
ABC(CTRL-G)D
```

is treated as:

```
ABCD
```

when processed by EDIT.

3. The following command converts horizontal tab characters to single blanks:

```
*SET TABS
*
```

The following command sets the tabs in columns 5, 10, and 20:

```
*SET TABS 5 10 20
*
```

With TABS set to these columns, if you type the text:

```
A(CTRL-I)B(CTRL-I)C(CTRL-I)D
```

the text may look incorrect on the screen but when you list the line you will see that your EDIT file is correct. For example:

As soon as you type CTRL-I, the cursor jumps down to the next line. Don't worry. Keep typing.

You see that EDIT converted each CTRL-I to the correct number of blank spaces.

```
*ADD 1
    1     A(CTRL-I)
B(CTRL-I)
C(CTRL-I)
D
    2     //
*LIST COL 1
          ....+....1....+....2....+....3
    1     A   B    C         D
*
```

When TABS is in effect, no actual CTRL-I characters remain in the text lines. If you want to retain CTRL-I characters in your file, use the command:

```
*SET CONTROL, NOTABS
*
```

When the CONTROL and NOTABS set options are in effect, EDIT adds CTRL-I characters to your EDIT file and does NOT convert them to blanks.

To change tab positions, enter a new SET TABS command with the new position(s) specified. When you specify new tab positions, EDIT clears the old tab positions.

If you horizontal tab past the last tab position, EDIT converts the horizontal tab to a single blank.

4. The following command writes out each line as you alter it:

```
*SET FREQ 1
*
```

This command causes a great number of disk accesses, perhaps diminishing system performance as a whole. If you set frequency higher than one, disk writes may be done more often than that frequency, but they are never done less often. For example, in the default case (SET FREQ 10), the file is valid at least every ten changed lines, perhaps more often.

5. The following command sets the QUIET option so that EDIT commands that normally list results, such as the CHANGE command, no longer list results:

```
*SET QUIET
*CHANGE "The"This" 6
*
```

6. The following command causes a block of ten lines to appear for editing by a REPLACE BLOCK command. (Recall that terminals that can't accommodate page mode editing ignore the BLOCK *setoption* and can't use the REPLACE BLOCK command; see the previous description of the BLOCK option under "What to Enter.") Given a 24-line screen, this command presents ten lines at a time and allows you to add up to 14 more lines of text:

```
*SET BLOCK 10
*
```

7. The following command sets the ditto character to an ampersand (&):

```
*SET DITTO &
*
```

   A single & copies all data from the previous line to the current line until it encounters the first nonblank. For example:

```
Copy all information
&        headings
```

   enters the following into the EDIT file:

```
Copy all information
Copy all headings
```

   Two & characters copy only the information between them. For example:

```
1064 Nut   34 A
1132 Bolt  && A
```

enters the following into the EDIT file:

```
1064 Nut   34 A
1132 Bolt  34 A
```

You can also use DITTO with the TABS feature. For example, the commands:

```
*SET TABS 10, DITTO &
*ADD
    1     Hammer horizontal tab$24.00
    2     Saw horizontal tab&
    3     //
*LIST 1/2
    1     Hammer    $24.00
    2     Saw       $24.00
*
```

tell EDIT how to set up the two-column text and what text to duplicate in column 2.

**Tip** The following options can be abbreviated as shown:

| | | |
|---|---|---|
| NO CONTROL | = | NC |
| NO DITTO | = | ND |
| NO QUIET | = | NQ |
| NO SHIFT | = | NS |
| NO TABS | = | NT |

## TEDIT Command

The TEDIT command allows you to start the PS TEXT EDIT (TEDIT) editing program from EDIT.

### What to Enter

```
TEDIT [ filename ] [ TEDIT open option ]
                                  [ ; TEDIT commands ]
```

*filename*

is the name of a file. If you do not specify *filename*, TEDIT opens the current file with the open options currently in effect. The TEDIT command accepts a full or a partial file name. (See "How EDIT Files Are Named" in Section 6 for information on file names.)

*TEDIT open option*

is one or more options that you can specify to control how TEDIT opens a file. The TEDIT open options are:

```
READONLY
ANYWAY
!
?
```

READONLY

opens the file for reading only.

ANYWAY

opens *filename* for READONLY when the file is not secured for WRITE access; without ANYWAY, TEDIT opens the file for WRITE access.

!

creates *filename* if *filename* doesn't exist, without prompting you with a confirmation message.

?

starts the HELP program for TEDIT.

If you do specify *open option*, you must also specify

*TEDIT commands*

is a list of one or more TEDIT commands. Separate multiple TEDIT commands with a semicolon.

**How to Use TEDIT**

- The TEDIT command is particularly useful when you want to use TEDIT from a program that starts EDIT as the default editor. Because you can supply a command string to TEDIT, you can optionally tell TEDIT how to open the file and where in the file you would like to begin.
- When you start TEDIT, any open OBEY file is closed with the current file. If you choose to edit the current file, either by naming it or by default, TEDIT positions you at the beginning of the file.
- EDIT commands cannot follow a TEDIT command on the same command line.
- When you exit TEDIT, you return to EDIT. The current file is the file you were editing before you started the TEDIT program. EDIT reinstates the existing open options, locates you at the beginning of the file, and displays this message:

```
CURRENT FILE IS $volume.subvolume.filename
```

- You cannot start TEDIT on an undefined file (a temporary file). TEDIT doesn't recognize the file name of a temporary file.

**Examples** The following examples illustrate how you can use the TEDIT command.

1. The command:

```
*TEDIT MYFILE !
```

   tells EDIT to start TEDIT for editing the file named MYFILE. The ! character tells TEDIT to create the file without prompting you if MYFILE doesn't already exist.

2. The command:

```
*PUT DUPFILE !; TEDIT; DISPLAY 55
```

   copies the current file into a new file named DUPFILE, starts TEDIT on the current file (since no file name was specified), and locates you on line 55 of DUPFILE, now the current file, where you can begin editing with TEDIT.

**Tips**

- ☐ EDIT could have trouble starting TEDIT for several reasons. If you receive a message in response to your TEDIT command, refer to Appendix A, "EDIT Error Messages," for additional information.
- ☐ For complete information regarding the TEDIT editing program, see the *PS TEXT EDIT Reference Manual* or the *PS TEXT EDIT* and *PS TEXT FORMAT User's Guide.*

## XEQ Command

The XEQ command starts the EDIT VS program, the screen editor. This command is useful only if you have a terminal with full-screen capabilities; see Appendix C, “Page Mode Editing.”

## ENV Command

The ?ENV command lists the current system name and volume name.

### What to Enter

```
?ENV [ SYSTEM ]
     [ VOLUME ]
```

SYSTEM

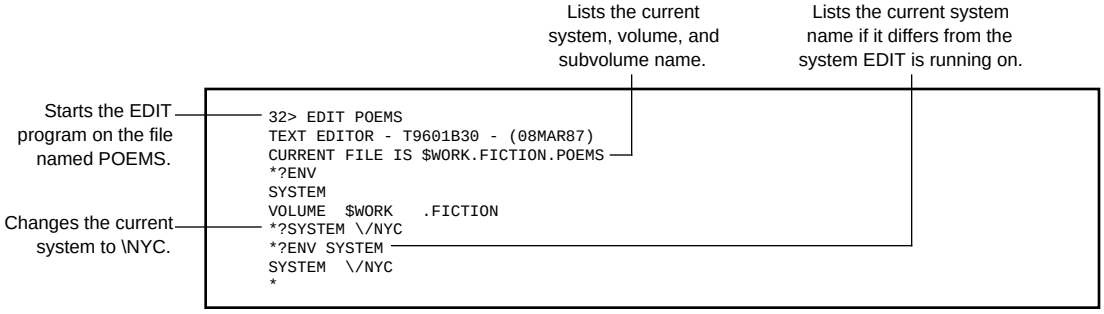
tells EDIT to list the current system name. If you have not used the ?SYSTEM command to change the current system, the ?ENV command returns a blank field. (See Example, following.)

VOLUME

tells EDIT to list the current volume name and subvolume name.

### Example

The following example illustrates the use of the ?ENV command.



## FILES Command

The ?FILES command lists the file names of all the files in a subvolume.

### What to Enter

```
?FILES [ [ \sysname. ] [ $volname. ]
                                    [ subvolname ] ]
```

*\sysname*

is a system name. If you omit *\sysname*, EDIT uses the current system name.

*$volname*

is a volume name. (See "How EDIT Files Are Named" in Section 6 for information on volume names.) If you omit *$volname*, EDIT uses the current volume name.

*subvolname*

is a subvolume name. (See "How EDIT Files Are Named" in Section 6 for information on subvolume names.) If you omit *subvolname*, EDIT uses the current subvolume name.

**Examples** 1. The following command lists the files on the current subvolume:

```
*?FILES
LETTER    TMSHT     TMST1
*
```

2. The following command lists the files in the subvolume FICTION on the volume $WORK:

```
*?FILES $WORK.FICTION
AESOP      COLE      JACK      JILL      PEEP      SHAKE
*
```

## ?SYSTEM Command

The ?SYSTEM command sets the current system.

### What to Enter

```
?SYSTEM [ \sysname ]
```

*\sysname*

is a system name. If you do not give a system name, EDIT sets the current system to the system that was current when you started the EDIT program.

## ?VOLUME Command

The ?VOLUME command sets the current volume and subvolume.

**What to Enter**

```
?VOLUME [ [ $volume.] [ subvolume ] ]
```

*$volname*

is a volume name. (See "How EDIT Files Are Named" in Section 6 for information on volume names.) If you omit *$volname*, EDIT sets the current subvolume on the current volume.

*subvolname*

is a subvolume name. (See "How EDIT Files Are Named" in Section 6 for information on subvolume names.) If you omit *$volname* and *subvolname*, EDIT sets the current subvolume to the subvolume that was current when you started the EDIT program.

**Example** For the following example, the current volume is $WORK and the current subvolume is FICTION.

```
33> EDIT
*?VOLUME $FUN.GAMES
*?FILES
MONOPOLY    SPACEINV    STARWARS
*?VOLUME
*GET $FUN.GAMES.MONOPOLY PUT MONOPOLY
CURRENT FILE IS $WORK.FICTION.MONOPOLY
*
```

This series of commands does these five things:

1. The EDIT command starts the EDIT program.
2. The ?VOLUME command makes $FUN.GAMES the current subvolume.
3. The ?FILES command lists the files in the current subvolume ($FUN.GAMES).
4. The ?VOLUME command moves you back to the volume and subvolume that was current when you started the EDIT program ($WORK.FICTION)
5. The GET and PUT commands places the contents of $FUN.GAMES.MONOPOLY into the file named MONOPOLY in the current subvolume ($WORK.FICTION) and makes $WORK.FICTION.MONOPOLY the current EDIT file.

# 5 Range Summary

## Introduction to Ranges

Most of EDIT commands require you to specify a range parameter. A range parameter tells the command which line or lines of text, column or columns, or specific string or strings to operate on.

Assume you have an EDIT file named AESOP that contains the lines:

```
12     The boy called out "Wolf, Wolf!"
13     and the villagers came out to help him.
14     A few days afterward he tried the same trick,
15     and again they came to his help.
16     Shortly after this a Wolf actually came,
17     but this time the villagers thought the boy was
18     deceiving them again and nobody came to his help.
19
20
21     A liar will not be believed, even when he speaks
22     the truth.
```

A line of text is all the characters that have the same line number. For example, the characters:

```
The boy called out "Wolf, Wolf!"
```

are one line of text with the line number 12.

A group of text is two or more contiguous lines of text. For example, the lines:

```
A few days afterward he tried the same trick,
and again they came to his help.
Shortly after this a Wolf actually came,
```

are a group of text with the beginning-line number 14 and the ending-line number 16.

When referring to columns, recall that an EDIT file generally appears as numbered from column 1 at the leftmost edge to column 80 at the rightmost edge of your screen; an EDIT file has a maximum width of 255 columns. A column is all the blank and nonblank characters that have the same column number in your file. For example, these highlighted characters:

```
The boy called out "Wolf, Wolf!"
and the villagers came out to help him.
A few days afterward he tried the same trick,
and again they came to his help.
Shortly after this a Wolf actually came,
but this time the villagers thought the boy was
deceiving them again and nobody came to his help.

A liar will not be believed, even when he speaks
the truth.
```

(including the spaces) make up column 31 in your file named AESOP.

A string is a series of characters such as a word, phrase, or number enclosed in quotation marks. For example, the string *his help.*:

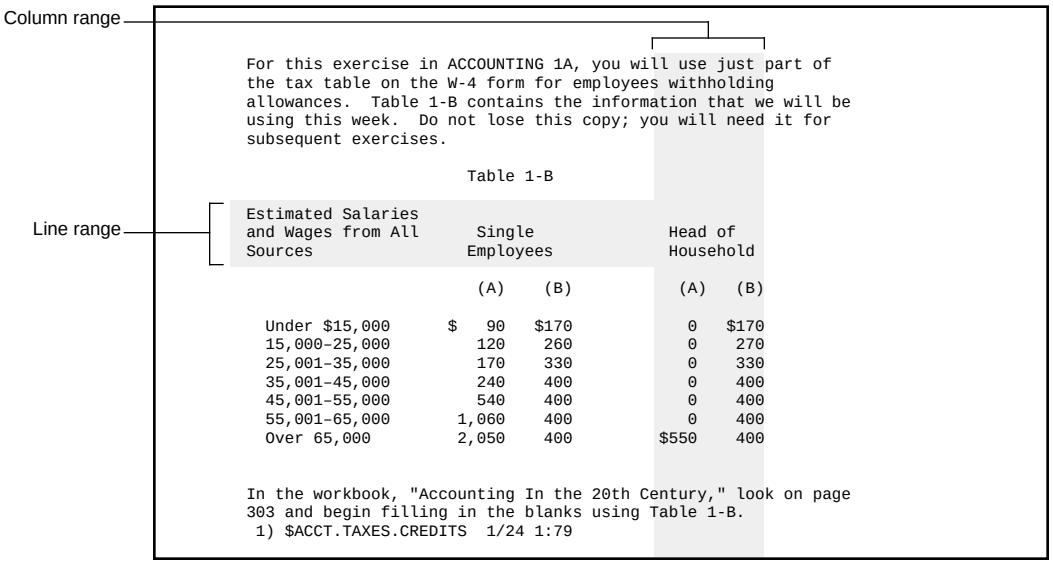
```
15     and again they came to his help.
18     deceiving them again and nobody came to his help.
```

appears in lines 15 and 18 of your file.

Ranges can be generally considered as ranges of lines or columns. Commands that accept line ranges allow EDIT to work on one or more lines of a file. Commands that specify a range of one or more columns tell EDIT to operate on that column range throughout the entire file.

Figure 5-1 illustrates the concept of line and column ranges.

**Figure 5-1. Concept of Ranges (Lines and Columns)**




EDIT recognizes nine specific range parameters in the various editor commands. Four of these ranges are called “range-list” parameters and are based on one of the four simpler ranges. “List” tells you that the range can contain more than one range, and each range is listed on the command line. For example, this command line contains a line-range parameter:

```
*LIST FIRST/20
```

and this command line contains a line-range-list (multiple

line-range) parameter:

```
*LIST 10/20 35 44/LAST
```

A range-list parameter appears in any editor command syntax that accepts optional repetitions of the simpler range. Each range-list parameter is described in context with its appropriate range.

All range parameters are summarized in Table 3. You can also turn to the RANGE tabs later in this section for detailed descriptions of each of the range parameters.

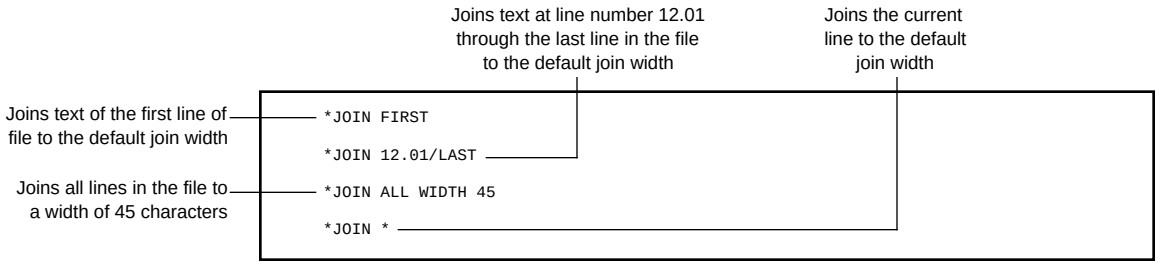
You do not have to memorize the different range parameters. When you look up an editor command, its syntax statement lists the range parameters it accepts. If you do not know the syntax of one or more of these range parameters, go to the appropriate range and look it up. For example, the syntax of the JOIN command is:

```
JOIN  [ QUIET ] [ line-range ] [ WIDTH rightcolumn ]
```

The JOIN command accepts the line-range parameter. The syntax of the line-range parameter is:

```
ALL
line / [ line ]
```

Consequently, some valid JOIN commands are:




An asterisk (*) in a range parameter references the current line. The current line is the last line number specified in the range parameter of the most recent editor command. For example, the following command lists the text on line number 8 and line number 10:

```
*LIST 8 10
```

Line number 10 is the current line. The following command starts adding text after line number 10:

```
*ADD *
```

**Table 5-1. Range Summary** (Page 1 of 3)

| Range | What It References | How You Specify The Range | Examples |
|---|---|---|---|
| Line-range | one or more lines of contiguous text | by line number, FIRST, LAST, or *, optionally followed by a signed number, or by beginning-number and ending-number, optionally followed by a signed number | 10.2<br>6+23<br>FIRST<br>LAST-7<br>F/10.7<br>18/206<br>L-53/L-10<br>7/*+22 |
| Line-range-list | one or more line-ranges | by line number, FIRST, LAST, or *, optionally followed by a signed number, or by beginning-number and ending-number, optionally followed by a signed number | 8.3<br>8 10 12<br>F *+10/L<br>12.3/* 42<br>1/5 7/L-6 |
| String-range | one or more lines of text with specified string | by character string optionally qualified by keywords NOT, BOTH, WORD, and PADDED | “Phoenix”<br>WORD “ab”<br>COL 7:26 “out” |

**Table 5-1. Range Summary** (Page 2 of 3)

| Range | What It References | How You Specify The Range | Examples |
|---|---|---|---|
| String-range-list | one or more string-ranges | by one or more character strings optionally qualified by keyowrds NOT, BOTH, WORD, and PADDED | “we”<br>BOTH “we” “you”<br>WORD “ab” “to”<br>“we” “it” “us” |
| Column-range | one or more columns | by one column number or keyword or by column numbers or keywords indicating the start and end of one range | 11<br>F:16<br>30:85<br>F:L |
| Column-range-list | one or more column-rnages | by one column number or keyword or by column numbers or keywords indicating the start and end of one or more ranges | 25<br>10:30<br>F:16 30 40<br>15 28:48 L<br>F:20 40 60:L<br>F 24 36 L |

**Table 5-1. Range Summary** (Page 3 of 3)

| Range | What It References | How You Specify The Range | Examples |
|---|---|---|---|
| Ordinal-range | one or more lines of contiguous text | by one or more integers, FIRST, or LAST that specify the order in which a string or line† appears in file | F<br>120<br>8/L |
| Ordinal-range- | one or more ordinal-ranges | by one or more integers, FIRST, or LAST that specify the order in which a string or line† appears in file | 12<br>F 55/80<br>39/L<br>12 28 78<br>F/38 42/88 |
| Range-specifier | two or more lines or groups of lines of contiguous text | by line number, FIRST, LAST, or *, (optionally followed by a signed number), by beginning-line and ending-line (optionally followed by a signed number), and by character string | 1 5, “Shy”<br>1/5, 7/L-6<br>WORD “ab”, 6/L |

† Only the GET command syntax accepts a reference to one or more lines in the ordinal-range parameter; all other commands reference one or more locations (such as the first, third, or tenth occurrence) of a string in a file when using the ordinal-range parameter.

## Line-Range Parameter

The line-range parameter references one or more contiguous lines in an EDIT file by specifying a line number or a keyword, optionally followed by a signed number (see *offset*, following).

### Syntax of the Line-Range Parameter

```
line-range: one of

   ALL
line [ / line ]
```

ALL

is all lines in your file.

*line* [ / *line* ]

is one or more lines you indicate either by a number, keyword, or a symbol. You use the forward slant (/) to separate beginning and ending points of a range; for example, F/L or 12/200.

*line*

is one of the following:

```
{ { line-number   } [ } +|- } offset ]   }
{ { FIRST         }                      }
{ { LAST          }                      }
{ { *             }                      }
```

*line-number*

is a line number between 0 and 99999.999; for example, 23132.23.

FIRST

is the first line in your file.

LAST

is the last line in your file.

*

is the current line.

*offset*

is a whole number that EDIT uses to compute how many lines to add to or subtract from *line* to arrive at the desired line.

**Examples** 1. When you specify a single line number as the range in an editor command, the command affects just that line. If the current file is:

```
1.1   I'd be tender. I'd be gentle.
1.2   And awfully sentimental
1.3   Regarding love and art.
1.4   I'd be friends with the sparrows
1.5   And the boy who shoots the arrows
1.6   If I only had a heart.
```

then:

Lists line 1.4

```
*LIST  1.4
    1.4   I'd be friends with the sparrows
*
```

2. If the current EDIT file is:

```
23     Mary had a little lamb,
24     Little lamb,
25     Little lamb.
26     Mary had a little lamb,
27     Whose fleece was white as snow.
```

then:

Lists line 23

Lists line 27

```
*LIST FIRST
    23     Mary had a little lamb,
*LIST LAST
    27     Whose fleece was white as snow.
*
```

3. If the current EDIT file is:

```
.01  Humpty Dumpty
.02  Sat on a wall.
.03  Humpty Dumpty
.04  Had a great fall.
.05  All the king's horsemen
.06  And all the king's men
.07  Couldn't put Humpty
.08  Together again.
```

then:

Lists the current line (.05)

Lists the fifth line before line .07

Lists line .05

Lists the second line after the current line

Lists the last line (too few lines exist to list the tenth line after the current line.)

```
*LIST .05
    0.05  All the king's horsemen
*LIST *
    0.05  All the king's horsemen
*LIST * + 2
    0.07  Couldn't put Humpty
*LIST .07 - 5
    0.02  Sat on a wall.
*LIST * + 10
    0.08  Together again.
*
```

4. If the current EDIT file is:

```
3     The itsy bitsy spider
4     Went up the water spout.
5     Down came the rain
6     And washed the spider out.
6.1   Out came the sun
6.2   And dried up all the rain.
7     Then the itsy bitsy spider
8     Went up the spout again.
```

then the commands in the following example cause EDIT to respond in this sequence:

- ☐ The first command lists the block of text bound by line number 6 and line number 7, which is the current line.
- ☐ The next command is invalid because beginning-line (7) is higher than ending-line (3).
- ☐ The next command lists the block of text bound by the first line and the third line before the current line (7).
- ☐ This next command lists the block of text bound by the current line (6) and the last line.
- ☐ The last command lists line number 6.1.

```
*LIST 6/7
     6     And washed the spider out.
     6.1   Out came the sun
     6.2   And dried up all the rain.
     7     Then the itsy bitsy spider
*LIST 7/3
         |Lo -- ERROR --
ALL RANGES MUST BE GIVEN AS LOWER/HIGHER
*LIST FIRST/*-3
     3     The itsy bitsy spider
     4     Went up the water spout.
     5     Down came the rain
     6     And washed the spider out.
*LIST */LAST
     6     And washed the spider out.
     6.1   Out came the sun
     6.2   And dried up all the rain.
     7     Then the itsy bitsy spider
     8     Went up the spout again.
*LIST 6.1/6.1
     6.1   Out came the sun
*
```

## Line-Range-List Parameter

The line-range-list parameter is based on the line-range parameter. It allows an optional repetition of the line-range parameter when an editor command will accept it. Because the repetition of the line-range is optional, an editor command offers more flexibility when the syntax specifies a line-range- list. A line-range-list parameter can represent a single line of text as well as something as complex as separate groups of text in a file.

### Syntax of the Line-Range-List Parameter

```
line-range-list: is the following
   line-range [ line-range ] ...
```

Refer to the previous presentation of the Line-Range Parameter for a full description.

### How to Use the Line-Range-List Parameter

- ☐ Separate each range with a space. (See the examples below and examples for following range-list parameters.)
- ☐ When you specify a line-range-list parameter, you can specify single line numbers in any order, but you must specify the beginning-line and ending-line of any group of contiguous lines of text in ascending order (lower to higher). (See examples below.)

**Examples** 1. If the current EDIT file is:

```
1.1   There was a maid on Scrabble Hill,
1.2   And if not dead, she lives there still;
1.3   She grew so tall, she reached the sky,
1.4   And on the moon, hung clothes to dry.
2.1   Wee Willie Winkie
2.2   Runs through the town,
2.3   Upstairs and downstairs,
2.4   In his nightgown;
2.5   Rapping at the window,
2.6   Crying through the lock,
2.7   "Are the children in their beds?
2.8   Now it's eight o'clock."
```

then:

```
*LIST 1.1 2.1 2.4/2.6 LAST
    1.1   There was a maid on Scrabble Hill,
    2.1   Wee Willie Winkie
    2.4   In his nightgown;
    2.5   Rapping at the window,
    2.6   Crying through the lock,
    2.8   Now it's eight o'clock."
*
```

lists lines 1.1, 2.1, 2.4, 2.5, 2.6, and 2.8.

2. If the current file is:

```
11     Tom, Tom
12     The piper's son
13     Stole a pig
14     And away he run
15     The pig was eat
16     And Tom was beat,
17     And Tom went crying
18     Down the street.
19     Tom, Tom
20     The piper's son
21     He learned to play
22     When he was young;
23     But all the tunes
24     That he could play
25     Was "Over the hills
26     And far away."
```

   then:

```
*LIST 16/17 12 LAST-1/LAST
    16     And Tom was beat,
    17     And Tom went crying
    12     The piper's son
    25     Was "Over the hills
    26     And far away."
*
```

   lists lines 16, 17, 12, 25, and 26.

   Notice that the LIST command lists the lines in the order in which you specified them. Line number 26 is now the current line.

3. If the current file is the same file as in the preceding example (Example 2), and if 26 is the current line, the following command results in an error message:

```
*LIST *-10 *-4/*-5 *-13
           |Lo  --  ERROR --
ALL RANGES MUST BE GIVEN AS LOWER/HIGHER
```

If you change the command and specify:

```
*LIST *-10 *-5/*-4 *-13
   16     And Tom was beat,
   21     He learned to play
   22     When he was young;
   13     Stole a pig
*
```

then EDIT lists lines 16, 21, 22, and 13.

Notice that line number 26 remains the current line until EDIT executes the command, then line number 13 becomes the current line.

4. If the current file is:

```
1     Mary had a little lamb
2     Whose fleece was white as snow
3     And everywhere that Mary went
4     The lamb was sure to go.
```

then:

```
*LIST ALL
    1     Mary had a little lamb
    2     Whose fleece was white as snow
    3     And everywhere that Mary went
    4     The lamb was sure to go.
*
```

lists all the lines in the file.

## String-Range Parameter

The string-range parameter is a character string, accompanied by optional keywords, that references one or more lines in an EDIT file.

### Syntax of the String-Range Parameter

```
string-range: is the following

   [ string-option ] "string"
   [ COL column-range-list ]
   [ NUM ordinal-range-list ]


   string-option: is the following

      [ NOT    ]
      [ BOTH   ]
      [ WORD   ]
      [ PADDED ]
```

NOT

instructs EDIT command to find the lines in the EDIT file that do not contain the character string *string*. If you omit NOT (and are not using COL or NUM keywords), EDIT finds all the lines that contain *string*.

BOTH

tells EDIT command to find the lines in the EDIT file that contain both uppercase and lowercase occurrences of the character string *string* If you omit BOTH, EDIT only finds the occurrences of *string* that exactly match what you type.

WORD

tells EDIT command to find the lines in the EDIT file in which the character string *string* is a word. A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line. If you omit WORD, EDIT finds all occurrences of *string*.

PADDED

tells EDIT to treat the line as though it contains all blanks from the last nonblank column through column 239.

*string*

is a character string to search for in your EDIT file. The quotes (") separate *string* from the rest of the command. You can also enclose *string* within a pair of single apostrophes (') or right slants (/). You must, however, use the same separator at the beginning and end of the string.

COL column-range-list

is the keyword COL followed by a column-range-list, indicating that EDIT should search for *string* in one or more specified columns. (See the Column-Range-List description, following.)

`NUM ordinal-range-list`

is the keyword NUM followed by an ordinal-range-list, indicating that EDIT should search for the occurrences of *string* at one or more ordinal positions (first, third, seventh, and so on) of lines in the file. (See the following Ordinal-Range-List description.)

**How to Use the String-Range Parameter**

- In addition to locating strings on specific (literal) lines, you can use the NUM ordinal-range-list option to locate one or more strings in your file by the ordinal position (first, third, tenth, and so on) of the line or lines containing that string.

  The ordinal position of the lines containing a string refers to how often *string* occurs on one or more lines in the file. EDIT locates the first line that contains *string*, goes to the next line that contains *string*, and so on, until it locates *string* in the ordinal position of the line or lines specified in the ordinal-range parameter. For example, if you specify NUM 5 with *string*, EDIT will locate the fifth line of your file that contains the string, regardless of its line number. (See Example 7.)

- When you give a left column number and a right column number, the right column number must be equal to or greater than the left column number (for example, 5:5 is valid, FIRST:LAST is valid, 15:7 is not valid). If you only specify a left column number, *string* must begin in that column; if you specify a left and right column number, *string* must be within the boundaries of the column.

- A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line. For example, if your file contains the lines:

```
Old King Cole was a merry old soul,
And a merry old soul was he;
He called for his pipe, and he called for his bowl,
And he called for his fiddlers three.
Every fiddler, he had a fine fiddle,
And a very fine fiddle had he;
Twee, tweedle-dee, tweedle-dee, went the fiddlers.
Oh, there's none so rare as can compare
With King Cole and his fiddlers three.
```

and you ask EDIT to list all the lines that contain the word fiddle, EDIT lists the lines:

```
Every fiddler, he had a fine fiddle
And a very fine fiddle had he;
```

If you ask EDIT to list the string fiddle, EDIT lists these lines:

```
And he called for his fiddlers three.
Every fiddler, he had a fine fiddle,
And a very fine fiddle had he;
Twee, tweedle-dee, tweedle-dee, went the fiddlers.
With King Cole and his fiddlers three.
```

**Examples** The current EDIT file for the following examples is:

```
11     Tom, Tom
12     The piper's son
13     Stole a pig
14     And away he run.
15     The pig was eat
16     And Tom was beat,
17     And Tom went crying
18     Down the street.
19     Tom, Tom
20     The piper's son
21     He learned to play
22     When he was young;
23     But all the tunes
24     That he could play
25     Was "Over the hills
26     And far away."
```

1. The command:

```
*LIST BOTH "he"
   12     The piper's son
   14     And away he run.
   15     The pig was eat
   18     Down the street.
   20     The piper's son
   21     He learned to play
   22     When he was young;
   23     But all the tunes
   24     That he could play
   25     Was "Over the hills
*
```

lists every line in the file that contains the character string *he*, *HE*, *He*, or *hE*.

2. The command:

```
*LIST BOTH WORD "he"
   14     And away he run.
   21     He learned to play
   23     When he was young;
   24     That he could play
*
```

   lists every line in the file that contains the word *he, HE, He,* or *hE*.

3. The command:

```
*LIST NOT BOTH "tom"
   12     The piper's son
   13     Stole a pig
   14     And away he run.
   15     The pig was eat
   18     Down the street.
   20     The piper's son
   21     He learned to play
   22     When he was young;
   23     But all the tunes
   24     That he could play
   25     Was "Over the hills
   26     And far away."
*
```

   lists the lines in the file that do not contain the character string *tom, Tom, TOm, TOM, tOM,* or *toM*.

4. The command:

```
*LIST "Tom" COL 1
   11     Tom, Tom
   19     Tom, Tom
*
```

   lists the lines that contain the character string *Tom*, starting in column 1.

5. The command:

```
*LIST "he" COL 3:10
   18     Down the street.
   22     When he was young;
   24     That he could play
*
```

   lists the lines that contain the character string *he* between columns 3 and 10.

6. The command:

```
*LIST WORD "as"
*
```

   lists no lines because any occurrences of *as* do not meet the word criteria explained previously in "How to Use the String-Range Parameter." (Compare with the next example.)

7. The command:

```
*LIST "as" NUM 4
   25     Was over the hill
*
```

   lists the fourth line containing an occurrence of the string *as* in the file (notice that the keyword WORD was omitted).

## String-Range-List Parameter

The string-range-list parameter is based on the string-range parameter. It allows an optional repetition of the string-range parameter when an editor command will accept it. Because the repetition of the string-range is optional, an editor command offers more flexibility when the syntax specifies a string-range- list. A string-range-list parameter can represent one or many character strings.

### Syntax of the String-Range-List Parameter

```
string-range-list: is
   string-range [, string-range ] ...
```

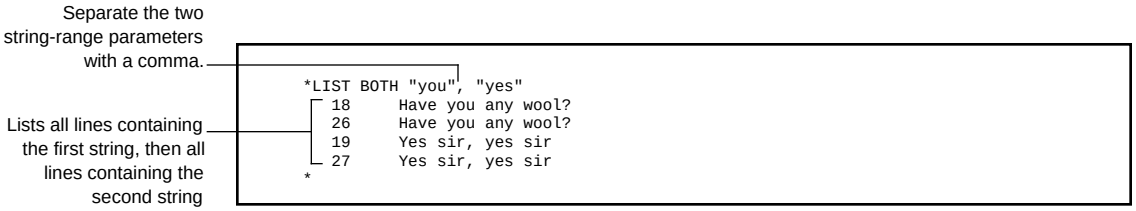
Refer to the previous presentation of the String-Range Parameter for a full description and tips on using this range parameter.

#### Examples

The current EDIT file for the following examples is:

```
17     Baa baa black sheep
18     Have you any wool?
19     Yes sir, yes sir
20     Three bags full.
21     One for the master
22     One for the dame
23     One for the little boy
24     Who lives down the lane.
25     Baa baa black sheep
26     Have you any wool?
27     Yes sir, yes sir
28     Three bags full.
```

1. The command:




lists all the lines containing either of the strings *you* or *yes*.

2. The command:

```
*LIST "ba" COL 1:7, BOTH "b"
   17     Baa baa black sheep
   25     Baa baa black sheep
   17     Baa baa black sheep
   20     Three bags full.
   23     One for the little boy
   25     Baa baa black sheep
   28     Three bags full.
*
```

lists all the lines containing the character string *ba* in the first seven columns, then lists all lines containing the character string *b*. As EDIT lists all lines that contain the first string- range of the string-range-list parameter, then all lines that contain the next string-range, and so on, some overlap might occur (as in the preceding example).

3. The command:

```
*LIST BOTH "yes" NUM 2, "a" NUM 5
   27     Yes sir, yes sir
   22     One for the dame
*
```

lists the second line that contains the string *yes* and the fifth line that contains the string *a*.

## Column-Range Parameter

The column-range parameter describes one or more columns that you indicate with numbers or a keyword.

### Syntax of the Column-Range Parameter

```
column-range: is
   column [ : column ]
```

`column`

is one of the following. You use a colon (:) to separate the beginning and ending points of the range; for example, F:20 or 12:200. The column on the left of the colon must be less than or equal to the column on the right.

```
{ column-number   }
{ FIRST           }
{ LAST            }
```

*`column-number`*

is a column number between 1 and 239; for example, 23 or 132.

`FIRST`

is column 1.

`LAST`

is the rightmost nonblank column.

### How to Use the Column-Range Parameter

- ☐ You can use the column-range independently and also as an option within the string-range-list parameter. When you give a left column number and a right column number, the right column number must be equal to or greater than the left column number (for example, 5:5 is valid, FIRST:LAST is valid, 15:7 is not valid). If you only specify a left column number, *string* must begin in that column; if you specify a left and right column number, *string* must be within the boundaries of the column.
- ☐ If you want to pinpoint a column number (for example, to determine the column where a particular string begins within a file), use the LIST COL command with a line range. LIST COL tells EDIT to list one or more lines while supplying a column number template above each displayed line. (See "LIST Command" in Section 4 for additional information.)

#### Examples

The current EDIT file for the following examples is:

```
6      As I was going to St. Ives
7      I met a man with seven wives
8      Each wife had seven sacks
9      Each sack had seven cats
10     Each cat had seven kits
11     Kits, cats, sacks, and wives
12     How many were going to St. Ives?
```

1. The command:

```
*LIST "seven" COL 18:LAST
    7     I met a man with seven wives
*
```

   lists all lines that contain the string *seven* between columns 18 and the last nonblank column.

2. The command:

```
*LIST "seven" COL 15:22
    7     I met a man with seven wives
    8     Each wife had seven sacks
    9     Each sack had seven cats
*
```

lists all lines that contain the string *seven* between columns 18 and 22.

3. The command:

Note that EDIT accepts a longer string here.

```
*CHANGE 15:19 "thirteen" 8/9
    8     Each wife had thirteen sack
    9     Each sack had thirteen cats
*
```

changes the characters in columns 15 through 19 of lines 8 and 9 to the string *thirteen*.

## Column-Range-List Parameter

The column-range-list parameter is based on the column-range parameter. It allows an optional repetition of the column-range parameter when an editor command will accept it. Because the repetition of the column-range is optional, an editor command offers more flexibility when the syntax specifies a column- range-list. A column-range-list parameter can represent a single column or several areas, delimited by pairs of column numbers, in one line of your file.

### Syntax of the Column-Range-List Parameter

```
column-range-list: is
   column-range [ column-range ] ...
```

Refer to the previous presentation of the Column-Range Parameter for a full description on using this range parameter.

#### Examples

The current EDIT file for the following examples is:

```
20     1, 2, buckle my shoe
21     3, 4, close the door
22     5, 6, pick up sticks
23     7, 8, lay them straight
24     9, 10, a big fat hen
```

1. The command:

```
*CHANGE 3 6 "  " 20/23
   20     1,   2,   buckle my shoe
   21     3,   4,   close the door
   22     5,   6,   pick up sticks
   23     7,   8,   lay them straight
*
```

   adds two blank spaces at columns 3 and 6 in lines 20 through 23.

2. The command:

```
*LIST "r" COL FIRST 3:6 9
*
```

returns a prompt because EDIT could not find the string *r* in any of the columns listed in the command line. But if you change line 23 to:

```
23     7,   8,   lay them straighter
```

and add LAST to the column-range-list in the command line:

The column-range LAST is relative within a specified line-range.

```
*LIST "r" COL FIRST 3:6 9 LAST
   21     3,   4,   close the door
   23     7,   8,   lay them straighter
*
```

EDIT lists all lines that contain the string *r* in any of the column ranges typed on the command line. In this case, EDIT found an *r* string in the LAST column range of lines 21 and 23.

## Ordinal-Range Parameter

The ordinal-range parameter describes the ordinal position of a line in an EDIT file.

The ordinal position of a line in an EDIT file refers strictly to the order of that line in the file. For example, if you specify 5 as the line number, EDIT will locate the fifth line of your file, regardless of the actual numbering sequence currently in effect in your file. EDIT views the file in terms of ordinal numbering (first, second, tenth, and so on); for other range parameters, EDIT uses the literal line numbering (line number 1, 3.01, 13.6, and so on).

With one exception (see Note, following), EDIT commands that accept the ordinal-range parameter use it as an option of the string-range parameter to reference one or more occurrences of a string in a file. EDIT locates the lines of the file that contain *string*, thus ordering them in an ordinal sequence, and is then able to locate *string* by ordinal position of the line or lines specified in the ordinal-range parameter.

For example, the command LIST "A" NUM 5 asks EDIT to locate the fifth line containing string *A*. The ordinal-range parameter is always preceded by the keyword NUM.

### Syntax of the Ordinal-Range Parameter

```
ordinal-range: is
   ordinal-position [ / ordinal position ]
```

`ordinal-position`

is the position in the file, indicated as one of the following.

```
{ number   }
{ FIRST    }
{ LAST     }
```

*number*

is an integer that refers to the position of a line in an EDIT file.

FIRST

is the first position of a string or line in the file.

LAST

is the last position of a string or line in the file.

**Note** The GET command syntax is the exception to the rule that the ordinal-range parameter is always used as an option of the string-range parameter. In the GET command syntax, you use the ordinal-range parameter to reference one or more lines in a file instead of one or more locations of a string. The ordinal-range parameter, however, when used with the GET command, still requires the keyword NUM.

**Examples** Suppose the current EDIT file named BIRD contains these lines:

```
1     The king was in his counting-house
2     Counting all his money.
3     The queen was in the parlor
3.1   Eating bread and honey.
3.2   The maid was in the garden
4     Hanging up the clothes.
4.1   Along came a blackbird
5     And snipped off her nose!
```

1. The command:

```
*LIST "the" NUM FIRST
   3      The queen was in the parlor
*
```

   lists the first line that contains the string *the*, which is line 3. Note that *The* occurs as the first word of lines 1 and 3, but without the BOTH keyword in the command line, EDIT only matched the string *the* exactly as it was typed.

2. The command:

```
*LIST "in" NUM 4/6
    3.1   Eating bread and honey.
    3.2   The maid was in the garden
    4     Hanging up the clothes
*
```

   lists the fourth, fifth, and sixth lines that contain the string *in*, which are lines 3.1, 3.2, and 4. The first four lines of the file contains the string *in* six times; note that line 1 has *in* as both the word *in* and as a string inside the words *king* and *counting*. However, EDIT is searching for the first line that contains the string, the second line, and so on; EDIT ignores how many times the string occurs on each line.

3. As the GET command can be used to move lines into your current file from other files, the GET command accepts the ordinal-range parameter to reference the ordinal positions of lines in a file rather than strings. Suppose for the following example that the EDIT file named PIE contains these lines:

```
12     Sing a song of sixpence
13.1   A pocket full of rye.
13.2   Four-and-twenty blackbirds
13.3   Baked in a pie.
14     When the pie was opened,
15     The birds began to sing.
16     Wasn't that a dainty dish
16.1   To set before the king!
```

The command:

```
*GET PIE NUM 5 TO 2
```

returns with this message:

```
LAST NEW LINE IS 2.1 <- 14
CURRENT FILE IS $WORK.POEMS.BIRD
```

This tells you that EDIT added line 14 from PIE to BIRD and that line 14 of PIE is now line 2.1 of BIRD. Remember, though, that line 14 (in ordinal position) is the fifth line of PIE and is the line specified by the NUM 5 parameter of this GET command. Now, when you list BIRD:

```
*LIST ALL
     1     The king was in his counting-house
     2     Counting all his money.
     2.1   When the pie was opened,
     3     The queen was in the parlor
     3.1   Eating bread and honey.
     3.2   The maid was in the garden
     4     Hanging up the clothes.
     4.1   Along came a blackbird
     5     And snipped off her nose!
*
```

the GET command added the fifth line of PIE to the file, regardless of its literal line number in PIE.

## Ordinal-Range-List Parameter

The ordinal-range-list parameter is based on the ordinal-range parameter. It allows an optional repetition of the ordinal-range parameter when an editor command will accept it. Because the repetition of the ordinal-range is optional, an editor command offers more flexibility when the syntax specifies an ordinal- range-list. An ordinal-range-list parameter can represent one or more lines in your file.

### Syntax of the Ordinal-Range-List Parameter

```
ordinal-range-list: is
   ordinal-range [ ordinal-range ] ...
```

Refer to the previous presentation of the Ordinal-Range Parameter for a full description.

### Examples

Suppose the current file named SIXPENCE contains these lines:

```
12     Sing a song of sixpence
13.1   A pocket full of rye.
13.2   Four-and-twenty blackbirds
13.3   Baked in a pie.
14     When the pie was opened,
15     The birds began to sing.
16     Wasn't that a dainty dish
16.1   To set before the king!
16.2
17     The king was in his counting-house
18     Counting all his money.
20     The queen was in the parlor
21     Eating bread and honey.
22     The maid was in the garden
23     Hanging up the clothes.
24     Along came a blackbird
25     And snipped off her nose!
```

1. The command:

Separate the ordinal-ranges with spaces.

```
*LIST "k" NUM F 3 6
   13.1   A pocket full of rye.
   13.3   Baked in a pie.
   24     Along came a blackbird
*
```

lists the first, third, and sixth lines that contain the string *k* in the text. EDIT ignores the line numbers when interpreting an ordinal-range or ordinal-range-list parameter.

2. The command:

Separates the lower and higher line numbers defining an ordinal-range.

```
*LIST BOTH "the" NUM 2 4/L
   15     The birds began to sing.
   17     The king was in his counting-house
   20     The queen was in the parlor
   22     The maid was in the garden
   23     Hanging up the clothes.
*
```

lists the second then the fourth through last lines that contain the string *the* (qualified by the keyword BOTH, so EDIT ignores case when searching for the string).

3. As the GET command can be used to move lines into your current file from other files, the GET command accepts the ordinal-range-list parameter to reference the ordinal positions of lines in a file rather than strings. Suppose for the following example that the EDIT file named TART contains these lines:

```
33     The Queen of Hearts
34     Baked some tarts
35     All on a summer's day.
36     The Knave of Hearts
37     He stole those tarts
38     And took them clean away.
```

The command:

```
*GET TART NUM F/2 6 TO 21
```

replies with the message:

```
LAST NEW LINE IS 21.3 <- 38
CURRENT FILE IS $WORK.POEMS.SIXPENCE
```

This tells you that EDIT added the first, second, and sixth line (in ordinal position) of TART to SIXPENCE and that line 38 of TART (the last line added from TART) is now line 21.3 of SIXPENCE. Remember, though, that when EDIT locates lines of a specified ordinal-range parameter, it ignores the literal line numbering.

Now, when you list SIXPENCE:

```
*LIST ALL
   12     Sing a song of sixpence
   13.1   A pocket full of rye.
   13.2   Four-and-twenty blackbirds
   13.3   Baked in a pie.
   14     When the pie was opened,
   15     The birds began to sing.
   16     Wasn't that a dainty dish
   16.1   To set before the king!
   16.2
   17     The king was in his counting-house
   18     Counting all his money.
   20     The queen was in the parlor
   21     Eating bread and honey.
   21.1   The Queen of Hearts
   21.2   She made some tarts
   21.3   And took them clean away.
   22     The maid was in the garden
   23     Hanging up the clothes.
   24     Along came a blackbird
   25     And snipped off her nose!
*
```

you can see that the GET command inserts three lines of TART to the file at line 21.

## Range-Specifier Parameter

The range-specifier parameter allows you to indicate combinations of line-range and string-range parameters.

### Syntax of the Range-Specifier Parameter

```
range-specifier

   { line-range-list [ , string-range-list ] |
   string-range      [ line-range-list     ] }
      [ RANGE { line-range-list |
      string-range [ line-range-list ] } [ , ... ] ]


line-range-list:

   line-range [ line-range] ...


string-range-list:
   string-range [ string-range] ...
```

`string-range`

specifies one or more strings. See the String-Range parameter.

RANGE

is a keyword that tells EDIT to look for the specified *string*

in the line-range-list or string-range that follows the keyword. RANGE and the range-list that follows it cannot stand on its own; these elements always refer to the line-range or *string* that precedes the keyword.

**Note** When you specify a repetition of this range parameter that includes the keyword RANGE, you must use a comma before the keyword.

### How to Use the Range-Specifier Parameter

Because the range-specifier parameter allows you to use the line-range-list and string-range-list parameters, the following tips highlight specific information you need to keep in mind when using any of the range-list parameters.

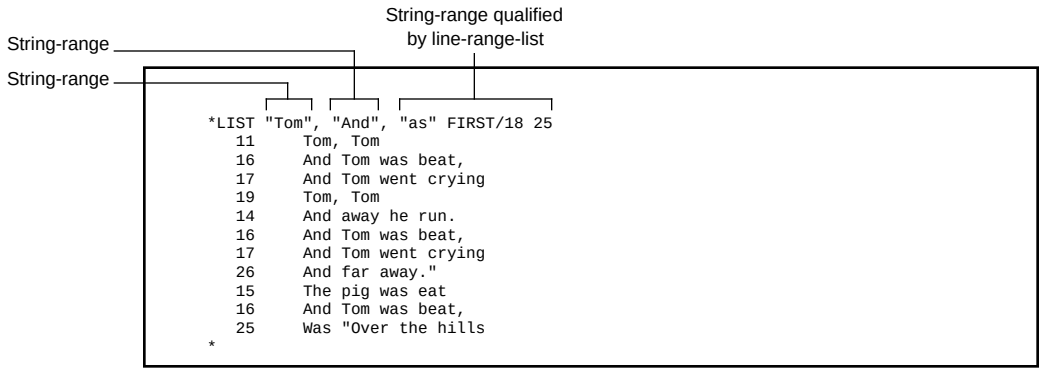
- ☐ Separate each line-range in a line-range-list with a space.
- ☐ When you specify a line-range-list parameter, you can specify single line numbers in any order, but you must specify the beginning-line and ending-line of blocks of text in ascending order (lower to higher).
- ☐ The string-range parameter allows you to use the keyword WORD whenever you want EDIT to find only the words that match *string*. A word is defined as (1) any character string that is preceded and followed by a space or any character other than a number or letter or (2) a character string that occurs at the end of a line. (See Example 2.)
- ☐ A line-range-list following a string-range qualifies that string- range. (See Examples 1 and 3.)
- ☐ In the range-specifier syntax, the comma between the line- range-list and the subsequent string-range defines these two elements as separate; EDIT executes one independently of the other. (See Example 4.)
- ☐ Rather than locating strings on specific numbered lines with the string-range parameter, you can use the NUM ordinal-range- list option to locate one or more strings in your file by the ordinal position (first, third, tenth, and so on) of the line or lines containing that string. The ordinal-range is always used as an option of the string-range parameter. See the Ordinal-Range description for more information.

- You can use the column-range-list independently and also as an option within the string-range-list parameter. When you use a column-range by specifying a left column number and a right column number, the right column number must be equal to or greater than the left column number (for example, 5:5 is valid, FIRST:LAST is valid, 15:7 is not valid). If you only specify a left column number, *string* must begin in that column; if you specify a left and right column number, *string* must be within the boundaries of the column.
- If you want to pinpoint a column number (for example, if you need to determine the column where a particular string begins within a file), use the LIST COL command with a line range. LIST COL tells EDIT to list a column number template above each displayed line. (See the LIST command in Section 4 for more information.)

**Examples** The current EDIT file for the following examples is:

```
11     Tom, Tom
12     The piper's son
13     Stole a pig
14     And away he run.
15     The pig was eat
16     And Tom was beat,
17     And Tom went crying
18     Down the street.
19     Tom, Tom
20     The piper's son
21     He learned to play
22     When he was young;
23     But all the tunes
24     That he could play
25     Was "Over the hills
26     And far away."
```

1. The command:




is performed in this order:

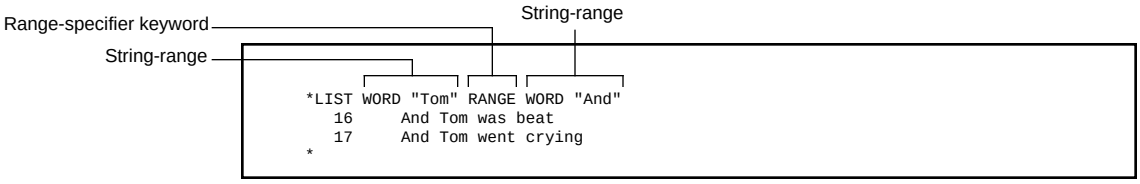
- ☐ LIST “Tom”
- ☐ LIST “And”
- ☐ LIST “as” FIRST / 18 25

EDIT first lists all the lines that contain the strings *Tom* and *And*, then lists all the lines that contain the string *as* which fall within the specified line-range.

As EDIT works sequentially through the options of the range-specifier parameter, satisfying the first one then moving on to the next one, the results of a command might cause lines to appear more than once (as is, for example, the case here).

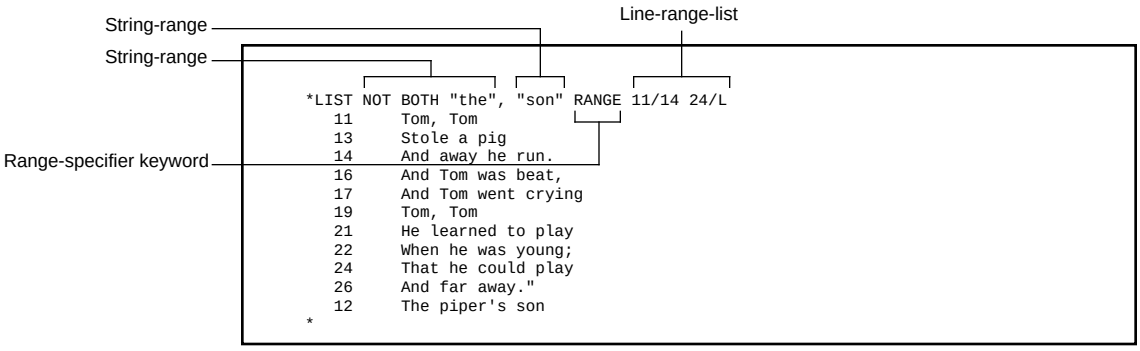
Compare the results of this command with the results of Example 2.

2. The command:




lists the lines that contain both the words *Tom* and *And*. The RANGE keyword and the string-range "And" work together to specify the range criteria for this LIST command.

3. The command:



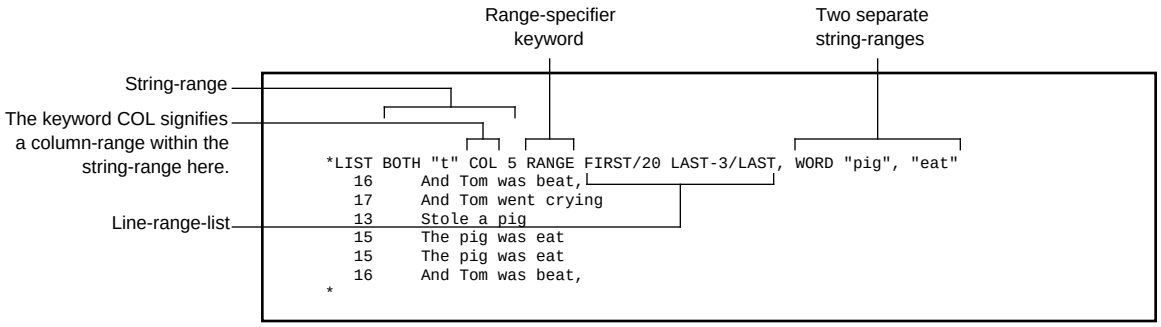

is performed in this order:

☐ LIST NOT BOTH "the"

☐ LIST "son" RANGE 11/14 24/L

EDIT lists the lines throughout the file that do not contain the character string *the*, *The*, *THe*, *THE*, *tHE*, or *thE*, then lists the lines within the specified line-range (lines 11 through 14 and 24 through the last line) that contain the string *son*.

4. The command:

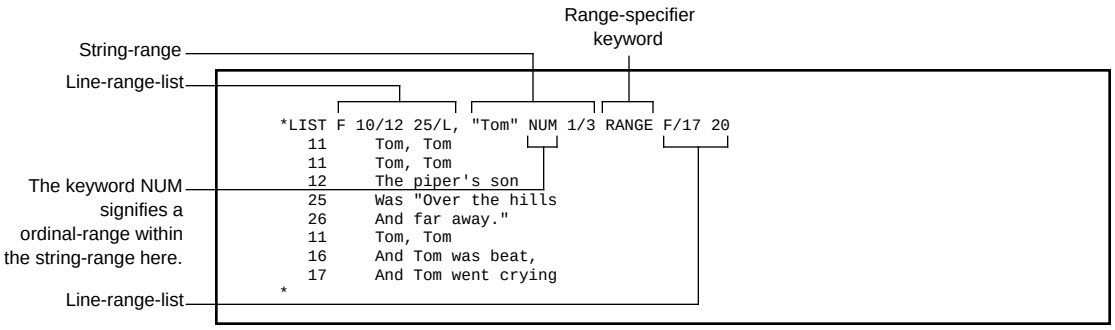



is performed in this order:

- ☐ LIST BOTH “t” COL 5 RANGE FIRST / 20 LAST-3 / LAST
- ☐ LIST WORD “pig”
- ☐ LIST “eat”

EDIT first lists the lines from the first line through line number 20 and the third from the last line through the last line that have the string *t* or *T* in column 1. EDIT then lists all lines that contain the word *pig* and finally lists all lines that contain the string *eat*.

5. The command:




is performed in this order:

- ☐ LIST F 10/12 25/L
- ☐ LIST "Tom" NUM 1/3 RANGE F/17 20

EDIT first lists the text on the first line, lines 11 and 12, and lines 25 through the last line of the file (note there is no line 10, so EDIT cannot list it). It then lists the first, second, and third lines that contain the string *Tom* within the range of the first line through line 17 as well as line 20.

# 6 Handling Your EDIT Files

When you are using the EDIT program, you are modifying an existing EDIT file or are creating a new file to add text to it. As you work with your EDIT files, you should keep several points in mind:

- When you use EDIT to modify an existing EDIT file, changes are made to the actual source file, not to a copy of the file.
- You use the EDIT program to create EDIT files.
  - The file code for EDIT files is 101.
  - EDIT file line numbers range from 0 to 99999.999.
  - Each line of text contains 0 to 255 printable characters.
- You use the QUERY editor command to determine the amount of unused space in an EDIT file; you use the PUT command to decrease the amount of unused space.

## Creating a Backup Copy of Your EDIT File

When you use EDIT to modify an existing EDIT file, your additions and changes are made to the actual source file, not a copy of the file. Therefore, for your own protection, you should use the PUT command to make a backup copy of an EDIT file when you plan to make extensive additions or changes to it. Having a backup copy can make it easier for you to recover from costly errors, such as deleting a large portion of text by mistake.

Making a backup copy of an EDIT file means that you are asking EDIT to duplicate the file and put it in another file location, just in case you might need it.

To make a backup copy of the EDIT file named BEN, type:

```
EDIT BEN; PUT BENSAVE
```

This command line starts the EDIT program on the file named BEN and puts a copy of the text from BEN in the file named BENSAVE.

```
33> EDIT BEN; PUT BENSAVE
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.NONFICTION.BEN
*
```

See Section 4 if you would like more information and examples of the PUT command.

## EDIT Files

An EDIT file is a specially formatted disk file. When you use the EDIT program to create a disk file, it always creates an EDIT file and assigns the file a file code of 101.

### How EDIT Files Are Named

EDIT files have names. You enter the full name of an EDIT file in this format:

```
$volname.subvolname.filename
```

Partial names of an EDIT file are:

```
subvolname.filename
```

and

```
filename
```

- *Volname* is the name of a disk pack mounted on a disk drive. A *volname* begins with a dollar sign ($), followed by from one to seven alphanumeric characters, and the first character must be alphabetic. System operators name the disk packs.
- *Subvolname* is the name of a set of related files on a disk pack. A *subvolname* can be from one to eight alphanumeric characters, and the first character must be alphabetic. You create and name your own subvolumes.
- *Filename* is the name of a particular EDIT file within a subvolume on a volume. A *filename* can be from one to eight alphanumeric characters, and the first character must be alphabetic. You create and name your own EDIT files.

The command that starts EDIT (the EDIT command) accepts a full or a partial filename as a parameter. When you give a partial filename as a parameter to the EDIT command, it uses the names of your current default volume and subvolume to expand the partial filename into a full filename. If you need more information about volumes, subvolumes, and files, see the *GUARDIAN 90 Operating System Utilities Reference Manual*.

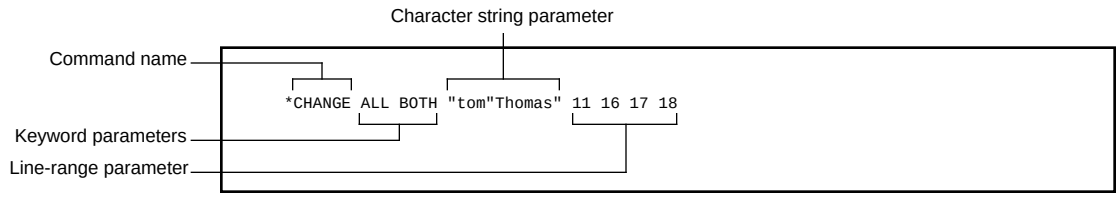
## Line Numbers in EDIT Files

An EDIT file contains lines of ASCII text that you enter at your terminal. Recall that EDIT assigns each line of text a line number according to your specifications. A line number has from one to five digits followed by an optional decimal point and zero, one, two, or three digits of a fraction. Valid line numbers in an EDIT file are 0 through 99999.999. Here are some examples:


When you use the EDIT program:

- ☐ The terminal screen displays the line number of each line of text in the EDIT file. Figure 1 shows what you see on the screen when you use the EDIT program.
- ☐ Most editor commands require you to specify a line number or range of lines on which you want the commands to operate.

For example:



See Section 5, "Range Summary," for an introduction to ranges.

- You can run out of line numbers. For example, if your text is numbered:

```
23.101 When widows exclaim loudly against second marriages,
23.102 I would always lay a wager that the man,
23.103 if not the wedding day,
23.104 is absolutely fixed on.
```

and you want to add text between line number 23.102 and 23.103, you cannot do it because there is no line number available. When this happens to you, you must renumber your EDIT file. (See the NUMBER command in Section 4.)

## Text Lines and Printable Characters

Each text line in an EDIT file contains from zero characters to a maximum of 239 to 255 printable characters. (Because of the way in which EDIT stores characters internally, the maximum number of characters that fit on a text line depends on the number and distribution of the blank spaces in the line.) A line with zero characters is blank.

Any nonprintable character inserted into an EDIT file may adversely affect EDIT. Therefore, you should insert only printable characters into an EDIT file. The printable characters are:

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
{ } ! @ # $ % ^ & * ( ) - _ +
= [ ] ~ " ` : ; ? / > . < ,
blank space

EDIT identifies each character position in a line by a column number; the first column (on the left) is column 1, the last is column 255. You do not see column numbers on the screen when you use EDIT. However, the LIST command with the COL keyword will display column numbers on the screen. (The LIST command is described in Section 4.)

While it is true that a text line can contain up to 255 characters, terminal screens are only 80 characters wide. Because EDIT does not allow you to move the screen window left and right, text lines that are longer than 80 characters occupy more than one line on the screen. Figure 6-1 shows how lines longer than 80 characters look when you use the EDIT program.

### Continuation Lines in the EDIT Program

In EDIT, you create a continuation line (a line longer than 80 characters) by typing to the end of the screen and then allowing the cursor to wrap to the next screen line. Figure 6-1 shows two continuation lines: line number 9 continues onto two screen lines, and line number 11 continues onto three screen lines.

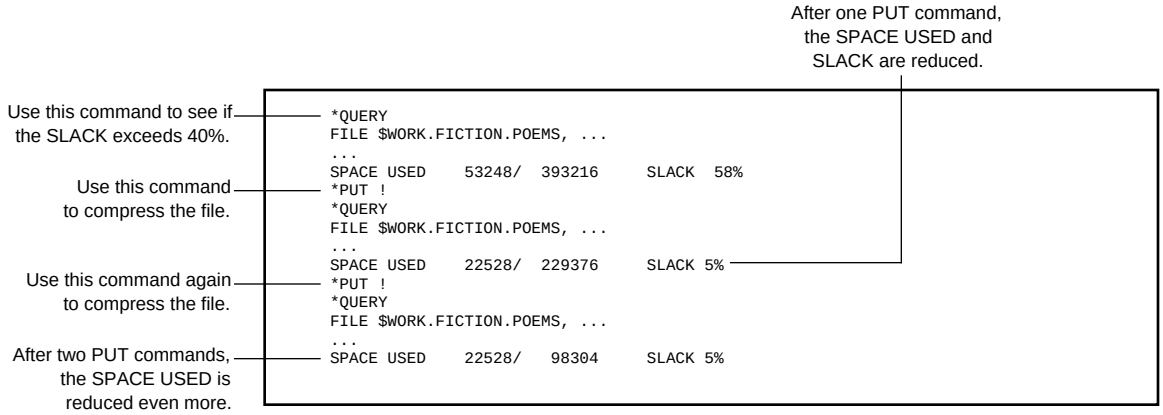
**Figure 6-1. Continuation Lines in EDIT**

```
    8     He that goes a borrowing goes a sorrowing.
    9     Dost thou love life?  Then do not squander time, for that is the stuff
life is made of.
   10     Little strokes fell great oaks.
   11     The next thing most like living one's life over again seems to be a rec
ollection of that life, and to make that recollection as durable as possible by
putting it down in writing.
```

## Compressing Space in an EDIT File

EDIT makes EDIT files recoverable from process failures. To do this recovery, EDIT copies the modified lines of text into a new area in the disk file as you make modifications to an EDIT file. EDIT can reuse portions of the disk file that have been obsoleted as a result of editing. However, portions of the file that are obsolete often remain on disk, taking up space.

To determine the percentage of an EDIT file that is unused, you use the QUERY command. If the unused portion (SLACK) is greater than 40%, you should use two PUT commands to compress the file. The first PUT command compresses the file. The second PUT command maximizes the compression. For example:




For more information and examples, see the PUT and QUERY commands in Section 4.

# Appendix A EDIT Error Messages

## EDIT Error Messages

This appendix lists the EDIT error messages in alphabetical order. A brief explanation of the message and a reference to helpful information accompanies each error message.

Most of the EDIT program error messages you will receive tell you that you incorrectly entered an editor command. You will find that some of these error messages are clear and concise about what you did wrong, while others are not. When you receive an error message that you do not understand, don't panic if it is not explained here. Turn to the description of EDIT command that produced the error message, compare what you entered with the examples of that command, then try the command again.

```
filename FILE HAS EXCEEDED AVAILABLE DISC SPACE
```

There is not enough space on this file's disk volume to allocate the extents for the file. Report this error message to your system operator. To save the editing you've done in your file, use the PUT command to put the text in the file into a new file on another volume.

```
filename IS NOT AN EDIT TYPE FILE
```

You invoked the EDIT program on a file that does not have a 101 (EDIT) or 102 (TTEXT) file code. You can use the GET command to create an EDIT file from an unstructured file with fixed-length records. See the GET command description in Section 4 for help.

```
file1 IS NOT A VALID EDIT FILE
```

You attempted to insert text from *file1* into *file2*. However, something is wrong with *file1*. To solve this problem, type:

```
*GET file1 PUT newfile1
```

This command creates a new EDIT file, puts the text from *file1* into it, and makes the new EDIT file the current file. Then type:

```
*GET file2
```

This command makes *file2* the current EDIT file. Now try inserting text from *newfile1* into *file2*.

```
A CHARACTER STRING IN QUOTES IS EXPECTED HERE
```

You incorrectly entered one of several editor commands that require you to specify a character string. Try the command again or see the appropriate command description in Section 4 for help.

```
A LIST OF ORDINAL ELEMENTS IS EXPECTED HERE
```

You incorrectly specified the ordinal elements following the NUM or COL keyword in a command or range parameter or you incorrectly specified the column range for the CHANGE command. Try again or see the appropriate command or range parameter description.

```
AN ERROR HAS TERMINATED THE 'IN' FILE
AN ERROR HAS TERMINATED THE 'OBEY' FILE
```

An error terminated the execution of a command file. See the OBEY command description for Section 4 for help.

```
AN I/O ERROR OCCURRED ON THE OUTPUT FILE
```

An I/O error terminated the EDIT command. Try the command again.

```
AN I/O ERROR TERMINATED THE 'PUT' OPERATION
```

An I/O error terminated the PUT command. Try the command again.

```
AN I/O ERROR PREVENTS PURGING THE OLD COPY
```

An I/O error terminated the command. Try the command again.

```
AN INCREMENT SMALLER THAN .001 WOULD BE NEEDED
```

You tried to add a line of text to an EDIT file but there is no line number available for you to do so. Renumber lines, then try again. See "Renumbering to Accommodate Added Lines" in Section 3 and the GET command description in Section 4 for help.

```
AN INCREMENT SMALLER THAN .001 WOULD BE NEEDED
ATTEMPTING TO ADD LINE linenumber
LAST LINE TO BE ADDED FROM SOURCE FILE IS linenumber
```

You were using the GET command to add text to an EDIT file when the available line numbers ran out before all the text was added. See the examples in the GET command description, Section 4, for instructions on how to recover.

```
CAN'T GET AND PUT TO SAME FILE IN ONE COMMAND
```

You incorrectly specified the GET command. See the GET command description in Section 4 for help.

```
COMMUNICATION FAILURE (FILE ERROR = errornum)
```

You were using the TEDIT command to start the TEDIT editing program when either EDIT had a communication problem with the startup message to TEDIT or EDIT had a problem with the TEDIT process once it was started. This is an internal error that you cannot correct. Try the command again, or tell your system operator.

```
CPU FAILURE: xx
```

Your file should be undamaged after a processor failure. Just start editing again. If you re-enter EDIT and receive the FILE IS INVALID message, follow the EDIT Error Recovery Procedure in Appendix B.

```
EXCLUSIVE ACCESS TO THIS FILE IS NOT AVAILABLE
```

You receive this message when the EDIT program cannot exclusively access a file, for example in the OUT option of the LIST command. Wait a few minutes, then try the command again.

```
{ filename } FILE ERROR: errornum
{     ??   }
```

An unrecoverable error occurred with the file named *filename*. *errornum* is the file system error number.

```
LINE line EXCEEDS 'OUTLEN' BY num CHARACTERS
```

You receive this message when the SET OUTLEN command is not set large enough to accommodate the number of characters in the line.

```
LINE line IS IN THE WAY
```

Renumber lines, then try again. See “Renumbering to Accommodate Added Lines” in Section 3 for details.

```
LINE line WOULD EXCEED 255 CHARACTERS
```

Text lines are limited to 255 characters. EDIT prints this message whenever an editor command results in a text line that exceeds 255 characters. EDIT truncates the text line to 255 characters.

```
NOT ENOUGH MEMORY FOR THIS FILE
```

There is not enough memory available for EDIT to execute the current editor command. Divide the text in the file into two files. Then, try the command again.

```
$devname NOT READY
```

The device named *$devname* is not ready. To continue, enter a carriage return after making the device ready. To abort the command, type:

```
1 STOP
```

```
ONLY 3 DIGITS ARE PERMITTED AFTER THE DECIMAL
```

You incorrectly entered a line number. See “Line Numbers” in Section 1 for help.

```
$devname PAPER OUT
```

The device named *$devname* is out of paper. To continue, enter a carriage return after loading paper. To abort the command, type:

```
1 STOP
```

```
PROCESS CREATION FAILURE (NEWPROCESS ERROR = errornum)
```

You were using the TEDIT command to start the TEDIT editing program when EDIT had a problem creating the process that would start TEDIT. This is an internal error that you cannot correct. Try the command again, or tell your system operator.

```
SORRY, THIS FEATURE HAS NOT YET BEEN INSTALLED
```

EDIT was unable to start the TEDIT program because TEDIT is not yet installed on your system.

```
TABULATION HAS OVERFLOWED A LINE
```

The expansion of tab characters to multiple blanks results in a line that exceeds 255 characters. EDIT truncates the line to 255 characters.

```
TEXT LOST IN PHYSICAL PAGE
```

This message states that you have lost some text from an EDIT file. There is no way to recover; you must re-enter the lost text.

```
THE 'BOTH' & 'WORD' OPTIONS ARE INVALID HERE
```

You used a CHANGE command incorrectly. See the CHANGE command description in Section 4 for help.

```
THE CURRENT TEXT FILE IS TEMPORARY
```

You did not name the current EDIT file. (The GUARDIAN 90 operating system purges temporary EDIT files when the temporary file is no longer the current file or when you exit from EDIT.) See the ADD command description in Section 4 for help.

```
THE NAMED FILE ALREADY EXISTS
```

You responded to a "NAME THE NEW FILE" prompt with the name of an EDIT file that already exists. See the examples in the ADD command description in Section 4 for help.

```
THE RENUMBERING WOULD OVERLAP EXISTING LINES
```

You attempted to move text to existing line numbers or you attempted to renumber lines to line numbers that already exist. See the MOVE command or the NUMBER command in Section 4 for help.

```
THIS CHARACTER STRING IS NOT TERMINATED
```

You issued one of the many editor commands that accept a character string argument and you did not terminate the character string with a string separator. See the appropriate command description for help.

```
THIS COMMAND ONLY WORKS INTERACTIVELY
```

EDIT must be running in an interactive mode (rather than noninteractively or completing an OBEY command) to be able to start TEDIT. This means that the IN and OUT parameters of the command that initially starts EDIT must name the same device, which must be a terminal. By default, both parameters name your terminal when you type EDIT at your command interpreter.

If you get this message, ensure that EDIT is in an interactive mode, then try the TEDIT command again. (See "Running the EDIT Program" in Section 4 for more information about the syntax of the EDIT command.)

```
THIS FILE CANNOT BE OPENED (FILE ERROR = errornum)
```

You were using the TEDIT command to start the TEDIT editing program when EDIT had a problem opening the process that would start TEDIT. This is an internal error that you cannot correct. Try the command again, or tell your system operator.

```
THIS FILE IS FULL - 'PUT' IN ANOTHER FILE
```

The physical disk file is full. You must use the PUT command to put the text into another file, possibly on another volume. See the PUT command in Section 4 for help.

```
THIS FILE IS INVALID, SYNDROME n LOGICAL PAGE i j DO YOU WANT TO RECOVER?
```

When executing a GET command, EDIT found internal errors in the text file. This is normal if a disk containing the file, the processor running EDIT, or the EDIT process malfunctioned. Turn to the EDIT Error Recovery Procedure in Appendix B for instructions on how recover.

```
THIS FILE'S FILECODE IS NOT SOURCE TYPE
```

You attempted to retrieve text from a file that is not an EDIT file (a file with a 101 file code). You can retrieve text from non-EDIT format files but only by using the PUT keyword with the GET command. See the GET command description in Section 4 for help.

```
THIS LINE CANNOT BE BROKEN UP SHORT ENOUGH
```

During a JOIN operation, EDIT encountered a line that did not have a blank character between column one and the specified join width. Consequently, the JOIN command, which breaks and joins lines only at blank characters, did not shorten the line.

```
THIS RANGE IS TOO COMPLEX FOR ME TO HANDLE
```

You specified a range-specifier parameter with more than six RANGE keywords or you specified a range expression that was larger than the EDIT program expected a range to be. See the range-specifier parameter description in Section 5 for help.

```
TOO MANY ASSOCIATIVE CONDITIONS WERE SPECIFIED
```

See "THIS RANGE IS TOO COMPLEX FOR ME TO HANDLE," above.

```
TOO MANY TAB STOPS
```

You used the SET TABS command to set more than 20 tab stops. See the SET command description in Section 4 for help.

```
TRY EDIT /MEM 32/
```

Your EDIT process used all its available data space. This rarely happens. When it does happen, exit from EDIT. Then type:

```
1> EDIT filename /MEM 32/
```

where *filename* is the name of the file you were editing when the error occurred. This command tells the command interpreter or TACL process to allocate more data space for your EDIT process.

```
TRY EDIT /MEM 64/
```

Your EDIT process used all its available data space. This rarely happens. When it does happen, exit from EDIT. Then type:

```
1> EDIT filename /MEM 64/
```

where *filename* is the name of the file you were editing when the error occurred. This command tells the command interpreter or TACL process to allocate more data space for your EDIT process.

# Appendix B EDIT Error Recovery Procedure

## Error Recovery

EDIT is programmed to keep your EDIT file correct and to remember all your editing, whenever possible. Annoyances that you will encounter when editing include:

- ☐ Mistyping editor commands. When you misspell an editor command or type an incorrect command syntax, EDIT displays an error message that tells you what you did wrong. Simply retype the command. (Appendix A lists the EDIT error messages.)
- ☐ Mistyping text when using the EDIT program. When using the EDIT program, you cannot cancel modifications you make to the text in the file. You must manually fix any mistakes you make.

  Occasionally, you may need to deal with a problem that requires you to follow a specific recovery procedure in order to save your editing. The remainder of this appendix describes that procedure.

## Recovery Procedure

You can perform one recovery procedure from the EDIT program. When you receive an error message, look up the message in Appendix A, "EDIT Error Messages." If you receive the following error message:

```
THIS FILE IS INVALID.  DO YOU WANT TO RECOVER?
```

follow this error recovery procedure:

1. Respond with "no" or "n".
2. Exit the EDIT program by typing:

   ```
   *EXIT
   ```

3. Use the FUP DUP command at the command interpreter to duplicate the file to another file that you specify. For example:

   ```
   FUP DUP SHAKE,SAVE
   ```

4. Reenter the EDIT program and edit the original file. For example:

   ```
   EDIT SHAKE
   ```

5. This time, when EDIT displays the message:

   ```
   THIS FILE IS INVALID.  DO YOU WANT TO RECOVER?
   ```

   respond with a "yes" or "y".

6. Check the file. If there is no problem with the file, purge the duplicate file that was not recovered after you exit the EDIT program. If there is a problem with the file, send copies of the file, error messages, and any related information to Tandem.

# Appendix C Page Mode Editing

This appendix describes the following:

- ☐ The screen editing capabilities of the EDIT VS program
- ☐ Two EDIT commands, ADD BLOCK and REPLACE BLOCK, that mimic page mode editing
- ☐ XEQ, the EDIT command that starts the EDIT VS program.

The EDIT VS program is detailed in the following paragraphs; see "EDIT Commands Requiring Full-Screen Terminals" later in this appendix for descriptions of the three EDIT commands.

## Introduction to EDIT VS

EDIT VS is a program that works with EDIT. EDIT VS performs many of the same functions as EDIT plus a few that EDIT does not. In EDIT VS, all of the functions are linked to function keys. You communicate with EDIT VS by pressing one of the 16 numbered function keys located across the top of the terminal keyboard. When you press a function key, the function occurs at the current line (the line containing the cursor).

EDIT VS is a page mode editor. It works with a full screen of text, which contains 24 80-character lines. EDIT VS buffers (or stores) up to five screens of text. As you press function keys to perform various edit functions, EDIT VS takes lines or pages out of the EDIT file as needed and returns those not used.

### What Is Screen Editing?

When you use the screen editor, you sit down at a terminal and type text onto the screen as if it were a piece of paper. You press specific keys on the terminal keyboard to:

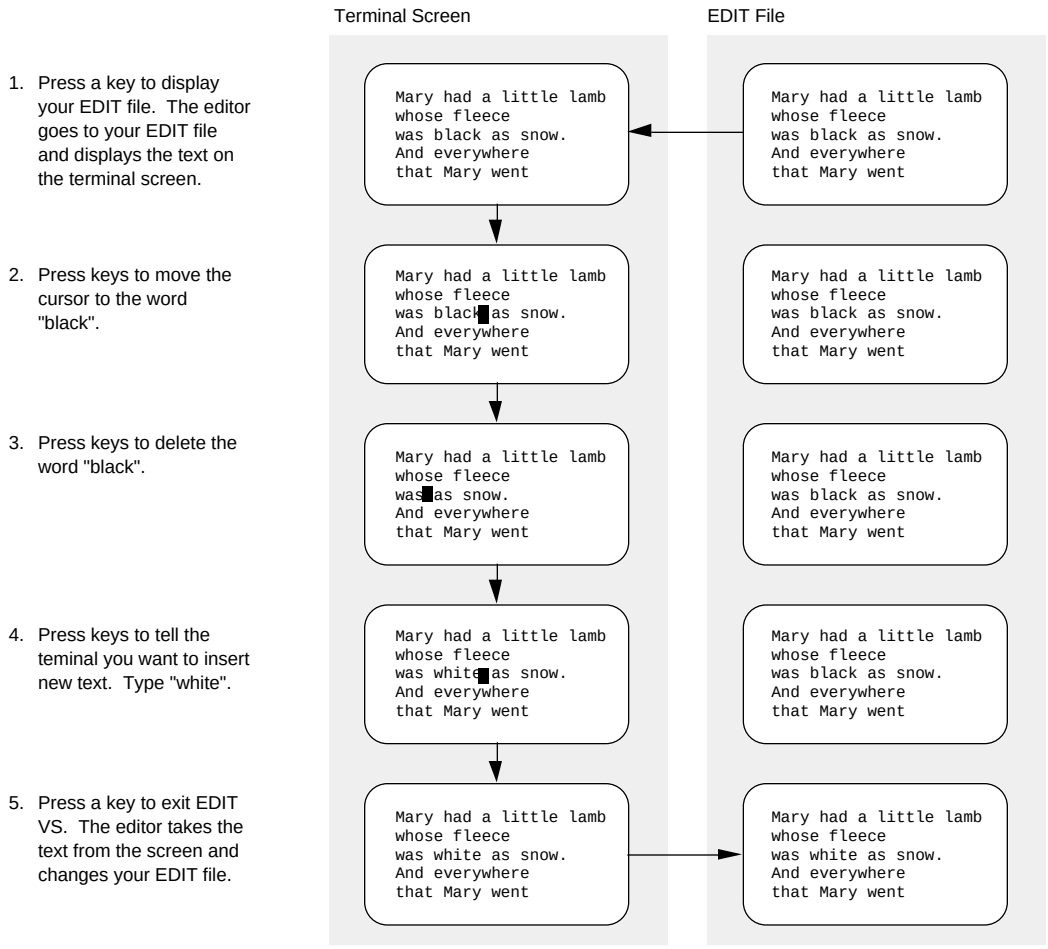
- ☐ Control the text on the screen (for example, insert blank lines)
- ☐ Perform editing functions (for example, delete characters)
- ☐ Move the cursor

As you use the various keys, notice the following:

- ☐ The action takes place at the cursor.
- ☐ You choose the 24 lines of text that appear on the screen by pressing keys.
- ☐ You control the lines of text on the screen by pressing keys.
- ☐ You change the text in your EDIT file by typing over the text on the screen.
- ☐ You do not see line numbers on the screen.

The EDIT VS program reads the text on the terminal screen and changes the text in your EDIT file to match the text on the screen. Figure C-1 illustrates the concept of screen editing.

**Figure C-1. Screen Editing**



When you press a function key that causes EDIT VS to read the screen, EDIT VS stores any changes on the screen in a "recovery" file. Should ten minutes elapse without your pressing a function key, EDIT VS reads the screen and stores changes in the recovery file. This procedure makes it possible for you to recover text if there is a failure. (See Appendix E, "EDIT VS Error Recovery Procedures.")

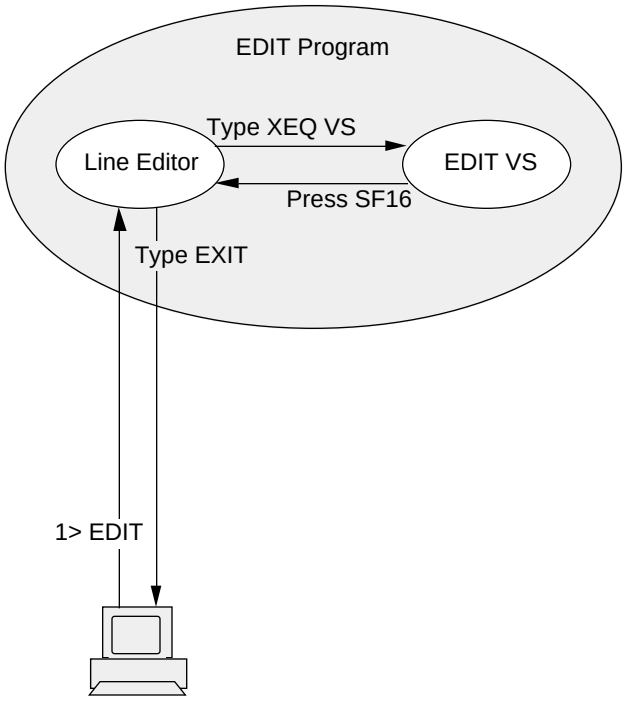
EDIT VS has a buffer stack (or holding area) for lines deleted or copied from the text. The buffer stack holds up to 24 lines. The DELETE and COPY function keys always push lines onto the stack. (Sometimes the JOIN and BREAK functions push lines onto the stack.) The RETRIEVE function keys take lines from the stack (retrieving the most recently added line or lines first), put them back into the file, and display them on the screen.

## Relationship Between the Line Editor and the Screen Editor

Figure C-2 shows the relationship between the line editor and the screen editor. Notice that the screen editor is called from the line editor. You can easily switch from the line editor to the screen editor and from the screen editor to the line editor. You can take full advantage of the features of each editor by switching back and forth between editors.

EDIT VS communicates constantly with EDIT. For example, EDIT VS relies on EDIT to assign line numbers to the text in your EDIT file.

**Figure C-2. Relationship Between the Line Editor and the Screen Editor**



## EDIT VS and Your Terminal

Your terminal must be able to operate in full-screen page mode to use EDIT VS. The Tandem 6530 is such a terminal, although EDIT VS works the same way on other Tandem terminals with full-screen capabilities. Here the 6530 terminal keyboard illustrates the various keys you can use when running EDIT VS at your terminal.

For more information about the Tandem 6530 terminal, refer to the *653X Multi-Page Installation and Operation Guide*.

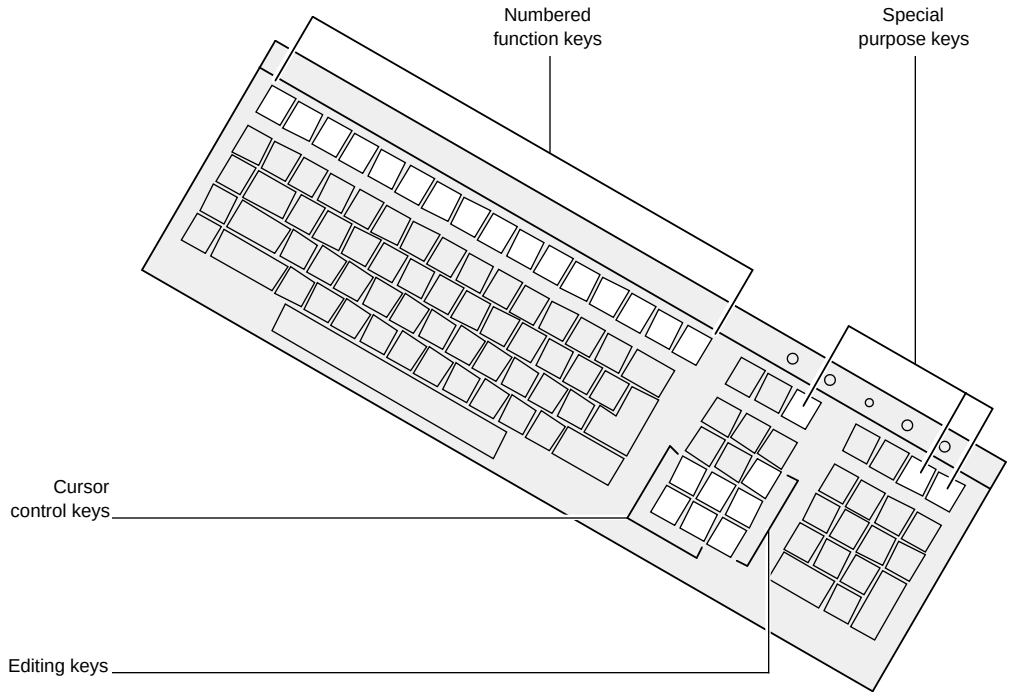
With the keyboard of your terminal, you use the keys to control the functions of your terminal and to communicate with the computer. The keys you see on your terminal keyboard can include:

- *Alphanumeric keys* are the standard typewriter-style keys used to enter alphabetic and numeric characters.
- The *numeric key pad* is organized in the standard adding- machine layout and used when large amounts of numeric data need to be rapidly entered from the keyboard.
- *Cursor control keys* are used to position the cursor anywhere on the screen.
- *Editing keys* are used to insert or delete one or more characters or lines on the screen.
- *Special purpose keys* are unrelated keys such as BREAK, PRINT, and RESET that provide your terminal with expanded capabilities and control.
- *Numbered function keys* are used to perform whatever function they are assigned by the program you are running on the computer.

When you use EDIT VS:

- You press the *cursor control keys* to position the cursor on the screen.
- You press a *numbered function key* or an *editing key* to perform an editor function at the cursor. Figure C-3 illustrates the keyboard of a Tandem 6530 terminal and highlights the various keys that work with EDIT VS.

**Figure C-3. The 6530 Terminal Keyboard**




The cursor control keys and the editing keys are described in the 653X terminal manual listed previously. The numbered function keys and EDIT functions they trigger are described in numerical order in this appendix. In addition, the template that defines the numbered function keys (and allows you to avoid memorizing the function linked to each key) is described and illustrated in the "EDIT VS Template" discussion, which appears later in this appendix.

## Starting EDIT VS

To use EDIT VS, use EDIT to read or create an EDIT file. Then invoke EDIT VS with the following EDIT command:

```
XEQ VS [ line-range ]
```

(See the XEQ command, described later in this appendix, for more specifics about this command.)

So, for example, to start the EDIT program and create a new EDIT file named AESOP, type:

```
1 EDIT AESOP
TEXT EDITOR - T9601B30 - (08MAR87)
$WORK.FICTION.AESOP DOES NOT EXIST. SHALL I CREATE IT? yes
CURRENT FILE IS $WORK.FICTION.AESOP
*
```

$WORK.FICTION.AESOP means there is a file named AESOP, in the subvolume named FICTION, on the volume named $WORK. The file named AESOP that you just created will have the volume name of your default volume and the subvolume name of your default subvolume.

You now have an empty EDIT file named AESOP, and EDIT is waiting for you to type a command. At the prompt, type the command that starts EDIT VS:

```
*XEQ VS
```

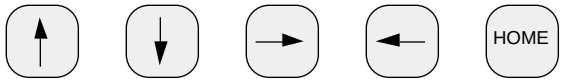
As you start EDIT VS to begin working in a new file, your screen is empty except for the cursor, which is in the upper left-hand corner.

## Using the Cursor Control and Editing Keys

You use the cursor control and editing keys when you position the cursor or add or delete characters and lines on the screen. These are the basic keys you use to manipulate the cursor and the space on the screen to complement the more sophisticated functions you can perform with the numbered function keys.

### Moving the Cursor

To the right of your alphanumeric (main) keyboard, there are four arrow keys and the HOME key:




These four arrow keys are the cursor control keys. When you use the EDIT VS program, you press the cursor control keys as well as the RETURN key to move the cursor around the screen. When you move the cursor around an empty screen, you will notice:

- From the top to the bottom of the screen, there are 24 lines on which you can place the cursor. This means a screen holds up to 24 lines of text.
- From the left side to the right side of the screen, there are 80 columns in which you can place the cursor. This means a screen holds up to 80 characters per line of text.
- If you press the up (↑) cursor key when the cursor is on the top line of the screen, the cursor moves to the bottom line of the screen. If you press the down (↓) cursor key when the cursor is on the bottom line of the screen, the cursor moves to the top line of the screen.
- If you press the right (→) cursor key when the cursor is in column 80, the cursor moves to column 1 one line down. If you press the left (←) cursor key when the cursor is in column 1, the cursor moves to column 80 one line up.

- When you want to move the cursor multiple spaces, you can press any cursor key and hold it down.
- When you want to position the cursor in the upper-left-hand corner of the screen, you can press HOME.
- When you want to move the cursor from its current position to the column 1 one line down, you can press RETURN.

### Adding Text to a New File

Your new file named AESOP is empty. Your terminal screen is blank. To add text to AESOP, press HOME (if the cursor is not already in the upper-left-hand corner), and begin to type. Enter your text as though you were just writing on a piece of paper. If you make spelling mistakes, don't worry; see "Correcting Typing Errors," which follows.

If you are using the screen editor and ten minutes go by without your pressing one of the numbered function keys, your terminal beeps and EDIT VS reads the screen. During the time it takes VS to read the screen, the cursor disappears from the screen. When VS is through reading the screen, it returns the cursor to its position prior to the sound of the beep.

Any characters you type while EDIT VS is reading the screen are lost. So be patient. Wait for VS to read the screen, then resume typing when the cursor returns to the screen.

### Correcting Typing Errors

Once you have lines of text on your screen, you could also have spelling mistakes or spacing problems in your text. You can use the cursor control keys to locate the cursor at a misspelled word or line in your file. You then use the editing keys to add or delete one or more spaces or lines to correct any spelling or spacing errors.

The three editing keys, located near the cursor control keys at the right-hand side of the keyboard, are:

### Replacing One or More Characters

Use the cursor control keys to move the cursor to sit on the first (left-most) incorrect character. Notice when you use the cursor control keys that the cursor moves across the text on the screen without erasing it.

Simply type over the incorrect characters to replace them.

### Inserting or Deleting Characters

Use the cursor control keys to move the cursor to where you'd like to insert one or more characters. Each time you press CHAR INS, the text from the cursor to the end of the line moves to the right one space. Then type the characters you want to insert in the blank space you've created.

To delete one or more characters, position the cursor on the first (left-most) character you want to delete, then press CHAR DEL. Each time you press the key, the character under the cursor disappears and the text moves one space to the left.

In addition to inserting or deleting nonblank characters, you can use these editing keys to add or delete blank characters.

### Inserting or Deleting Lines

Use the cursor control keys to move the cursor to the line where you'd like to insert one or more lines. The line can be blank or filled with text. Press DEL LINE INS, and the text moves down the screen one line. Continue to press the key to continue adding blank lines.

You can also use the F5 (INSERT BLANK LINES) function key to add lines, but you should use the editing key to insert one or more lines, particularly at the head of a file. The F5 key is described later in this appendix.

To delete a line, position the cursor on the line you want to delete. If you want to delete more than one, place the cursor on the top line of those lines. As you hold down SHIFT press DEL LINE INS. The text or blank spaces on the line where the cursor is positioned disappears, and the text moves up one line. Press it again to continue deleting lines.

You can also use the SF5 (DELETE AND SAVE LINE) function key to delete one line at a time. This key is described later in this appendix.

## Exiting EDIT VS

When you are finished using the screen editor, press SF16 (press SHIFT and hold it down while you press F16) to exit from the EDIT VS program and return to the EDIT program. When you press SF16, your screen goes blank as you return to the line editor, and EDIT prompts you with the asterisk prompt in the upper-left corner of your screen. At this point, you can use the line editor to modify your EDIT file or you can exit from the EDIT program and return to the command interpreter. To return to the command interpreter, type:

```
*EXIT
```

## Entering EDIT VS to Edit an Existing File

Until you purge an EDIT file from the computer system, you can edit the file. To enter EDIT VS from the EDIT program to edit the file named AESOP, type:

```
1 EDIT AESOP
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.FICTION.AESOP
*XEQ VS
```

The first command, typed here at the command interpreter, starts the EDIT program on the file named AESOP. The second command, typed at the EDIT prompt, starts the EDIT VS program and by default puts the last line in the file at the top of the screen. (You can, however, start EDIT VS with any line at the top of the screen. See the description of the XEQ command later in this appendix.)

## Editing With the Numbered Function Keys

There are four types of editor functions triggered by the numbered function keys:

1. Paging Operations. The paging functions display a new page on the screen. You can move forward or backward in the EDIT file one screen page (24 consecutive lines), 4 screen pages, 8 screen pages, or 16 screen pages at a time. You can also move to the first and last screen page of the EDIT file. If the terminal bell beeps during a paging function, there is no more text. For example, if you press NEXT PAGE and hear a beep, there is no next page because you have reached the end of the file.
2. Scrolling Operations. The scrolling functions move you ahead or back in the EDIT file. You can scroll either one line or eight lines.
3. Block Operations. The block functions insert, delete, copy, or retrieve consecutive lines of text (a block). The block functions work on default blocks or on defined blocks.
   - ☐ A *default block* is the current line through the end of the screen.
   - ☐ A *defined block* is a block for which you indicate the first character and the last character. See the SF9 (DEFINE BLOCK) function key for instructions on defining a block.
4. Other Operations. The other functions let you perform a variety of editing functions such as breaking and joining lines, finding character strings, and manipulating columns.

When you press a function key:

- If the cursor is in column 80 of the first screen line, EDIT VS assumes that you have NOT changed the text on the screen since you pressed the previous function key. This means EDIT VS does not save the changes if you have changed the text on the screen and if the cursor is in column 80 of the first screen line.
- If the cursor is not in column 80 of the first screen line, EDIT VS compares each line on the screen with the EDIT file. EDIT VS flags the lines that differ and puts them into the EDIT file.

## EDIT VS Template

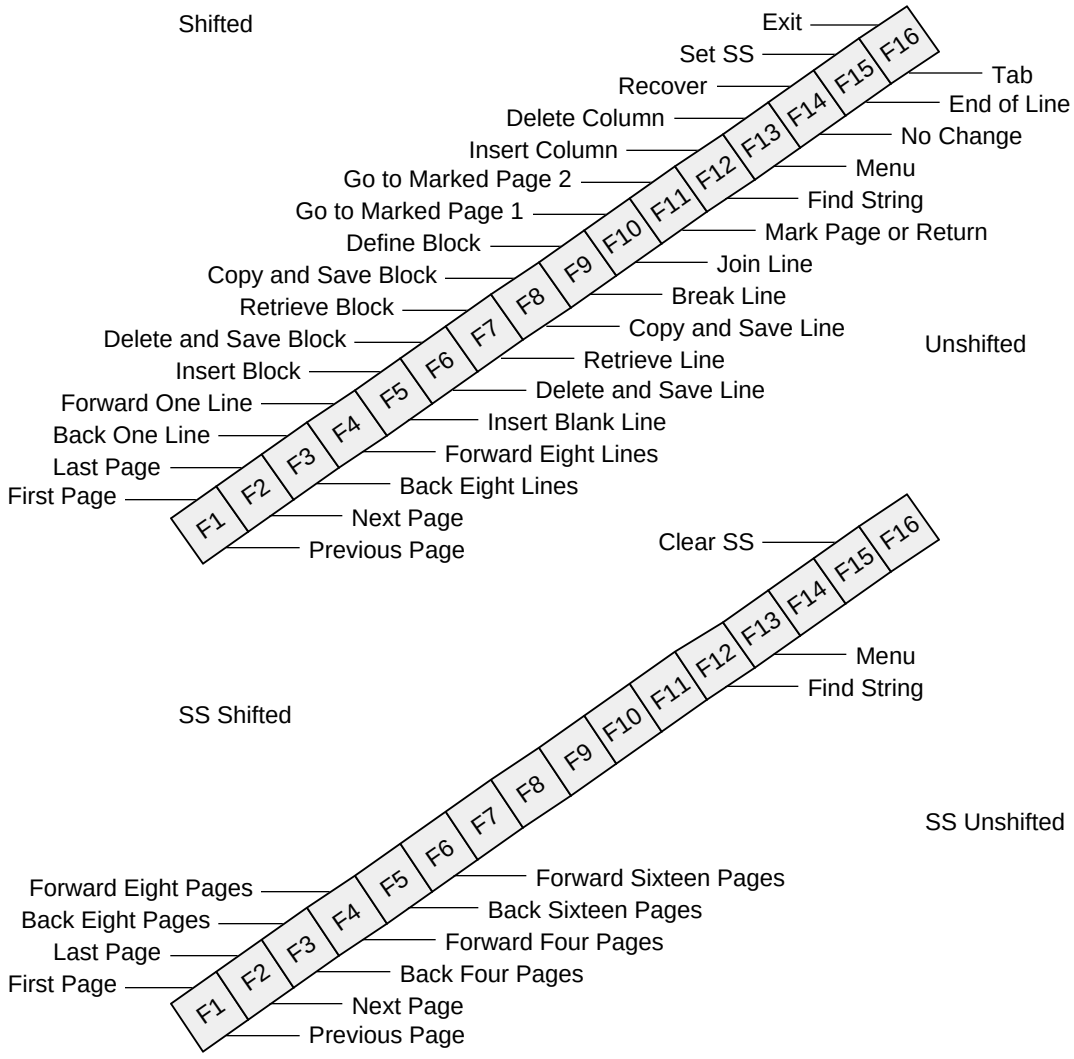
Tandem makes a function key template that fits your terminal and defines each numbered function key for the EDIT VS program. If you do not have a template for your terminal, see your manager and get one. The template makes it possible for you to use the screen editor without memorizing the edit function linked to each function key.

Figure 8 simulates the terminal template by displaying all of the numbered function keys with a brief description of each key. Take a look at Figure 8. Notice it has two rows of definitions because you can use each function key alone or with SHIFT. This allows each key to issue two basic commands.

- The top row defines the edit functions performed by the function keys when you press one as a shifted key. This means that you press SHIFT and hold it down while you press the function key once. In this manual, when you need to press a shifted function key, you'll see "press SF*number*" (for example, "press SF12"), and each key name is boxed.
- The bottom row defines the edit functions performed by the function keys when you press them once. In this manual, when you need to press a function key, you'll see "press F*number*" (for example, "press F8"), and each key name is boxed.

  Certain keys issue additional commands in the Super Shift mode. See the SF15 (SET SS) function key for details.

**Figure C-4. Operations of the EDIT VS Numbered Function Keys**



## EDIT VS Function Key Summary

The following summary, in numerical order by key, describes the function performed by each numbered function key.

**F1 (PREVIOUS PAGE)** The F1 function key moves the screen page backward in the file 24 lines.

**F2 (NEXT PAGE)** The F2 function key moves the screen page forward in the file 24 lines.

**F3 (BACK EIGHT LINES)** The F3 function key moves the screen page backward in the file eight lines.

**F4 (FORWARD EIGHT LINES)** The F4 function key moves the screen page forward in the file eight lines.

**F5 (INSERT BLANK LINE)** The F5 function key inserts a blank line before the line with the cursor, regardless of where the cursor is positioned in the line.

**Note** Do not use this function key to insert blank lines at the beginning of an EDIT file. If you want blank lines before the first line that contains text, add the blank lines using the EDIT program ADD command before entering EDIT VS or use the editing key DEL LINE INS (unshifted) once you are in the screen editor.

**F6 (DELETE AND SAVE LINE)** The F6 function key deletes the line with the cursor from the screen and saves it in the 24-line buffer stack. The deleted line remains in the stack until one of the following occurs:

- ☐ You press the F7 (RETRIEVE) function key to remove the line from the stack and insert it back into the text.
- ☐ The buffer stack contains more than 24 lines. EDIT VS deletes the oldest line from the stack.
- ☐ You exit the EDIT program.

**F7 (RETRIEVE LINE)** The F7 function key removes the last line from the 24-line buffer stack and places that line on the screen on the line before the current line. (EDIT VS puts the line with the cursor in the 24-line buffer stack when you press the F6 (DELETE LINE) and the F8 (COPY LINE) function keys. See the descriptions of the F6 and F8 function keys for details.)

**F8 (COPY AND SAVE LINE)** The F8 function key makes a copy of the line with the cursor and places the line in the 24-line buffer stack. The line remains in the stack until one of the following occurs:

- You press the F7 (RETRIEVE) function key to remove the line from the stack and insert it back into the text.
- The buffer stack contains more than 24 lines. EDIT VS deletes the oldest line from the stack.
- You exit the EDIT program.

The copied line remains unchanged on the screen.

**F9 (BREAK LINE)** The F9 function key breaks the current line into two lines. The position of the cursor determines where EDIT breaks the line. All characters preceding the cursor remain on the current line; all characters from the cursor to the end of the line move to a new line following the current line. The left margin of this new line lines up with the left margin of the current line.

**F10 (JOIN LINE)** The F10 function key moves words between the current line and the following line so that the current line is not longer than 70 characters. If the current line is less than 70 characters, words from the line following the current line can move up to the current line. If the current line is greater than 70 characters, words from the current line are moved to a new line following the current line. The left margin of this new line lines up with the left margin of the current line.

The default line width is 70 characters. If you want to join lines to a width greater or less than 70 characters, use the F13 (MENU) function key to change the line width to any length between a minimum of 10 characters and a maximum of 80 characters. (See the F13 function key for details.)

EDIT VS does not join lines with a backslash character ( ) in column 1.

**F11 (MARK PAGE OR RETURN)**

The F11 function key has two functions: It marks a page for the SF10 and SF11 (GO TO) function keys, or it returns you to a previously displayed page.

- ☐ To mark a page, press F11 (MARK PAGE) then immediately press SF10 (GO TO 1) or SF11 (GO TO 2). The pressing of these two keys marks the current screen page. Once you mark a page, you can go to that page at any time simply by pressing the SF10 (GO TO 1) or SF11 (GO TO 2) function key. When you exit EDIT VS, EDIT VS unmarks any marked pages.

  When you mark a page, EDIT VS writes the line number of the top line of the page on the GOTO-1 line or the GOTO-2 line of the VS Menu. You can display this menu (press F13 ) and see this line number. You can change this line number. See the F13 function key for details.

- ☐ To return to a previously displayed page, press F11 (MARK PAGE) twice. EDIT VS returns to the page that was displayed on the screen just before the most recent SF1 (FIRST PAGE), SF2 (LAST PAGE), SF10 (GO TO 1), SF11 (GO TO 2), or F13 (MENU) to set a line number. The return process only works after you give one of those five commands.

**F12 (FIND STRING)**

The F12 function key finds a string when used in conjunction with the F13 (MENU) function key. To use F12, follow these steps:

1. The text on your terminal screen is your current screen.
2. Press F13 (MENU) to display the menu screen. Then move the cursor to the menu line that reads:

   ```
   [W][B]"string"   <-
   ```

   Type in the string that you want to find, enclosing it in quotes and placing it to the right of the arrow. (See the F13 function key for details.)
3. After you enter the string into the menu, exit the menu by pressing ANY function key. EDIT VS returns you to your current screen.

4. Press F12 (FIND STRING). EDIT VS, starting at the current screen page, searches forward for the string and places the cursor on the first occurrence of the string that it finds.
5. The cursor moves to the next occurrence of the string each time you press F12
6. When EDIT VS reaches the end of your EDIT file, the terminal bell beeps. If you want EDIT VS to go to the beginning of your EDIT file and continue its search for the string, you must press SF1 (FIRST PAGE) to display the first screen page of your file.
7. Press F12 The cursor then moves to the next occurrence of the string.

The string that you enter into the menu remains there until you either change the string or exit EDIT VS.

**F13 (MENU)** The F13 function key displays the VS menu. This menu contains the current status of your EDIT file. The menu shows:

- ☐ The current line number (see LINENO [1], following)
- ☐ The string you want to locate (see [W] [B]"*string*", following)
- ☐ The current join width (see JOIN WIDTH [70], following)
- ☐ The line numbers of the first lines of the marked pages (see GOTO-1 and GOTO-2, following)
- ☐ The line number used by the return function (see RETN, following)

The VS menu looks like:

```
VIRTUAL SCREEN - T9601B30 - (08MAR87)
LINENO [1]         <-
[W][B]"string"     <-
JOIN WIDTH [70]    <-
COMMAND
(PRESS ANY FUNCTION KEY TO SIGNAL ENTRY)
GOTO-1 = UNDEFINED
GOTO-2 = UNDEFINED
RETN   = UNDEFINED
```

- LINENO [1] <-- displays the current line number. For example, if the cursor were on line number 141 of the EDIT file when you pressed the F13 (MENU) function key, this line would be:

```
LINENO [141]   <-
```

  When you exit this menu, EDIT VS returns you to the current line number.

  If you do not want to return to the current line number, type a new line number to the right of the arrow. Then, when you exit the menu, the line you specified is at the top of the screen, unless that line was on the screen before you entered the menu. If that line was on the screen, the screen remains unchanged.

- [W] [B]"*string*" <-- is the prompt for you to enter a string. If you enter a string, you can then search for the string in your EDIT file by exiting the menu and pressing the F12 (FIND STRING) function key. (See F12 for details.)

  You must enclose the string within a pair of quotes (”), single apostrophes (’), or forward slants (/). You can precede the string with a W, which is equivalent to the WORD keyword in the string-range parameter, or with a B, which is equivalent to the BOTH keyword in a string-range parameter. (Turn to the string-range tab for an explanation of these keywords.) For example, if you enter:

```
[W][B]"string"  <-  W "matter"
```

  EDIT searches the EDIT file for the word *matter*. If you enter the string:

```
[W][B]"string"  <-  B /oh dear/
```

EDIT searches the EDIT file for both uppercase and lowercase occurrences of the string *oh dear*. If you enter:

```
[W][B]"string"  <-  'Oh dear, what can the matter be'
```

EDIT searches for the string that exactly matches the string you typed.

If you enter a string, it remains in the menu until you return to the menu and change the string or until you exit the EDIT program.

- JOIN WIDTH [70] <-- displays the current join width used by the F10 (JOIN LINE) function. Because the default join width of the editor is 70, the value in brackets is displayed initially as 70.

  You can change the join width used by the F10 (JOIN LINE) function by typing a new join width here. The possible range of join widths is from 10 through 80. If you change the join width, the new width remains in effect until you return to the menu and change the width or until you exit the EDIT program.

- COMMAND is the prompt for a command string. If you want to enter a command, type a command string to the right of COMMAND and press any function key. Currently, no commands are valid in this field.

- GOTO-1 = UNDEFINED displays the line number of the top line of the marked page you set by pressing the F11 (MARK PAGE) function key and the SF10 (GO TO 1) function key. (See F11 and SF10 for details.) The line number is displayed in the field that initially displays UNDEFINED.

- GOTO-2 = UNDEFINED displays the line number of the top line of the marked page you set by pressing the F11 (MARK PAGE) and the SF11 (GO TO 2) function keys. (See F11 and SF11 for details.) The line number is displayed in the field that initially displays UNDEFINED.

- RETN = UNDEFINED displays the line number of the line to which EDIT returns you when you press F11 twice. (See F11 for details.) If you have not used the SF1 (FIRST PAGE), SF2 (LAST PAGE), SF10 (GO

TO 1), SF11 (GO TO 2), or F13 (MENU), no line number is displayed in the field that initially displays UNDEFINED.

- If you are using the VS menu and ten minutes go by without your entering text in any of the fields, EDIT VS returns you to the current screen of your EDIT file.

**F14 (NO CHANGE)** The F14 function key followed by a paging or scrolling function key (such as F1) tells EDIT VS to ignore any changes you made to the text on the screen. (You get the same result by placing the cursor in column 80 of the first screen line and pressing a paging or scrolling function key.)

**F15 (END OF LINE)** The F15 function key moves the cursor to the first blank position following the last nonblank at the end of the current line. If the line contains 80 characters, the cursor jumps to the end of the next line.

**F16 (TAB)**

**Note** Before you use the F16 function key, you must use the EDIT program SET command to set the horizontal tabs. See the SET command in Section 4 for details.

The F16 function key moves the cursor forward to the next horizontal tab column. If you press F16 and the cursor is beyond the last tab, the cursor moves to the beginning of the next line. If no tabs are set, press F16 to position the cursor in column 80.

**SF1 (FIRST PAGE)** The SF1 function key displays the first screen page of the file.

**SF2 (LAST PAGE)** The SF2 function key displays the last screen page in the file.

**SF3 (BACK ONE LINE)** The SF3 function key moves the screen page backward in the file one line.

**SF4 (FORWARD ONE LINE)** The SF4 function key moves the screen page forward in the file one line.

**SF5 (INSERT BLOCK)** The SF5 function key inserts the default block or a defined block of blank lines in front of the current line. (Default and defined blocks are defined earlier in the "Editing With the Numbered Function Keys" discussion and explained in the SF9 (DEFINE BLOCK) function key description.)

**SF6 (DELETE AND SAVE BLOCK)** The SF6 function key deletes the default block or a defined block from the screen and saves it in the 24-line buffer stack. (Default and defined blocks are defined earlier in the "Editing With the Numbered Function Keys" discussion and explained in the SF9 function key description.) The block remains in the stack until one of the following occurs:

- ☐ You press the F7 (RETRIEVE) function key to remove the line from the stack and insert it back into the text.
- ☐ The buffer stack contains more than 24 lines. EDIT VS deletes the oldest line from the stack.
- ☐ You exit the EDIT program.

To delete and save a block:

1. Move the cursor to the begining of the block you want to delete and press SF9 (DEFINE BLOCK).
2. Move the cursor to the end of the block you want to delete.
3. Press SF6 (DELETE AND SAVE BLOCK). The defined block disappears from the screen and is placed in the EDIT VS buffer stack.

**SF7 (RETRIEVE BLOCK)** The SF7 function key retrieves the entire block of text that is currently saved in the 24-line buffer stack and inserts it in front of the current line.

When you press the SF7 function key:

- ☐ EDIT VS breaks the current line at the cursor and inserts the block if the 24-line buffer stack contains a defined block and the current line is nonblank.
- ☐ EDIT VS inserts the block prior to the current line if the 24-line buffer stack contains a default block.

**SF8 (COPY AND SAVE BLOCK)**

The SF8 function key copies the default block or a defined block from the screen and saves it in the 24-line buffer stack. (Default and defined blocks are defined earlier in the "Editing With the Numbered Function Keys" discussion and explained in the SF9 function key description.) The copied block stays unchanged on the screen. The block remains in the stack until one of the following occurs:

- ☐ You press the F7 (RETRIEVE) function key to remove the line from the stack and insert it back into the text.
- ☐ The buffer stack contains more than 24 lines. EDIT VS deletes the oldest line from the stack.
- ☐ You exit the EDIT program.

You can only define a block within a screen; therefore, the most text you can copy and save at a time is the 24 lines of one screen.

To copy and save a block:

1. Move the cursor to the beginning of the block you want to copy and press SF9 (DEFINE BLOCK).
2. Move the cursor to the end of the block you want to copy, then press SF8 (COPY AND SAVE BLOCK).

**SF9 (DEFINE BLOCK)**

The SF9 function key defines the first character of a block of text.

To define a block, you must press two function keys:

- ☐ Position the cursor at the first character of the block and press the SF9 (DEFINE BLOCK) function key.
- ☐ Move the cursor to the last character of the block and press the SF6 (DELETE BLOCK), SF8 (COPY BLOCK), or SF5 (INSERT BLOCK) function key. If you do not press one of these three block function keys immediately after pressing SF9 , EDIT VS cancels the define block function.

See the SF6 and SF8 function key descriptions for more details on deleting or copying a block.

**SF10 (GOTO MARKED PAGE 1)** The SF10 function key displays the marked page assigned to this key (see the F11 function key).

**SF11 (GOTO MARKED PAGE 2)** The SF11 function key displays the marked page assigned to this key (see the F11 function key).

**SF12 (INSERT COLUMN)** The SF12 function key inserts a blank character into each line from the current line to the bottom of the screen. The characters to the right of and including the cursor move one space to the right each time you press SF12. You can use this function key to justify tabular text.

**SF13 (COLUMN DELETE)** The SF13 function key deletes a character from each line from the current line to the last line on the screen. The column deleted is indicated by the current position of the cursor. When the column of characters is deleted, the rest of the line moves into the position created by the deletion of the character. This command is useful for justifying tabular material.

**SF14 (RECOVER)** The SF14 function key recovers the current page if the page is cleared from the screen. (Accidentally pressing the CLEAR SPACES key on a 6511 terminal clears the screen page.) When the page returns to the screen, it will NOT contain changes you made to it prior to clearing the screen.

**SF15 (SET SUPER SHIFT)** When you press the SF15 function key, you set the Super Shift (SS) capability of VS. A blinking indicator appears on the screen. Some of the function keys, both shifted and unshifted, are given new functions in Super Shift. However, you press the shifted and unshifted function keys just as you would in non-Super Shift mode. The SS function keys are:

- SS F1 (PREVIOUS PAGE). The SS F1 function key displays the previous screen page.
- SS F2 (NEXT PAGE). The SS F2 function key displays the next screen page in the file.
- SS F3 (BACK FOUR PAGES). The SS F3 function key moves back four screen pages from the current page and displays that page.

- ☐ SS F4 (FORWARD FOUR PAGES). The SS F4 function key moves forward four pages from the current page and displays that page.
- ☐ SS F5 (BACK SIXTEEN PAGES). The SS F5 function key moves back sixteen pages from the current page and displays that page.
- ☐ SS F6 (FORWARD SIXTEEN PAGES). The SS F6 function key moves forward sixteen pages from the current page and displays that page.
- ☐ SS F7 through SS F11 are not used.
- ☐ SS F12 (FIND STRING). The SS F12 function key performs the same function as the F12 function key.
- ☐ SS F13 (MENU). The SS F13 function key performs the same function as the F13 function key.
- ☐ SS F14 through SS F16 are not used.
- ☐ SS SF1 (FIRST PAGE). The SS SF1 function key displays the first screen page of the file.
- ☐ SS SF2 (LAST PAGE). The SS SF2 function key displays the last screen page in the file.
- ☐ SS SF3 (BACK EIGHT PAGES). The SS SF3 function key moves back eight pages from the current page and displays that page.
- ☐ SS SF4 (FORWARD EIGHT PAGES). The SS SF4 function key moves forward eight pages from the current page and displays that page.
- ☐ SS SF5 through SS SF14 are not used.
- ☐ SS SF15 (CLEARS SS). The SS SF15 function key clears the SS mode and returns you to regular VS mode.
- ☐ SS SF16 is not used in SS mode.

**SF16 (EXIT)** The SF16 function key terminates the EDIT VS program and returns you to the EDIT program.

## EDIT Commands Requiring Full-Screen Terminals

The three EDIT commands described here—ADD BLOCK, REPLACE BLOCK, and XEQ—are discussed with the EDIT VS editing functions because, like EDIT VS, these block mode EDIT commands require a terminal that has a full-screen capability (such as a Tandem 653X or emulator). They do not work with terminals that display one line at a time and can only run EDIT, a program primarily designed for line editing.

Two commands you can type at the EDIT prompt allow you to create and modify text as if you were using a screen editor, then to enter the text into an EDIT file one full screen (24 lines) at a time. These two commands are:

- ADD BLOCK. This command adds up to one full screen of text to an EDIT file.
- REPLACE BLOCK. This command displays a block of text from the current EDIT file, lets you modify the text, then inserts the text back into the EDIT file.

Users will find that these commands are precursors to commands now present in EDIT VS. You might occasionally find a use for ADD BLOCK; chances are you might never use REPLACE BLOCK.

The third command, XEQ (EXECUTE), is the EDIT command you type to invoke the EDIT VS program.

## ADD BLOCK Command

The ADD BLOCK command captures the text on the terminal screen and adds it to an EDIT file, allowing you to mimic page mode editing at your terminal.

### What to Enter

```
ADD BLOCK [ line-range [ BY incr ] ]
```

`line-range`

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

If you omit a range parameter, EDIT begins adding text where the last ADD BLOCK command terminated. If there was no previous ADD BLOCK command, EDIT begins adding text at the end of the EDIT file.

`BY`

specifies the numbering increment of the line numbers assigned to the text that you add to the EDIT file. If you omit BY, EDIT chooses the increment (either 1, .1, .01, or .001).

`incr`

is a number from .001 through 10.

## How to Use ADD BLOCK

You use the ADD BLOCK command when you have text on your terminal that you want to add to an EDIT file. For example, you might want to incorporate the results of a command interpreter command into an EDIT file or you might want to keep 20 lines of an editing session.

The following example demonstrates how you can use ADD BLOCK with a command interpreter command.

### Example

1. You type a command interpreter command. For example, you type:

```
13 USERS *
```

The command prints information on the screen. Your screen looks like this:

```
14 USERS *

GROUP . USER       I.D. #   SECURITY   DEFAULT VOLUMEID

CLASS.JONES        009,001     NUNU     $GUEST.JONES
CLASS.SMITH        009,002     NUNU     $GUEST.SMITH
CLASS.LINCOLN      009,003     NUNU     $GUEST.LINCOLN
CLASS.SHAW         009,004     NUNU     $GUEST.SHAW
CLASS.WILDE        009,005     NUNU     $GUEST.WILDE
CLASS.TOMLIN       009,006     NUNU     $GUEST.TOMLIN
CLASS.FRANKS       009,007     NUNU     $GUEST.FRANKS
CLASS.LEWIS        009,008     NUNU     $GUEST.LEWIS
CLASS.RUSSELL      009,009     NUNU     $GUEST.RUSSELL
15
```

2. You name the EDIT file into which you want to add the text on your screen. For example, you type:

```
15 EDIT ZUSERS !
```

and EDIT creates an EDIT file named ZUSERS. Your screen now looks like this:

```
14 USERS *

GROUP . USER       I.D. #   SECURITY   DEFAULT VOLUMEID

CLASS.JONES        009,001     NUNU     $GUEST.JONES
CLASS.SMITH        009,002     NUNU     $GUEST.SMITH
CLASS.LINCOLN      009,003     NUNU     $GUEST.LINCOLN
CLASS.SHAW         009,004     NUNU     $GUEST.SHAW
CLASS.WILDE        009,005     NUNU     $GUEST.WILDE
CLASS.TOMLIN       009,006     NUNU     $GUEST.TOMLIN
CLASS.FRANKS       009,007     NUNU     $GUEST.FRANKS
CLASS.LEWIS        009,008     NUNU     $GUEST.LEWIS
CLASS.RUSSELL      009,009     NUNU     $GUEST.RUSSELL
15EDIT ZUSERS !
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $GUEST.TALCLASS.ZUSERS
*
```

3. You type:

```
*ADD BLOCK 25 BY 2
```

   to put the terminal in page mode and tell EDIT when it adds the text on the screen to the EDIT file to start at line number 25 and number each line by an increment of 2. Your screen now looks like this:

```
14 USERS *

GROUP . USER       I.D. #   SECURITY   DEFAULT VOLUMEID

CLASS.JONES        009,001     NUNU     $GUEST.JONES
CLASS.SMITH        009,002     NUNU     $GUEST.SMITH
CLASS.LINCOLN      009,003     NUNU     $GUEST.LINCOLN
CLASS.SHAW         009,004     NUNU     $GUEST.SHAW
CLASS.WILDE        009,005     NUNU     $GUEST.WILDE
CLASS.TOMLIN       009,006     NUNU     $GUEST.TOMLIN
CLASS.FRANKS       009,007     NUNU     $GUEST.FRANKS
CLASS.LEWIS        009,008     NUNU     $GUEST.LEWIS
CLASS.RUSSELL      009,009     NUNU     $GUEST.RUSSELL
15EDIT ZUSERS !
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $GUEST.TALCLASS.ZUSERS
*ADD BLOCK 25 BY 2
```

4. You use the cursor keys to move the cursor. You use these three editing keys:

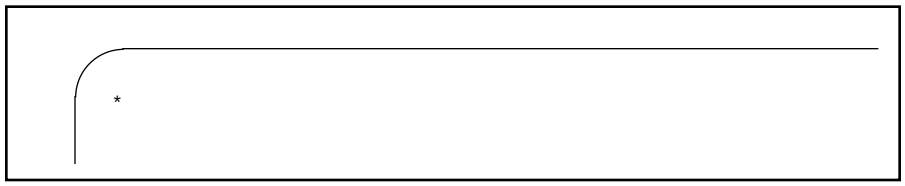

   to insert and delete lines and characters. You use the letters, numbers, and punctuation keys to type over the text on the screen. You modify the text on the screen so that the screen now looks like this:

```
CLASS.JONES        009,001     NUNU     $GUEST.JONES
CLASS.SMITH        009,002     NUNU     $GUEST.SMITH
CLASS.LINCOLN      009,003     NUNU     $GUEST.LINCOLN
CLASS.SHAW         009,004     NUNU     $GUEST.SHAW
CLASS.WILDE        009,005     NUNU     $GUEST.WILDE
CLASS.TOMLIN       009,006     NUNU     $GUEST.TOMLIN
CLASS.FRANKS       009,007     NUNU     $GUEST.FRANKS
CLASS.LEWIS        009,008     NUNU     $GUEST.LEWIS
CLASS.RUSSELL      009,009     NUNU     $GUEST.RUSSELL
```

5. You move the cursor to the end of the last line on the screen that you want to add to ZUSERS.

```
CLASS.JONES        009,001     NUNU     $GUEST.JONES
CLASS.SMITH        009,002     NUNU     $GUEST.SMITH
CLASS.LINCOLN      009,003     NUNU     $GUEST.LINCOLN
CLASS.SHAW         009,004     NUNU     $GUEST.SHAW
CLASS.WILDE        009,005     NUNU     $GUEST.WILDE
CLASS.TOMLIN       009,006     NUNU     $GUEST.TOMLIN
CLASS.FRANKS       009,007     NUNU     $GUEST.FRANKS
CLASS.LEWIS        009,008     NUNU     $GUEST.LEWIS
CLASS.RUSSELL      009,009     NUNU     $GUEST.RUSSELL
```

You press any one of the numbered function keys (F1 through F16). When you press the function key, the text between the top left-hand corner of the screen and the cursor is added to ZUSERS. Your screen now looks like this:



6. To prove to yourself that the result of the USERS command is really in the ZUSERS EDIT file, type:

```
*LIST 25/LAST
   25     CLASS.JONES        009,001     NUNU    $GUEST.JONES
*  27     CLASS.SMITH        009,002     NUNU    $GUEST.SMITH
   29     CLASS.LINCOLN      009,003     NUNU    $GUEST.LINCOLN
   31     CLASS.SHAW         009,004     NUNU    $GUEST.SHAW
   33     CLASS.WILDE        009,005     NUNU    $GUEST.WILDE
   35     CLASS.TOMLIN       009,006     NUNU    $GUEST.TOMLIN
   37     CLASS.FRANKS       009,007     NUNU    $GUEST.FRANKS
   39     CLASS.LEWIS        009,008     NUNU    $GUEST.LEWIS
   41     CLASS.RUSSELL      009,009     NUNU    $GUEST.RUSSELL
*
```

**Tips**

- If you press BREAK while the terminal is in page mode, EDIT stops the ADD BLOCK command and returns the terminal to conversational mode; no text is entered.
- ADD BLOCK captures all the characters on the screen; for example, ADD BLOCK can capture the line numbering, error messages displayed on the terminal screen, the interactive process of using the FIX command, and so on. This differs from the PUT command, for example, which moves just the text lines of your file to another specified file.
- If, instead, you use the ADD BLOCK command when you're in an EDIT file (unlike when you're at the command interpreter, as in the preceding example), you would just follow steps 3 through 6 of the ADD BLOCK example. You do not need to specify the file name in which to place the text. EDIT simply adds the captured text to the end of your current file.

**Note** Users with terminals on an AM6520 line should use other EDIT commands, such as PUT, instead of ADD BLOCK when wanting to capture text. The ADD BLOCK command, used on these terminals, causes the command to clear the terminal screen first before placing the text in a specified EDIT file and placing the cursor at the top left corner of the screen. This anomaly can cause a loss of the original text.

## REPLACE BLOCK Command

The REPLACE BLOCK command lets you edit text a block at a time.

### What to Enter

```
REPLACE BLOCK  { line-range  }
```

`line-range`

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

### How to Use REPLACE BLOCK

Study the following pages to see an example of using the REPLACE BLOCK command.

1. You're editing a file that contains the following lines:

```
 1     I am the very model
 2      of a modern Major-General,
 3     I've information vegetable,
 4      animal, and mineral,
 5     I know the kings of England,
 6      and I quote the fights historical,
 7     From Marathon to Waterloo,
 8      in order categorical;
 9     I'm very well acquainted too
10      with matter mathematical,
11     I understand equations,
12      both the simple and quadratical,
13     About binomial theorem
14      I'm teeming with a lot of news--
15     With many cheerful facts
16      about the square of the hypotenuse.
17     I'm very good at integral
18      and differential calculus,
19     I know the scientific names
20      of beings animalculous;
              .
              .
              .
65     For my military knowlege,
66      though I'm plucky and adventury,
67     Has only been brought down
68      to the beginning of the century;
69     But still in matters vegetable,
70      animal, and mineral,
71     I am the very model
72      of a modern Major-General.
```

You type the command:

```
*REPLACE BLOCK ALL
```

to put the terminal in page mode and print every line in the file, one block at a time, on the screen. The default number of lines in a block is 16. You can change the block size to a number between 1 and 20 by using the BLOCK option of the SET command. (See the SET command,

following, for details.) Because you haven't set the block size, EDIT prints the first 16 lines in the file on the screen. Your screen then looks like this:

```
I am the very model
 of a modern Major-General,
I've information vegetable,
 animal, and mineral,
I know the kings of England,
 and I quote the fights historical,
From Marathon to Waterloo,
 in order categorical;
I'm very well acquainted too
 with matter mathematical,
I understand equations,
 both the simple and quadratical,
About binomial theorem
 I'm teeming with a lot of news--
With many cheerful facts
 about the square of the hypotenuse.
```

2. You use the cursor keys to move the cursor. You use these editing keys:


to insert and delete lines and characters. You use the letters, numbers, and punctuation keys to type over the text on the screen. You modify the text on the screen so that the screen now looks like this:

```
I am the very model of a modern Major-General,
I've information vegetable, animal, and mineral,
I know the kings of England, and I quote the fights
 historical,
From Marathon to Waterloo, in order categorical;
I'm very well acquainted too with matter mathematical,
I understand equations, both the simple and quadratical,
About binomial theorem I'm teeming with a lot of news--
With many cheerful facts about the square of the
 hypotenuse.
```

3. You move the cursor to the end of the last line on the screen.

```
I am the very model of a modern Major-General,
I've information vegetable, animal, and mineral,
I know the kings of England, and I quote the fights
 historical,
From Marathon to Waterloo, in order categorical;
I'm very well acquainted too with matter mathematical,
I understand equations, both the simple and quadratical,
About binomial theorem I'm teeming with a lot of news--
With many cheerful facts about the square of the
 hypotenuse.
```

You press any one of the numbered function keys (F1 through F16). When you press the function key, the text between the top left-hand corner of the screen and the cursor is inserted back into the EDIT file. At this point, the old text in the EDIT file that was displayed in the block is replaced with the edited screen image.

Notice that the number of lines on the screen after editing need not match the number displayed by EDIT. If fewer lines are present after editing, lines in the EDIT file represented by the original screen image are deleted accordingly (and there is a gap in the line numbers); if more lines are present after editing, line numbers are adjusted (if possible) so that the additional lines fit into the EDIT file.

Your screen now looks like this:

```
I'm very good at integral
 and differential calculus,
I know the scientific names
 of beings animalculous;
In short, in matters vegetable,
 animal, and mineral,
I am the very model
 of a modern Major-General.
I know our mythic history,
 King Arthur's and Sir Caradoc's,
I answer hard acrostics,
 I've a pretty taste for paradox,
I quote in elegiacs
 all the crimes of Heliogabalus
In conics I can floor
 peculiarities parabolous.
```

4. EDIT continues to display 16-line blocks of text from the EDIT file on your screen until it has displayed the entire file or until you type the characters / / in columns 1 and 2 of a line. If you type / / in columns 1 and 2 and then press any function key, EDIT asks the following question:

```
SHALL I DELETE THE REMAINING LINES?
```

   If you respond with “Y”, “y”, “YES”, or “yes”, EDIT deletes the line containing the / / sequence and any remaining lines in the range. Any other response leaves the line with the / / sequence and any remaining lines in the EDIT file unchanged. No matter what your response, the REPLACE BLOCK command terminates.

5. If you press \X(\S\UP0\DI-1(BREAK) while EDIT is presenting a block or while the terminal is in page mode, the REPLACE BLOCK command stops, the terminal is brought back to conversational mode (EDIT), and EDIT prompts for a command; the text in the text file represented by the current block is left unchanged.

## XEQ Command

The XEQ (EXECUTE) command invokes another program from within the EDIT program. Currently, the only program that can be invoked with the XEQ command is EDIT VS.

### What to Enter

```
XEQ VS [ line-range ]
```

VS

is the partial file name of the screen editor program. See "How to Use XEQ."

line-range

references one or more contiguous lines in an EDIT file. Turn to "Line-Range Parameter" in Section 5 for a full explanation of this range.

### How to Use XEQ

The full file name of the EDIT VS program is $SYSTEM.SYSTEM.VS. When you give the command to start VS, EDIT searches your current *volume.subvolume* for a file named VS. If EDIT finds a file named VS in your current *volume.subvolume*, it assumes that file is the EDIT VS program and attempts to run it. Therefore:

```
NEVER name a file VS.
```

When EDIT does not find a file named VS in your current *volume.subvolume*, it searches $SYSTEM.SYSTEM, finds a file named VS, and runs the EDIT VS program.

### Examples

For the following examples, the EDIT file named SHAW contains the lines:

```
1     People are always blaming their circumstances for
2     what they are.  I don't believe in circumstances.
3     The people who get on in this world are the people
4     who get up and look for the circumstances they
5     want, and, if they can't find them, make them.
```

1. The command:

```
1> EDIT SHAW
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.NONFICT.SHAW
*XEQ VS FIRST
```

```
People are always blaming their circumstances for
what they are.  I don't believe in circumstances.
The people who get on in this world are the people
who get up and look for the circumstances they
want, and, if they can't find them, make them.
```

starts the EDIT program at the command interpreter prompt, then invokes the EDIT VS program and displays the first line in the file on the top of the screen. If you don’t specify a line-range, EDIT VS displays the last line of your file at the top of the screen.

2. VS can also be entered as part of the EDIT command:

```
1> EDIT SHAW; XEQ VS
TEXT EDITOR - T9601B30 - (08MAR87)
CURRENT FILE IS $WORK.NONFICT.SHAW
```

```
want, and, if they can't find them, make them.
```

# Appendix D EDIT VS Error Messages

The EDIT VS errors listed here require you to follow specific recovery procedures in order to save your editing. The EDIT VS recovery procedures are described in Appendix E.

```
I MALFUNCTIONED. PLEASE REPORT TO TANDEM COMPUTERS:
```

An internal consistency check within EDIT failed. See Procedure F in Appendix E for instructions.

```
LINES NOT ADDED:
```

After an ADD BLOCK operation, EDIT prints the line numbers of any lines that it could not add to the file because an increment in line numbers of less than .001 would be required.

```
PROCESSOR FAILURE:xx
2>
```

See Procedure C in Appendix E for information on how to recover from this error.

```
RECOVERY OPERATION ABORTED
```

EDIT is unable to complete the recovery operation. Examine your EDIT file. If it is incorrect, you will have to fix it manually.

```
RENUMBERING ERROR
```

See Procedure A in Appendix E for information on how to recover from this error.

```
STACK DUMP ...
```

See Procedure F in Appendix E for information on how to recover from this error.

```
TRAP ...
```

See Procedure A in Appendix E for information on how to recover from this error.

```
The terminal disconnects from phone
```

See Procedure A in Appendix E for information on how to recover from this error.

```
The text on your screen disappears and
all you see is a command interpreter.
```

See Procedure B in Appendix E for information on how to recover from this error.

```
The text on your screen disappears and the
herringbone pattern (<<<<<<<<<) appears.
```

See Procedure D in Appendix E for information on how to recover from this error.

```
The text on your screen disappears and
the diagnostic test pattern flashes on
the screen.
```

See Procedure E in Appendix E for information on how to recover from this error.

```
VS ABORTED BECAUSE OF I/O ERROR - $devname

** LINES 149.11 TO 210 FOR EDIT-FILE $volname.subvolname.filename
   SAVED IN VS RECOVERY FILE $volname.subvolname.filename.
   DO "X VS : $volname.subvolname.filename:" TO RECOVER DATA
```

See Procedure A in Appendix E for information on how to recover from this error.

```
VS ABORTED BECAUSE OF I/O ERROR - $devname (191)
PLEASE EXAMINE FILE BEFORE DOING RECOVERY.
RECOVERY MAY NOT BE NECESSARY.

** LINES 1 TO 24 FOR EDIT-FILE $volname.subvolname.filename
   SAVED IN VS RECOVERY FILE $volname.subvolname.filename.
   DO "X VS : $volname.subvolname.filename :" TO RECOVER DATA
```

See Procedure A in Appendix E for information on how to recover from this error.

```
You press a function key and nothing happens.
```

See Procedure E in Appendix E for information on how to recover from this error.

# Appendix E EDIT VS Error Recovery Procedures

EDIT is programmed to keep your EDIT file correct and to remember all your editing, whenever possible. Annoyances that you will encounter when editing with the screen editor include:

- ☐ Pressing an invalid numbered function key while in EDIT VS. When you press an invalid function key, the terminal beeps and EDIT does not perform the operation triggered by that key.
- ☐ Mistyping text while in EDIT VS. You can cancel any modifications you made to text since you pressed the most recent function key by pressing the SF14 (RECOVER) function key. (See the description of SF14 in Appendix C for details.)
- ☐ Receiving communication errors that garble the text on the screen. You recover from most communication errors by pressing the SF14 (RECOVER) function key. If this action does not restore the text that was on the screen prior to the communication error, press BREAK and go to Procedure C. Occasionally, you might need to deal with a problem that requires you to follow a specific recovery procedure in order to save your editing. The remainder of this appendix describes the various procedures you can use to recover when editing in EDIT VS.

## EDIT VS Recovery File

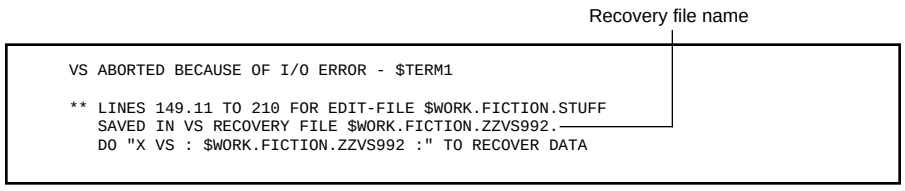
When EDIT VS is running, it keeps a special file named:

```
$volname.subvolname.ZZVSnnn
```

in your current subvolume. This recovery file supports the SF14 (RECOVER) function key and is also used for some of the recovery procedures outlined in the following paragraphs. When EDIT exits normally, it deletes this recovery file. So, you only see this recovery file in your subvolume when:

- ☐ EDIT VS terminates abnormally
- ☐ You do a ?FILES command from EDIT after you use EDIT VS

Some EDIT VS error messages (listed in Appendix D) tell you which recovery file to use in the recovery procedure. For example, the following error message tells you to use the recovery file named ZZVS992 in the recovery procedure:




Some EDIT VS error messages, such as those associated with processor failures, do not tell you which recovery file to use in the recovery procedure. When the error message does not name the recovery file, use one of the following commands to list the files in your current subvolume:

- ☐ The FILES command, typed at the command interpreter. For example:

```
3 FILES
```

- ☐ The FUP INFO * command, typed at the command interpreter. For example:

```
4 FUP INFO *
```

- ☐ The ?FILES command. For example:

```
*?FILES
```

If you find only one file named ZZVSnnn in your subvolume, use that file in the recovery procedure. If you find several files named ZZVSnnn, use the FUP INFO * command to check the dates on the files. Then use the newest file in the recovery procedure.

**Note** Make a habit of purging any ZZVSnnn files that you don't need. This not only saves disk space but also prevents confusion when you need to recover.

## Recovery Procedures

EDIT VS has seven different recovery procedures. When you receive an error message, look up the message in Appendix D, "EDIT VS Error Messages." Then, if indicated, see the appropriate error recovery procedure described in the following paragraphs.

### Recovery Procedure A

1. If the error message includes the recovery file name, write this file name on a piece of paper. If the error message does not include the recovery file name, use the ?FILES command (explained earlier in this appendix under "EDIT VS Recovery File") to find the name of the recovery file. Then write this file name on a piece of paper.
2. At the asterisk prompt, type:

```
*XVS
```

   This command returns you to EDIT VS.

3. Use the numbered function keys to page through the EDIT file. Check to see if the file is correct. If the file is not correct, go to Step 4. If the file is correct, continue editing—you do not need to recover.
4. Press SF16 (EXIT) to return to EDIT.
5. Before you recover into the current EDIT file, you must check the recovery file and make sure it is correct. Type:

```
*GET filename
```

   where *filename* is the name of a new EDIT file. For example:

```
*GET SAVE
$WORK.FICTION.SAVE DOES NOT EXIST. SHALL I CREATE IT? yes
CURRENT FILE IS $WORK.FICTION.SAVE
*
```

6. Now type:

```
*XVS : ZZVSnnn :
```

   where ZZVSnnn is the name of the recovery file.

7. The following message appears on the screen. Reply yes.

```
DATA IN RECOVERY FILE IS FOR LINES nn TO nn
OF EDIT FILE $WORK.FICTION.MYFILE
--WARNING--CURRENT FILE IS $WORK.FICTION.SAVE
SHALL I CONTINUE (Y?N)? yes
```

   Check this EDIT file.

☐ If this EDIT file contains the 24 lines of text that were on the screen prior to the error, then the recovery file is correct. Press SF16 (EXIT). Then go to Step 8.

☐ If this EDIT file does not contain the correct text (for example, is nothing but blank lines), you cannot use it to recover. Go to Step 13, follow those instructions, then manually correct your EDIT file.

8. If the original error message was a numbering error or if you have renumbered the EDIT file that was current when the error occurred, go to Step 9. Otherwise, go to Step 13.

9. Press HOME to place the cursor in the upper left-hand corner of the screen. Then press SF6 (DELETE AND SAVE BLOCK) to delete and save this screen of text.

10. Press SF7 (RETRIEVE BLOCK) to retrieve the deleted block of text.

11. Press SF16 (EXIT) to exit EDIT VS.

12. At the asterisk prompt, renumber all the lines in the file by typing:

```
*NUMBER ALL
*
```

13. At the asterisk prompt, type:

```
*GET filename
```

    where *filename* is the EDIT file that you were editing when the error occurred. For example:

```
*GET MYFILE
CURRENT FILE IS $WORK.FICTION.MYFILE
*
```

    If you moved to this step from Step 7, stop here. Do not perform the remainder of the steps in this recovery procedure. Instead, look through the EDIT file and fix things you find wrong. Then continue editing.

14. If the original error message was a numbering error or if you have renumbered the EDIT file that was current when the error occurred, go to Step 16. Otherwise, go to Step 15.

15. At the asterisk prompt, type:

```
*XVS : ZZVSnnn :
```

    where ZZVSnnn is the name of the recovery file. The following message appears on the screen. Reply yes.

```
DATA IN RECOVERY FILE IS FOR LINES nn TO nn
OF EDIT FILE $WORK.FICTION.MYFILE
SHALL I CONTINUE (Y?N)? yes
```

Recovery is complete—you can continue editing.

16. Renumber all the lines in the file by typing:

```
*NUMBER ALL
*
```

17. Search the current EDIT file to determine where the recovered text belongs. Then, merge the text from the EDIT file that contains the recovery text into the current file by typing:

```
*GET filename TO lnum
```

where *filename* is the name of the EDIT file into which you saved the text from the recovery file and *lnum* is an available EDIT file line number. For example:

```
*GET SAVE TO 124.001
LAST NEW LINE IS 124.024 <- 436
CURRENT FILE IS $WORK.FICTION.MYFILE
*
```

18. Edit the current EDIT file to remove any duplicate lines.

### Recovery Procedure B

If you are using a Tandem 6511 or 6512 terminal and if the command interpreter prompts you just after you press a function key, then follow Steps 1 through 5. If you are using a Tandem 6520 or 6530 terminal, follow Steps 3 and 4.

1. Press RESET once to unlock the keyboard.
2. Press MODE to put the terminal into conversational mode.
3. Type:

```
1 PAUSE
```

   and EDIT VS should perform the operation for the key that you pressed prior to the prompt. (After you type PAUSE, if the terminal displays:

```
FILE SYSTEM ERROR 11
```

   or some other error message, you might have mistyped PAUSE. Try again.)

4. When you return to EDIT VS, locate and remove the command interpreter prompt character from your EDIT file.
5. If the keyboard is locked after you return to EDIT VS, press CONV MODE to take the terminal out of conversational mode.

   If this procedure does not work, the processor in which your EDIT VS process is running might have failed. In this case, go to Procedure C, below.

**Recovery Procedure C**

1. If the processor in which your EDIT VS process is running fails, usually the EDIT process is also gone. You can check this by typing the STATUS *, TERM command at the command interpreter. For example:

```
1> STATUS *, TERM
Process          Pri PFR %WT  Userid  Program file
             Hometerm
           7,100 148     005    8,13
$system.system.edit       $lilli
           7,102 148     004    8,13
$system.system.vs         $lilli
$e252   b 10,88  150     001    8,13
$system.sys03.tacl        $lilli
$e252     11,68  150     001    8,13
$system.sys03.tacl        $lilli
```

   If no EDIT or EDIT VS processes are running, go to Step 4; otherwise, go to Step 2.

2. To see if you can reenter EDIT, type the PAUSE command at the command interpreter. For example:

```
1> PAUSE
```

   If PAUSE doesn't work, press BREAK, then go to Step 3. If PAUSE does allow you to reenter EDIT, then continue editing.

3. To stop any EDIT and EDIT VS processes, use the STOP command at the command interpreter. For example:

```
3> STOP  7,102
4>
```

4. Because the error message did not name the recovery file, you must type the FILES or FUP INFO * command at the command interpreter to find the name of the recovery file. For example:

```
4 FILES
AESOP     PEEP      SHAKE     ZZVS972
5
```

5. Follow Steps 5, 6, 7, 8, 13, 14, and 15 of Procedure A.

**Recovery Procedure D**

If the herringbone pattern appears on your screen, press RESET three times to put the terminal into self-test mode. Then go to Procedure E.

**Recovery Procedure E**

Use this procedure if:

- ☐ You press a numbered function key and nothing happens.
- ☐ You press RESET three times and put the 6520 or 6530 into self-test mode.

Recovering from these problems is not difficult. However, if you are not careful, you will end up with 24 blank lines in your file instead of the 24 lines that were on your screen when the error occurred.

1. Press RESET twice to unlock the keyboard.
2. If you receive an I/O error message, such as this:

```
VS ABORTED BECAUSE OF I/O ERROR - $TERM1
```

go to Procedure A; otherwise, go to Step 3.

3. Press BREAK . If you receive a prompt from your command interpreter, go to Procedure C. If you do not receive a prompt, both processors controlling your terminal might be down or locked up. Try one or both of the following:

   - ☐ Wait until you get a prompt (: or 1) and then go to Procedure C.
   - ☐ Go to another terminal and type the STATUS command (STATUS *, TERM *termname*) at the command interpreter to list the processes running on your terminal. For example:


If you find editor processes running on your terminal, go to Step 3 of Procedure C; otherwise, wait until you get a prompt, then go to Procedure C.

## Recovery Procedure F

When you use EDIT VS, it makes internal consistency checks. If one of these checks fails or if a trap having a trap number in the range of {0:3} occurs, VS prints the message:

```
** PLEASE REPORT THIS FAILURE TO TANDEM COMPUTERS INC.
   (STACK DUMP PLACED IN FILE $dflt vol.dflt subvol.AAAAnnn)
```

The file named in the error message contains an image of the data stack at the time of the failure.

1. Copy the entire error message onto a sheet of paper.
2. Use the FUP INFO * command at the command interpreter to list the current subvolume. For example:

```
5 FUP INFO *
               CODE        EOF     LAST
 MODIF   OWNER  RWEP  ...
$BOOKS1.M079B00
AAAA831         111       4273  12NOV86  11:38
7,21  CUCU
AESOP           101       3608  29OCT86   8:55
   7,21  CUCU
MYFILE          101       2772  12NOV86  11:38
7,21  CUCU
SHAKE           101      46750   8NOV86  16:29
7,21  CUCU
ZZVS493         110      13312  12NOV86  11:38
7,21  CUCU
6
```

The stack dump file is identified by a file code of 111.

3. Use the FUP DUP command at the command interpreter to save the stack dump, the EDIT file, and the recovery file in a special subvolume. For example, to save the files in the subvolume named VSBUG, type:

```
5 FUP DUP AAAA831,VSBUG.*

6 FUP DUP ZZVS493,VSBUG.*

7 FUP DUP MYFILE,VSBUG.*
```

   where AAAA831 is the name of the stack dump file, ZZVS493 is the name of the recovery file, and MYFILE is the name of the EDIT file that you were editing when the error occurred.

4. Copy the subvolume named VSBUG onto tape. Send the tape and the sheet of paper on which you wrote the error message to the Software Development Department at Tandem.

**Recovery Procedure G**

After you reconnect your terminal, you need to determine if EDIT is still running, and if so, stop it. Follow Steps 1, 2, 3, and 5 of Procedure C.

# Index

## C

## D



## E

## F

## G



## I

## J



## K

058061 Tandem Computers Incorporated

## L

**M**

## N

## O



## P

## Q



## R

## S

## T

## U



## V



## W



## X

## Z



## Special Characters