Tektronix®
COMMITTED TO EXCELLENCE

PLEASE CHECK FOR CHANGE INFORMATION AT THE REAR OF THIS MANUAL.

8550
MICROCOMPUTER DEVELOPMENT LAB
EDITOR
Version 4.X
USER'S MANUAL

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number _____

070-3571-00

First Printing JUL 1980
Revised AUG 1981

LIMITED RIGHTS LEGEND

Software License No. _____

Contractor: Tektronix, Inc.
Explanation of Limited Rights Data Identification Method
Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1980 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

TABLE OF CONTENTS

Page

Section 1 LEARNING GUIDE

Introduction 1-1
Editor Overview 1-2
Demonstration Run 1-4
For Continued Learning 1-11

Section 2 OPERATING PROCEDURES

Introduction 2-1
The Essentials 2-2
Displaying Text 2-8
Line Manipulation 2-10
Character String Manipulation 2-14
Block Manipulation 2-18
Command Shortcuts 2-24

Section 3 COMMAND DICTIONARY

Introduction 3-1
Dictionary Page Format 3-1
Terminology 3-1
The Workspace Buffer 3-2
Command Lines 3-3
Special Keys 3-4
Editor Commands 3-6

Section 4 TECHNICAL NOTES
Section 5 ERROR MESSAGES
Section 6 GLOSSARY
Section 7 INDEX

ILLUSTRATIONS

Fig. No.

1-1 The role of the editor in programming 1-2
3-1 The workspace buffer 3-3

Section 1
LEARNING GUIDE

Page

Introduction 1-1

Editor Overview 1-2
Uses of the Editor 1-2
General Features 1-3
The Workspace 1-3

Demonstration Run 1-4
Create a Text File 1-4
How to Correct Input Mistakes 1-4
Set Tabs 1-5
Enter Text 1-5
Display Text 1-6
Save Text in a file 1-6
Modify a File 1-7
Get Text from the File into the Workspace 1-7
Move the Workspace Pointer 1-8
Substitute Text 1-8
Display Editor Line Numbers 1-9
Move a Block of Text 1-9
Find and Change a Character String 1-9
Enter Multiple Commands on a Line 1-10
Print a Copy on the Line Printer 1-10

For Continued Learning 1-11

Illustrations

Fig. No.

1-1 The role of the editor in programming 1-2

Section 1

LEARNING GUIDE

This Learning Guide provides a general overview of the DOS/50 Editor and offers you a simple demonstration for hands-on experience. When you are finished with this Learning Guide, you should be able to do simple editing and be ready to learn more about the editor by reading other sections of the manual.

This Learning Guide is divided into the following topics:

- **Editor Overview.** An explanation of the editor and its general features.
- **Demonstration Run.** How to invoke the editor, enter text, save the text in a file, and reedit an old file.
- **For Continued Learning.** A guide to using other parts of this manual to learn more about the editor.

EDITOR OVERVIEW

Uses of the Editor

The text editor is an important programming tool. It is useful whenever text must be created or modified. You can use the editor to enter and debug source programs, to prepare command files and data files, and to create documents. Figure 1-1 is a diagram of typical work flow, showing suitable editor tasks.

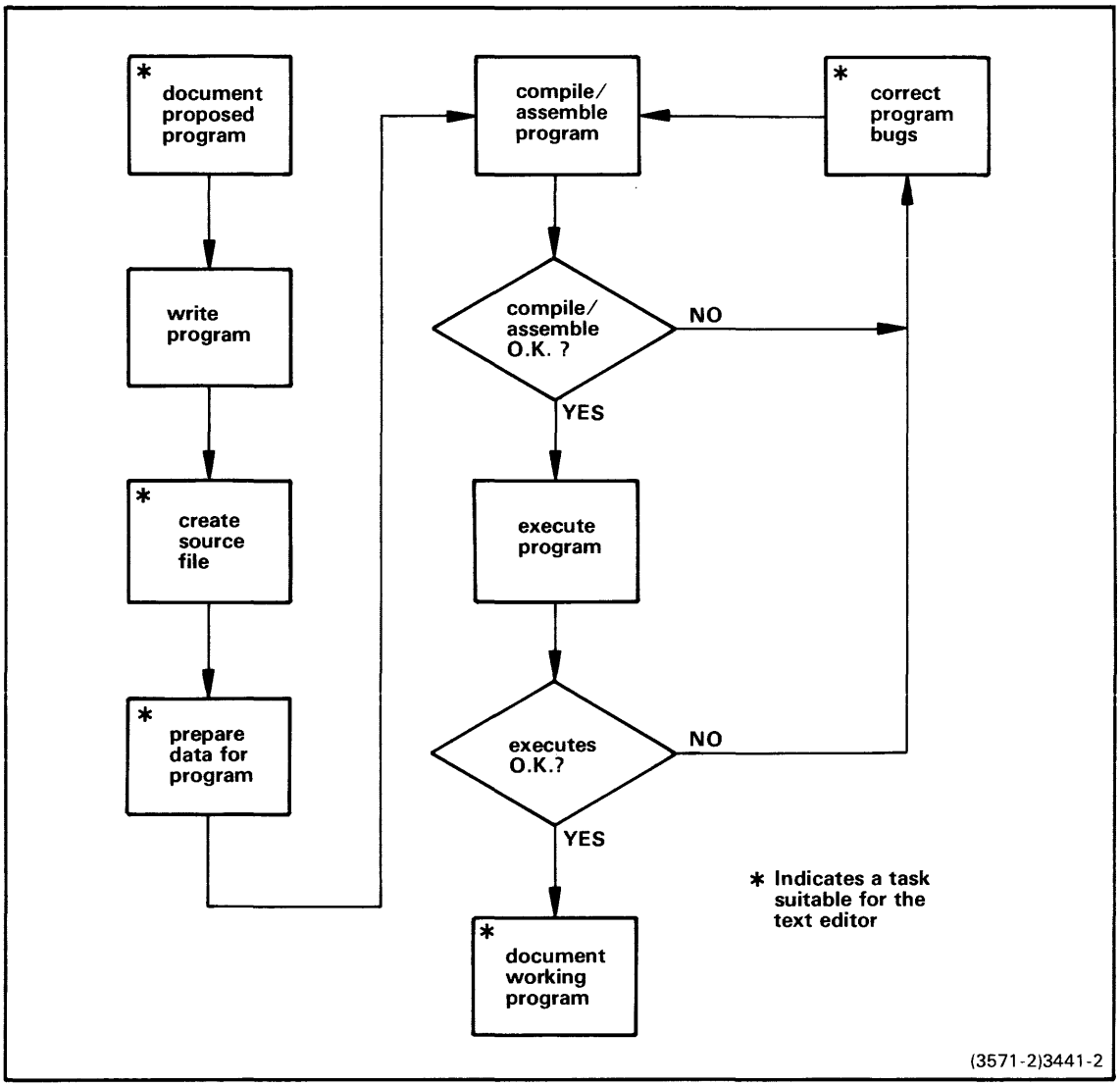


Fig. 1-1. The role of the editor in programming.

The editor is used to create and modify source programs, data files, and documentation. DOS/50 command files and editor command files may also be created with the editor.

General Features

The DOS/50 Editor is line-oriented: most editor commands operate on a single line of text. The editor accepts a command line, executes it, and prompts for another command line. Because the editor is not screen-oriented, it can be used on hard-copy terminals as well as on CRTs.

Here are some of the features of the DOS/50 Editor:

- Text lines can be entered from the keyboard or brought in from files.
- Tab stops can be used to align columns of text.
- Text may be stored in files with tab characters unexpanded.
- Text may contain control characters.
- Strings of characters can be found and modified.
- Blocks of text can be moved or repeated within the file.
- A text line can be referenced either by its number or by its position relative to the workspace pointer. Text can be displayed with or without line numbers.
- A command sequence can be repeated a specified number of times, or until a command within the sequence fails.
- Frequently used command sequences can be stored in command files or command macros and executed as they are needed.
- Entire editing sessions can be performed by command files without operator intervention.
- Files that exceed the size of the editor workspace can be edited conveniently in sections.
- The editor workspace may occupy up to 55K bytes of memory.
- Error messages tell not only what went wrong, but which command or parameter caused the error.
- Several editor features can be turned on or off to suit the needs of the current editing session.
- You may enter new command lines while a previous command line is being executed.

The Workspace

The TEKDOS Editor program uses 9K bytes of program memory. The remaining program memory is available for the workspace buffer. If your 8301 has 64K bytes of program memory, 55K bytes are available for the workspace. This translates to about 1400 lines of text averaging 40 characters each. The text being edited is stored in the workspace. A pointer in the workspace designates one of the lines of text in the workspace as the current line. Most of the editor commands execute relative to that workspace pointer. If there is no text in the workspace, the workspace pointer points to the end-of-text marker.

DEMONSTRATION RUN

In this demonstration run, you use the editor to create and modify a file named TIGERS. Before attempting to use the Editor, make sure you are familiar with the DOS/50 operating system, as described in the 8550 Microcomputer Development Lab System Users Manual.

Create a Text File

To invoke the editor and create the new file in the current directory, enter the following command:

```
> EDIT TIGERS<CR>
```

DOS/50 invokes the editor, creates a new file named TIGERS in the current directory, and sends the following message to the terminal:

```
** EDIT VER 3.0
** NEW FILE
*
```

If the message ** NEW FILE does not appear, the file TIGERS already exists in the directory. If you do not want to alter the existing TIGERS file, enter the editor command QUIT to exit the editor without harming the file. Then reinvoke the editor, using a new file name. If the file TIGERS already exists and you continue with this example, the previous contents of TIGERS will be saved with the new text in the file TIGERS and the old contents of TIGERS wll be saved in file TIGERS#.

The editor prompt character is the single asterisk (*). The asterisk prompt indicates that the editor is ready to accept an editor command.

How to Correct Input Mistakes

Before you begin to input text, you need to know how to correct your mistakes on the input line. You may correct only the line you are currently entering. Once you have entered a carriage return to end the line, no other corrections may be made to that line.

To Correct Characters One-By-One:.

- The BACKSPACE key on a refresh CRT terminal cancels the current character in the input buffer, erases it from the screen, and moves the cursor one space to the left. This may be repeated as many times as necessary.
- The DELETE key on a hard copy terminal cancels the current character in the input buffer and prints the deleted character. This may be repeated as many times as necessary.
- The RUBOUT key cancels the current character in the workspace and either moves the cursor one space to the left (default setting) or echoes the character on the screen, depending on the setting of the editor command CRT. This may be repeated as many times as necessary.

To Replace an Entire Line:.

- Press the escape (ESC) key once to delete the line. Then reenter the line from the beginning.

NOTE

If you accidentally press the ESC key twice in succession, the editor will be suspended and control will return to DOS/50. To notify you that this has occurred, a double DOS/50 prompt(>>) will be displayed on the terminal. To return to the editor enter the DOS/50 command **CONT EDIT or CONT *.** *You may then continue the editing session from the point where the double ESC was entered.*

To intentionally suspend the editor, please use the **SUSPEND** *command rather than the ESC key.*

Set Tabs

For this example you use special tab settings to list items within your text. The editor has default tab settings in columns 8, 16, 24, 32, 40, 48, 56, and 64. For this example, the tabs are reset to columns 4, 8, and 20 before you enter the text. (The special tab settings are not saved between edit sessions, so you may want to enter a remark line in your text to remind you of their location.) The standard tab character is contol-I (sent by the tab key). Because control-I is not a printing character, it is difficult to edit if the need arises. This example shows how to use a substitute tab character that does print. You may define an editor tab character in addition to the standard tab character. However, be sure it is not a character you are using in the text. In this example the percent sign (%) is used. After the text is entered with percent signs for tab characters, the editor tab character definition will be set to the percent sign to show how the tab characters are expanded to spaces. Enter the following command:

```
*TABS 4 8 20<CR>
*
```

Enter Text

You are now ready to enter your text. The new file contains no lines as yet. Since the workspace is empty, the workspace pointer is located at the end-of-text marker. The text that you enter will be inserted before the workspace pointer. Multiple lines of text are entered in input mode.

The editor command INPUT is used for entering text. There is no prompt character in input mode. Enter each new line immediately after the carriage return that terminates the previous line. A carriage return on an empty line terminates input mode.

Enter the following text (your entries underlined):

```
*INPUT<CR>
INPUT:
WHY EVERY FAMILY SHOULD OWN A TIGER:<CR>
%1.%PROTECTION%A FAMILY WITH A TIGER NEED NEVER<CR>
%%%WORRY ABOUT BURGLARS, SALESMEN, OR<CR>
%%%NEIGHBORHOOD DOGS.<CR>
%2.%AFFECTION%A WELL-TRAINED TIGER IS VERY AFFECTIONATE<CR>
%%%TO THE OWNER'S FAMILY.<CR>
%3.%ADMIRATION%A TIGER IS THE ULTIMATE STATUS SYMBOL,<CR>
%%%INSPIRING AWE AND ADMIRATION IN EVERYONE.<CR>
<CR>
*
```

When you enter the carriage return on the empty line, the editor exits from input mode and displays the asterisk prompt, indicating that you may enter the next editor command.

Display Text

After entering your text, you'll want to view it to see if it is correct. The editor command TYPE is used to display text. The TYPE command, without parameters, displays the current line only. To display all of the lines in the workspace, enter TYPE B-E. "B" stands for the beginning of the text in the workspace, while "E" stands for the end. Before you enter the TYPE command, change the tab character so you can view the expanded text.

```
*TAB%<CR>
*
```

The TAB % command causes all percent signs in the text to be expanded to spaces to line up the text in the previously defined tab columns.

```
*TYPE B-E<CR>
WHY EVERY FAMILY SHOULD OWN A TIGER:
   1.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
                    WORRY ABOUT BURGLARS, SALESMEN, OR
                    NEIGHBORHOOD DOGS.
   2.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
                    TO THE OWNER'S FAMILY.
   3.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
                    INSPIRING AWE AND ADMIRATION IN EVERYONE.
*
```

Save the Text in a File

Since the text you've displayed looks correct, go ahead and store it in the file TIGERS which you specified when you invoked the editor. The editor command FILE saves the contents of the workspace in the file and exits from the editor.

```
*FILE<CR>
** END OF TEXT

>
```

The ** END OF TEXT message shows that all the text was written from the workspace to the file. The > prompt character tells you that the editor mode has been terminated and the operating system is ready for the next DOS/50 command.

Modify a File

Let's alter the text you have just created. (The text could have been changed before you stored it, but you'll begin a new editing session to illustrate certain editor commands.) Invoke the editor with the EDIT command:

```
> EDIT TIGERS<CR>
 ** EDIT VER 3.0 **
*
```

Notice that the ** NEW FILE message does not appear here, since the file named TIGERS already exists in the current directory. Although you've invoked the editor, the contents of the file TIGERS are not yet in the workspace. You can verify this by entering the TYPE command:

```
*TYPE B-E<CR>
 ** END OF TEXT
*
```

This shows that the workspace pointer is pointing to the end-of-text marker; there are no lines of text in the workspace.

Get Text from the File into the Workspace

The editor command GET is used to transfer text from the file to the workspace. The GET command, without parameters, gets only the next line of the file. To transfer all the lines, enter the GET command with a parameter equal to or greater than the number of lines in the file. (In this case, 20 is large enough.)

```
*GET 20<CR>
 ** EOF
*
```

The message ** EOF shows that the all of the text in the file TIGERS has been copied into the workspace. The text has been placed before the current line in the workspace. You can use the TYPE command to make sure the text is in the workspace.

```
*TYPE B-E<CR>
WHY EVERY FAMILY SHOULD OWN A TIGER:
%1.%PROTECTION%A FAMILY WITH A TIGER NEED NEVER
%%%WORRY ABOUT BURGLARS, SALESMEN, OR
%%%NEIGHBORHOOD DOGS.
%2.%AFFECTION%A WELL-TRAINED TIGER IS VERY AFFECTIONATE
%%%TO THE OWNER'S FAMILY.
%3.%ADMIRATION%A TIGER IS THE ULTIMATE STATUS SYMBOL,
%%%INSPIRING AWE AND ADMIRATION IN EVERYONE.
*
```

The text looks like it did when we entered it. If we want to see it in tab expansion form, the previous tab settings must be entered.

```
*TAB%<CR>
*TABS 4 8 20<CR>
*TYPE 8-E<CR>
WHY EVERY FAMILY SHOULD OWN A TIGER:
   1.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
                    WORRY ABOUT BURGLARS, SALESMEN, OR
                    NEIGHBORHOOD DOGS.
   2.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
                    TO THE OWNER'S FAMILY.
   3.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
                    INSPIRING AWE AND ADMIRATION IN EVERYONE.
*
```

Move the Workspace Pointer

The position of the workspace pointer affects many of the editor commands. Searches are carried out from the workspace pointer to the end of the text. Any changes are made in the text of the current line, pointed to by the workspace pointer. Text is printed, displayed, or moved with reference to the workspace pointer.

The first place you'll change the text is in the first line. If every family owned a tiger, it would not be a status symbol. Therefore, lets change "every family" to "you". Move the workspace pointer to the first line of text with the BEGIN command:

```
*BEGIN<CR>
WHY EVERY FAMILY SHOULD OWN A TIGER
*
```

When the workspace pointer is moved, the new line that it points to is displayed.

Substitute Text

The editor command for substituting text is SUBSTITUTE. The command is followed by a delimiter (a character not found in either text string), the text to be replaced, a delimiter, the replacement text, and a delimiter. (All three delimiters must be the same character.)

```
*SUBSTITUTE /EVERY FAMILY/YOU/<CR>
WHY YOU SHOULD OWN A TIGER
*
```

The resulting text is displayed. The workspace pointer remains on the same line.

Display Editor Line Numbers

Many of the editor commands refer to line numbers; thus, it can be very useful to know the number of each line. The editor command NUMBER ON causes line numbers to be displayed along with the text.

```
*NUMBER ON<CR>
*TYPE B-E<CR>
    1: WHY YOU SHOULD OWN A TIGER:
    2:     1.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
    3:                      WORRY ABOUT BURGLARS, SALESMEN, OR
    4:                      NEIGHBORHOOD DOGS.
    5:     2.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
    6:                      TO THE OWNER'S FAMILY.
    7:     3.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
    8:                      INSPIRING AWE AND ADMIRATION IN EVERYONE.
```

Move a Block of Text

What if the block of text referring to protection would be better as the last item in the list? To move those three lines, use the editor command MOVE. This command includes two parameters: the range of lines to be moved, and the line before which the moved lines are to be placed. The section to be moved includes lines 2 through 4; the line before which they go is line 9 (the end of text).

```
*MOVE 2-4 9<CR>
** EOF
```

Now use the TYPE command to display the result:

```
*TYPE B-E<CR>
    1: WHY YOU SHOULD OWN A TIGER:
    2:     2.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
    3:                      TO THE OWNER'S FAMILY.
    4:     3.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
    5:                      INSPIRING AWE AND ADMIRATION IN EVERYONE.
    6:     1.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
    7:                      WORRY ABOUT BURGLARS, SALESMEN, OR
    8:                      NEIGHBORHOOD DOGS.
*
```

Find and Change a Character String

Now that you've moved the block of text, the paragraph numbers are in the wrong order. To correct the paragraph numbers, use the FSUBSTITUTE command, which finds and replaces a string of text. The command is followed by a delimiter (a character not found in either text string), the text to be searched for and replaced, a delimiter, the replacement string, and a delimiter. (All three delimiters must be the same character.) Before you enter the FSUBSTITUTE command, first enter the BEGIN command. This starts the search at the first line of the text. The period following BEGIN tells the editor not to display the line, as it normally would. Many commands that display the resulting line may be followed by a period if you do not want the line displayed.

```
*BEGIN.<CR>
*FSUBSTITUTE /1/3/<CR>
    6:     3.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
```

Enter Multiple Commands on One Line

To change each of the remaining two numbers, enter the BEGIN command and the FSUBSTITUTE command on one line. When you enter more than one command on a line, the commands must be separated by a colon (:). Most of the editor commands may be entered in short forms instead of the long forms we have been using so far. This example shows the use of the short forms of the commands. The short form of BEGIN is B ,the short form of FSUBSTITUTE is FS, and the short form of TYPE is T.

```
*B.:FS /3/2/<CR>
    4:     2.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
*B.:FS /2/1/<CR>
    2:     1.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
*T B-E<CR>
    1: WHY YOU SHOULD OWN A TIGER:
    2:     1.  AFFECTION    A WELL-TRAINED TIGER IS VERY AFFECTIONATE
    3:                      TO THE OWNER'S FAMILY.
    4:     2.  ADMIRATION   A TIGER IS THE ULTIMATE STATUS SYMBOL,
    5:                      INSPIRING AWE AND ADMIRATION IN EVERYONE.
    6:     3.  PROTECTION   A FAMILY WITH A TIGER NEED NEVER
    7:                      WORRY ABOUT BURGLARS, SALESMEN, OR
    8:                      NEIGHBORHOOD DOGS.
*
```

Print a Copy on the Line Printer

The text is now in the form you want. You may want to get a permanent copy, using a line printer or other hard copy peripheral attached to the system. To do this, you use the LIST command. The LIST command is much the same as the TYPE command. The line printer or other hard copy unit must, of course, be connected to the 8501 and be ready to receive the output.

```
*LIST B-E<CR>
*
```

No output is sent to the terminal. Only the prompt is displayed to show that the editor is ready for another command.

Now use the FILE command to save the changed text on the file and exit the editor. The changed text will replace the old text in the file. A backup file containing the old text will be created with the name TIGERS#.

```
*FILE<CR>
 ** END OF TEXT
 ** EOF

>
```

FOR CONTINUED LEARNING

In this Learning Guide we explained some elementary editor concepts. For a more detailed explanation, refer to the following sections of this manual:

- **Section 2, Procedures.** Describes a series of tasks and the commands needed to perform those tasks. All of the tasks include examples as well as instructions.
- **Section 3, Command Dictionary.** Provides a formal description of each editor command, its operation, and its syntax. Most command descriptions have illustrative examples. Commands are arranged alphabetically. The DOS/50 command EDIT is also described in detail.
- **Section 5, Error Messages.** Gives a thorough explanation of error messages what to do about them.

Section 2
PROCEDURES

Page

Introduction 2-1

The Essentials 2-2
Invoking the Editor 2-2
Exiting the Editor 2-3
Creating a File 2-4
Modifying an Existing File 2-5
Resetting the Tab Stops 2-6
Expanding Tab Characters to Spaces 2-6

Displaying Text 2-8
Displaying Text from the Workspace 2-8
Displaying Text from a File 2-8
Displaying Line Numbers with the Text 2-9

Line Manipulation 2-10
Moving the Workspace Pointer 2-10
Controlling the Current-Line Display 2-11
Moving Text within the Workspace 2-12
Repeating a Block of Text 2-13

Character String Manipulation 2-14
Finding a String 2-14
Replacing a String 2-15
Making a Global String Replacement 2-16

Block Manipulation 2-18
Editing a Large File by Sections 2-18
Moving Text Forward in the File 2-19
Rearranging Text from One or More Files 2-20
Saving Text on a File 2-21
Adding Text from a File 2-22

Command Shortcuts 2-24
Entering Several Commands on a Line 2-24
Repeating a Command Sequence 2-24
Creating and Using a Command Macro 2-25
Executing Commands from a Command File 2-26

Section 2

OPERATING PROCEDURES

Section 1, the Learning Guide, gave you a general overview of the DOS/50 Editor and presented a simple demonstration run. This section presents some common procedures for using the editor with your 8550 Microcomputer Development Lab.

Each procedure in this section is presented in the following format:

Description: A summary of the action(s) performed by the procedure.

Procedure: The information entered or displayed at the system terminal. The following conventions are used in the procedure description:

Underlined: A character sequence you enter.

No underline: A character sequence displayed by the editor or by DOS/50.

UPPERCASE: An exact character sequence; if these characters are underlined, enter them exactly as shown.

lowercase: A parameter you supply when you perform the procedure.

Parameters: A description of the values you supply.

Comments: The operating limits and options for the procedure.

Examples: One or more demonstrations of correct entry format.

See also: Cross-references to related procedures.

THE ESSENTIALS

Invoking the Editor

Description: This procedure invokes the editor and specifies the file to be created or edited.

Procedure: `> EDIT filespec`

Parameters: **filespec**—The file to be edited.

Comments: The full form of the EDIT command is described in the Command Dictionary section of this manual. The short form

```
EDIT filespec
```

is sufficient for routine applications.

If the specified file does not exist, it is created. If the file already exists, it is modified and the old version is saved under a backup name.

Examples: `> EDIT SUBS`

If SUBS already exists, it becomes the primary input file and a temporary primary output file is created. When you close the editing session with a FILE command, the edited contents of SUBS are copied to the primary output fle. The primary output file is renamed SUBS and the primary input file is renamed SUBS#.

If SUBS did not previously exist, there is no primary input file. The text written to the primary output file must come from the keyboard or from alternate input files. When a FILE command closes the editing session, the primary output file is saved under the name SUBS.

See also:

- Exiting the Editor
- Creating a File
- Modifying an Existing File

Exiting the Editor

Description: There are three ways to return control from the editor to DOS/50:

- The FILE command copies the contents of the workspace and the remainder of the primary input file to the primary output file, saves the primary output file, and terminates the editor.
- The QUIT command terminates the editor without further action. The primary output file is lost unless it existed before the editor was invoked.
- The SUSPEND command causes a temporary exit to DOS/50. To return to the editor, enter CONT * or CONT EDIT; to terminate the editor as with a QUIT command, enter ABORT * or ABORT EDIT.

Procedure:

```
*FILE
```

or

```
*QUIT
```

or

```
*SUSPEND
 .
 .
> CONT *
```

or

```
*SUSPEND
 .
 .
> ABORT *
```

See also:

- Invoking the Editor

Creating a File

Description: This procedure creates a new file from text entered from the system terminal.

Procedure:

```
> EDIT filespec
 ** EDIT VERSION 3.0
 ** NEW FILE
*INPUT
INPUT:
```

Enter your text here.
A carriage return on an empty line terminates input mode.

```
*FILE
 ** END OF TEXT
>
```

Parameters: **filespec**—The filespec of the file to be created.

Examples: In this example, "t" denotes the TAB key.

```
> EDIT DEMO1
 ** EDIT VERSION 3.0
 ** NEW FILE
*INPUT
INPUT:
tTITLEt"8080A DEMONSTRATION RUN PROGRAM"
tORGt100Ht; START PROGRAM AT ADDRESS 100
DEMOtLXItH,500Ht; SET TABLE POINTER
tMVItB,5t; SET PASS COUNTER
tXRAtAt; CLEAR ACCUMULATOR
LOOPtADDtMt; ADD BYTE FROM TABLE
<CR>
*FILE
 ** END OF TEXT

>
```

With the default tab stops in effect, the text looks like this when it is displayed:

```
        TITLE   "8080A DEMONSTRATION RUN PROGRAM"
        ORG     100H    ; START PROGRAM AT ADDRESS 100
DEMO    LXI     H,500H  ; SET TABLE POINTER
        MVI     B,5     ; SET PASS COUNTER
        XRA     A       ; CLEAR ACCUMULATOR
LOOP    ADD     M       ; ADD BYTE FROM TABLE
```

See also:

- Modifying an Existing File
- Resetting the Tab Stops
- Expanding Tab Characters to Spaces

Modifying an Existing File

Description: This procedure replaces a file created previously (by the editor or otherwise) with an edited version.

Procedure:

```
> EDIT filespec
 ** EDIT VERSION 3.0
*NEXT
 ** EOF
*                    Make the necessary changes in the file.
*FILE
 ** END OF TEXT
 ** EOF

>
```

Parameters: **filespec**—The filespec of the file to be modified.

Comments: The file is modified according to the editor commands you enter. The old version is given a backup file name.

The NEXT command brings the contents of the file into the workspace, moves the workspace pointer to the first line, and displays that line.

Examples:

```
> EDIT SHORTY
 ** EDIT VERSION 3.0
*NEXT
 ** EOF
THIS IS LINE ONE OF SHORTY.
*SUBSTITUTE/SHO/THE NEW SHO/
THIS IS LINE ONE OF THE NEW SHORTY.
*FILE
 ** END OF TEXT
 ** EOF

>
```

File SHORTY is modified. The new version of the file is named SHORTY and the old version is named SHORTY#.

See also:

- Creating a File
- Editing a Large File by Sections

Resetting the Tab Stops

Description: This procedure sets the tab stops to the columns you want.

Procedure:
```
*TABS n1 n2 n3 n4 n5 n6 n7 n8
```

Parameters: **n1** .. **n8**—The column numbers of the tab stops.

Comments: When you invoke the editor, the eight tab stops are at columns 8, 16, 24, 32, 40, 48, 56, and 64. You may specify up to eight new tab stops; all existing tab stops are lost. The column numbers must be in ascending order and in the range 1 to 127. Any text you display is aligned to the latest set of tab stops.

Examples:
```
*TABS 10 20 35
```

This command deletes the existing tab stops and sets new tab stops at columns 10, 20, and 35.

See also:
- Creating a File
- Expanding Tab Characters to Spaces

Expanding Tab Characters to Spaces

Description: When you use tab stops in creating a file, the tab characters you enter remain in the text. This procedure expands each tab character in the file into the appropriate number of spaces.

Procedure:
```
> EDIT filespec
 ** EDIT VERSION 3.0
*TAB t
*TABS n1 n2 n3 n4 n5 n6 n7 n8
*XTABS ON
*FILE
 ** END OF TEXT
 ** EOF

>
```

Parameters: **filespec**—The filespec of the file containing tab characters.

t—The tab character you used in creating the file. If you used the standard tab character (CTRL-I), omit the TAB command.

n1 .. **n8**—The tab stops to which the text is to be aligned. If you want to use the standard tab stops (columns 8, 16, 24, 32, 40, 48, 56, and 64), omit the TABS command.

Comments: The command XTABS ON tells the editor that any tab character copied from the workspace is to be replaced by spaces up to the next tab stop. The FILE command copies the text from the primary input file, **through the workspace**, to the primary output file. Each tab character is replaced by spaces as it is copied to the primary output file.

NOTE

The standard tab character is an acceptable delimiter in MDL and ASM source files, but may produce unexpected results in other types of files. A file containing non-standard tab characters (defined with the TAB command) is usually useless.

Examples:

```
> COPY TIGERS
WHY YOU SHOULD OWN A TIGER:
%1.%AFFECTION%A WELL-TRAINED TIGER IS VERY AFFECTIONATE
%%%TO THE OWNER'S FAMILY.

> EDIT TIGERS
 ** EDIT VERSION 3.0
*TAB %
*TABS 4 8 20
*XTABS ON
*FILE
 ** END OF TEXT
 ** EOF

> COPY TIGERS
WHY YOU SHOULD OWN A TIGER:
   1.  AFFECTION   A WELL-TRAINED TIGER IS VERY AFFECTIONATE
                   TO THE OWNER'S FAMILY.

>
```

See also:

- Creating a File
- Resetting the Tab Stops

DISPLAYING TEXT

Displaying Text from the Workspace

Description: This procedure displays workspace text on the system terminal or the line printer.

Procedure:

```
*TYPE lines
```

or

```
*LIST lines
```

Parameters: **lines**—The range or number of lines you want to display. If no parameter is specified, only the current line is displayed.

Examples:

```
*TYPE 20
```

This command displays 20 lines on the system terminal, starting with the current line.

```
*LIST 15-25
```

This command displays lines 15 through 25 on the line printer.

See also:

- Displaying Text from a File
- Displaying Line Numbers with the Text

Displaying Text from a File

Description: This procedure displays the contents of a file without affecting the text in the workspace.

Procedure:

```
*COPY lines filespec CONO
```

or

```
*COPY lines filespec LPT
```

Parameters: **lines**—The range or number of lines to be displayed.

filespec—The filespec of the file to be displayed.

Comments: CONO specifies the system terminal; LPT specifies the line printer.

Examples:

```
*COPY 101-120 MYFILE CONO
```

This command skips over the first 100 lines of MYFILE and displays the next 20 lines on the system terminal.

```
*COPY 999 HERFILE LPT
```

This command displays the first 999 lines of HERFILE on the line printer. If HERFILE contains fewer than 999 lines, the entire file is listed.

See also:

- Displaying Text from the Workspace

Displaying Line Numbers with the Text

Description: This procedure causes each text line displayed by the editor to be accompanied by its line number.

Procedure:

```
*NUMBER ON
```

Comments: To suppress the display of line numbers, enter the command NUMBER OFF.

Examples:

```
*TYPE 3
THURSDAY
FRIDAY
SATURDAY
*NUMBER ON:TYPE 3
    5: THURSDAY
    6: FRIDAY
    7: SATURDAY
*BEGIN
    1: SUNDAY
*NUMBER OFF:DOWN
MONDAY
```

LINE MANIPULATION

Moving the Workspace Pointer

Description: The editor provides several commands for moving the workspace pointer.

BEGIN moves the pointer to the first line of text. END moves the pointer past the last line. UP and DOWN move the pointer up and down the text, respectively.

FIND and FNEXT move the pointer to the next line containing a specified string of characters. To find the string, FIND searches only the text in the workspace. FNEXT searches the workspace, then the rest of the primary input file, until the string is found.

Procedure:

```
*BEGIN
```

or

```
*END
```

or

```
*UP n
```

or

```
*DOWN n
```

or

```
*FIND/string/
```

or

```
*FNEXT/string/
```

Parameters: **n**—The number of lines up or down the pointer is to move. If this parameter is omitted, the pointer moves one line.

/—The string delimiter: any character (except a space) not occurring in the string.

string—The string of characters to be found.

See also:

- Finding a String

Controlling the Current-Line Display

Description: Each of the following commands displays the current line after executing successfully:

BEGIN	END	FNEXT	NEXT	SUBSTITUTE
DOWN	FIND	FSUBSTITUTE	REPLACE	UP

This current-line display is controlled by the BRIEF flag.

Procedure:

```
*BRIEF ON
```

or

```
*BRIEF OFF
```

Comments: When the BRIEF flag is at its default setting (OFF), the current line is displayed whenever any of the above commands is executed. After you enter BRIEF ON, those commands will not automatically display the current line.

You may enter a period immediately after any of the above commands to reverse the BRIEF flag for that command.

Examples:

```
*BEGIN
LINE ONE          The new current line, line 1, is displayed.
*DOWN 3
LINE FOUR
*DOWN.3           The period suppresses display of line 7.
*BRIEF ON
*BEGIN            Line 1 is not displayed because the BRIEF flag is ON.
*DOWN 3           Line 4 is not displayed because the BRIEF flag is ON.
*DOWN.3
LINE SEVEN        The period reverses the BRIEF flag temporarily.
*BRIEF OFF
*UP
LINE SIX
*UP.              The period suppresses display of line 5.
```

Moving Text within the Workspace

Description: This procedure deletes a section of text from the workspace and inserts it elsewhere in the workspace.

Procedure: *MOVE lines n

Parameters: **lines**—The number or range of lines to be moved.

n—The moved text is inserted in front of line **n**. If **n** is omitted, the text is moved to the end of the workspace.

Examples:

```
*NUMBER ON:BEGIN.:TYPE 12
    1: JANUARY
    2: FEBRUARY
    3: MARCH
    4: APRIL
    5: MAY
    6: JUNE
    7: JULY
    8: AUGUST
    9: SEPTEMBER
   10: OCTOBER
   11: NOVEMBER
   12: DECEMBER
*MOVE 6-8 2
*TYPE 12
    1: JANUARY
    2: JUNE
    3: JULY
    4: AUGUST
    5: FEBRUARY
    6: MARCH
    7: APRIL
    8: MAY
    9: SEPTEMBER
   10: OCTOBER
   11: NOVEMBER
   12: DECEMBER
```

The MOVE command moves lines 6 through 8 in front of line 2. Lines are renumbered accordingly.

See also:

- Moving Text Forward in the File
- Rearranging Text from One or More Files

Repeating a Block of Text

Description: This procedure inserts copies of a selected section of workspace text at specified points in the workspace.

Procedure:

```
*SAVE lines
*BEGIN
*FIND/string1/:UNSAVE
*FIND/string2/:UNSAVE
 .
 .
*FIND/stringN/:UNSAVE
```

Parameters: **lines**—The number of lines or range of lines to be duplicated. The lines are inserted in front of each destination line.

stringN—A unique string in the Nth destination line.

/—The string delimiter: any character (except a space) not occurring in the string.

Comments: Each FIND-UNSAVE command line moves the workspace pointer to a new destination line and inserts a copy of the selected text in front of it.

Examples: The following command sequence inserts a line of dashes in front of every line in the workspace that contains the assembler directive ORG. (The line of dashes begins with a semicolon because it is an assembler comment line.)

```
*INSERT ;----------------------------------------
*UP.
*SAVE 1
*KILL
*BEGIN.
**<FIND/ ORG /:UNSAVE:DOWN.>
```

The INSERT-UP-SAVE command sequence creates a line of dashes and copies the line to the save area. KILL deletes the saved line from the workspace. BEGIN moves the workspace pointer to line 1. The FIND-UNSAVE-DOWN command sequence finds the next line containing the word ORG and inserts the line of dashes in front of that line. The repeat brackets around the FIND-UNSAVE-DOWN sequence indicate that the sequence is to be repeated until the FIND command fails.

Note that if the text contains tab characters, the search string should be /tORGt/, not / ORG /. (The "t" represents whatever tab character is in the text.)

See also:

- Finding a String

CHARACTER STRING MANIPULATION

Finding a String

Description: The commands FIND and FNEXT each move the workspace pointer to the next line containing a specified string.

If the string does not occur in the workspace at or after the current line, the FIND command responds NOT FOUND, but the FNEXT command keeps searching. FNEXT appends the workspace contents to the primary output file, loads up the workspace from the primary input file, and searches the new contents of the workspace. FNEXT stops when the section of text containing the string is in the workspace, or when the primary input file has been searched to its end.

Procedure:

```
*FIND/string/
```

or

```
*FNEXT/string/
```

Parameters: /—The string delimiter: any character (except a space) not occurring in the string.

string—The string of characters to be found.

Examples:

```
*NUMBER ON
*BEGIN.:TYPE 2
    1:AARDVARK, ADAM
    2:ACORN, ALICE
*FIND/BUFF/
   80:BUFFALO, WALTER
*BEGIN.:FNEXT/BUFF/
   80:BUFFALO, WALTER
```

The string "BUFF" occurs in the workspace, so it is found either by FIND or by FNEXT.

```
*FIND/BRU/
** NOT FOUND
*FNEXT/BRU/
   43:SPRUCE, BRUCE
```

The string "BRU" does not occur in the workspace after line 80, so FNEXT searches the primary input file, section by section. The string is finally found in line 43 of some later section.

See also:

- Replacing a String
- Moving the Workspace Pointer

Replacing a String

Description: The commands SUBSTITUTE and FSUBSTITUTE each find a string of characters in the text and replace that string with a new string. Each command searches for the string beginning at the current line and replaces only the first occurrence of the string.

If the string does not occur in the current line, SUBSTITUTE responds NOT FOUND, but FSUBSTITUTE keeps searching to the end of the workspace. If the string is found, the line that contains it becomes (or remains) the current line, the new string replaces the old string, and the modified line is displayed.

Procedure:

```
*SUBSTITUTE/oldstring/newstring/
```

or

```
*FSUBSTITUTE/oldstring/newstring/
```

Parameters: /—The string delimiter: any character (except a space) not occurring in either string.

oldstring—The string to be replaced.

newstring—The string that replaces **oldstring**.

Comments: SUBSTITUTE and FSUBSTITUTE may be abbreviated S and FS, respectively.

Examples:

```
*TYPE 4
THE CURENT LINE
THE NEXTL INE
THE AFTER THAT
THE LASTEST LINE
*S/R/RR/
THE CURRENT LINE
*FS/TL /T L/
THE NEXT LINE
*S/AFTER/ONE AFTER/
 ** NOT FOUND
*FS/AFTER/ONE AFTER/
THE ONE AFTER THAT
*FS/EST//
THE LAST LINE
*UP.3:TYPE 4
THE CURRENT LINE
THE NEXT LINE
THE ONE AFTER THAT
THE LAST LINE
```

See also:

- Finding a String
- Making a Global String Replacement

Making a Global String Replacement

Description: This procedure replaces every occurrence of a specified string of characters with a new string.

Procedure:

```
*BEGIN
**<FS/oldstring/newstring/>
```

or

```
*BEGIN
**<FS/oldstring/newstring/>  } Repeat these two lines
*NEXT                        } as necessary.
```

Parameters: /—The string delimiter: any character (except a space) not occurring in either string.

oldstring—The string to be replaced.

newstring—The string that replaces **oldstring**.

Comments: The first form of the procedure replaces all occurrences of **oldstring** in the workspace. The second form replaces all occurrences in the workspace **and** in the text from the remainder of the primary input file.

The FSUBSTITUTE command (short form FS) repeats until it fails to find a match. Each time a replacement is made, the modified line is displayed.

When there are no more occurrences of **oldstring** in the workspace, the message NOT FOUND is displayed. If you want to continue the global replacement past the current workspace contents, enter the NEXT command to bring the next section of text from the primary input file into the workspace. Then enter the FS command line again.

CAUTION

Either form of this procedure can be very destructive if you choose your search and replacement strings carelessly. Note the following points:

- *A search string that is too general may lead to unexpected replacements.*
- *A capital letter never matches a lowercase letter, and vice versa.*
- *The replacement string should not contain the search string. For example, the command line*

  ```
  5<FS/MAN/WOMAN/>
  ```

  replaces the first occurrence of "MAN" with "WOWOWOWOWOMAN".

Examples:

```
*TYPE B-E
A PROGRAM
THIS RAM AND THAT RAM
A PARAMETER
4K RAM
THE LOS ANGELES RAMS
*BEGIN
A PROGRAM
**<FS/RAM/rom/>
A PROGrom
THIS rom AND THAT RAM
THIS rom AND THAT rom
A PAromETER
4K rom
THE LOS ANGELES romS
 ** NOT FOUND
```

Every occurrence of "RAM" in the workspace is replaced by "rom". BEGIN moves the workspace pointer to the line 1 and displays that line. The FS command repeats until all occurrences of "RAM" in the workspace have been replaced. The modified line is displayed after each FS command.

See also:

- Replacing a String
- Repeating a Command Sequence

BLOCK MANIPULATION

Editing a Large File by Sections

Description: This procedure allows you to edit a file that is too large to fit into the workspace. The editor program occupies about 9K bytes of program memory, and the rest of program memory is available for the workspace. For example, if your 8301 has 64K of program memory, a file that occupies about 55K (1400 lines averaging 40 characters per line) can be stored in the workspace in one piece.

Procedure:

```
> EDIT filespec
 ** EDIT VERSION 3.0
*NEXT n
            Edit the first n lines of text.
*NEXT n
            Edit the next n lines of text.
*NEXT n
            Repeat until the entire file has been edited.
*FILE
 ** END OF TEXT
 ** EOF

>
```

Parameters:

filespec—The filespec of the large file you want to edit.

n—The number of lines in the next block of text to be edited. **n** may vary from one block to the next. If **n** is omitted, the workspace is filled to three-fourths of its capacity.

Comments: Each NEXT command appends the workspace contents (if any) to the primary output file, brings the next **n** lines from the primary input file into the workspace, and displays the new current line (line 1).

Examples:

```
> EDIT BIG
 ** EDIT VERSION 3.0
*NEXT 200
THE 1ST LINE IN THE FILE
*           Edit lines 1-200.
*NEXT 300
THE 201ST LINE IN THE FILE
*           Edit lines 201-500.
*NEXT 150
THE 501ST LINE IN THE FILE
*           Edit lines 501-650.
*FILE
 ** END OF TEXT
 ** EOF

>
```

See also:

- Modifying an Existing File

Moving Text Forward in the File

Description: This procedure deletes a section of text from the workspace and inserts it later in the file being edited.

Procedure:

```
*SAVE lines
*KILL lines
*FNEXT/destination-string/
```

The editor searches for the destination line.
The editor displays the destination line.

```
*UNSAVE
```

Parameters:

lines—The number of lines or range of lines to be moved.

destination-string—A unique string in the destination line. The text being moved is inserted in front of the destination line.

/—The string delimiter: any character (except a space) not occurring in the string.

Comments: This procedure is used to move text forward to that portion of the file that has not yet been brought into the workspace. Use the procedure "Rearranging Text from One or More Files" to move text backward in a large file. If the workspace contains both the text being moved and the destination line, use the MOVE command instead of either of these procedures.

The SAVE command copies the text being moved into the save area, and the KILL command deletes that text from the workspace. The FNEXT command searches for the destination line, first in the workspace, then in the rest of the primary input file. When the destination line is found, it is displayed as the new current line. The UNSAVE command inserts the text being moved into the workspace in front of the destination line.

Examples: In this example, the lines of the file are numbered for clarity. Lines 1001-1050 are moved forward in the file to line 4000.

```
*TYPE 3
LINE 1001
LINE 1002
LINE 1003
*SAVE 50
*KILL 50
*FNEXT/4000/
 ** END OF TEXT
 ** WORKSPACE FULL
LINE 4000
*UNSAVE
```

The SAVE command copies lines 1001-1050 into the save area, and the KILL command deletes those lines from the workspace. The FNEXT command appends the workspace contents to the primary output file, then brings the next block of text from the primary input file into the workspace. That block contains line 4000. The editor moves the workspace pointer to line 4000 and displays that line. The UNSAVE command inserts the contents of the save area (lines 1001-1050) in front of line 4000.

See also:

- Finding a String
- Moving Text within the Workspace
- Rearranging text from One or More Files

Rearranging Text from One or More Files

Description: This procedure copies sections of text from one or more existing files onto a new file. You may use this procedure to move sections of text forward or backward in a file, or to concatenate sections from several different files.

Procedure:

```
> EDIT newfile
 ** EDIT VERSION 3.0
 ** NEW FILE
*COPY lines1 file1
*COPY lines2 file2
 .
 .
*COPY linesN fileN
*FILE
 ** END OF TEXT

>
```

Parameters: **newfile**—The file on which the various sections of text are to be stored. **newfile**, which is created in this editing session, is the primary output file. There is no primary input file.

fileN—The file from which text is copied by the Nth COPY command.

linesN—The range of lines copied by the Nth COPY command.

Comments: **file1, file2, ... fileN** may all be the same file. Each COPY command appends the specified text to the primary output file, which is the default output file for the COPY command.

Examples:

```
> EDIT SORTED
 ** EDIT VERSION 3.0
 ** NEW FILE
*COPY 1-100 UNSORTED
*COPY 601-700 UNSORTED
*COPY 101-600 UNSORTED
*FILE
 ** END OF TEXT

>
```

The first 700 lines of file UNSORTED are rearranged and stored on the newly created file called SORTED. Lines 601-700 of UNSORTED are inserted after line 100.

See also:

- Moving Text within the Workspace
- Moving Text Forward in the File
- Saving Text on a File
- Adding Text from a File

Saving Text on a File

Description: This procedure copies text from the workspace to a file.

Procedure:

```
*PUT lines filespec
```

or

```
*PUTK lines filespec
```

Parameters: **lines**—The number or range of lines to be copied. If you omit this parameter, only the current line is copied.

filespec—The filespec of the file to which the text is copied. If the file does not exist, it will be created, If the file already exists, its previous contents are replaced by the copied text. If you omit this parameter, the text is appended to the primary output file.

Comments: PUT does not affect the workspace contents or pointer. PUTK deletes the copied text from the workspace; if the current line is deleted, the first line after the deleted text becomes the current line.

You may also use the commands FILE, FNEXT, or NEXT to copy text from the workspace to the primary output file.

Examples: `*PUTK B-E`

This command appends the workspace contents to the primary output file and clears the workspace.

`*PUT 40-73 TEMP`

This command saves workspace lines 40 through 73 on file TEMP. Any previous contents of TEMP are lost.

See also:
- Adding Text from a File
- Rearranging Text from One or More Files

Adding Text from a File

Description: This procedure inserts text from a file into the workspace in front of the current line.

Procedure: `*GET lines filespec`

Parameters: **lines**—The number or range of lines to be copied. If you omit this parameter, only one line is copied.

filespec—The file from which the text is copied. If you omit this parameter, text is copied from the primary input file.

Comments: You may also use the commands FNEXT or NEXT to bring text into the workspace from the primary input file.

Examples:

```
*GET 100
```

This command brings the next 100 lines from the primary input file into the workspace.

```
*GET 100 TEMP
```

This command brings the first 100 lines from file TEMP into the workspace.

```
*GET 201-300
```

This command skips over the first 200 lines of the primary input file and brings the next 100 lines into the workspace. Afterward, the primary input file pointer is at line 301.

See also:

- Saving Text on a File
- Rearranging Text from One or More Files

COMMAND SHORTCUTS

Entering Several Commands on a Line

Description: This procedure strings together several commands on a single command line.

Procedure: `*command:command:command`

Parameters: **command**—Any editor command.

Comments: A command line may contain any number of editor commands as long as the command line does not exceed 127 characters. The commands must be separated by colons (:). If the editor finds an error, all subsequent commands in the line are ignored.

Each of the following commands must be the last command on any command line on which it occurs.

COMMENT FILE PERFORM QUIT SUSPEND

Examples: `*BEGIN.:FIND/ERROR/:UP`

This command displays the first line containing the string "ERROR", then moves the workspace pointer to the preceding line and displays that line.

See also:
- Repeating a Command Sequence
- Creating and Using a Command Macro

Repeating a Command Sequence

Description: This procedure allows you to execute a sequence of commands a specified number of times without reentering the command sequence.

Procedure: `*n<command-sequence>`

Parameters: **command-sequence**—A sequence of editor commands separated by colons (:).

n—The number of times the command sequence is to be executed. If **n** is an asterisk (*), the command sequence repeats until a string search command (FIND, FNEXT, FSUBSTITUTE, or SUBSTITUTE) in the sequence fails to find a match, or until you press the ESC (escape) key.

Comments: A repeated command sequence may be nested within another repeated sequence.

Examples:

```
*10<LIST B-E>
```

This command line lists the workspace contents on the line printer 10 times.

```
**<FIND/BYTE/:DOWN.>
```

This command line displays every line containng the string "BYTE" on the system terminal. (The first asterisk is the editor prompt character.)

See also:
- Creating and Using a Command Macro

Creating and Using a Command Macro

Description: This procedure stores a frequently-used command line (called a macro) and assigns it an identification number. That command line is executed whenever its identification number is specified in a MACRO command.

Procedure: To create a macro:

```
*MACRO n=command-line
```

To execute that macro:

```
*MACRO n
```

To list all currently defined macros:

```
*MACRO
```

Parameters: **command-line**—A sequence of editor commands separated by colons (:).

n—The macro identification number: any integer in the range 1 to 127.

Examples:

```
*MACRO10=BEGIN.:TYPE 6:END:UP.3:TYPE 3
 .
 .
*M10
THESE
ARE
THE
FIRST
SIX
LINES
 ** END OF TEXT
THESE ARE
THE LAST
THREE LINES
```

Macro number 10 is defined as a command line to list the first six lines and the last three lines in the workspace. Those lines are listed whenever macro number 10 is invoked. MACRO may be abbreviated M.

See also:

- Entering Several Commands on a Line
- Repeating a Command Sequence

Executing Commands from a Command File

Description: This procedure causes the editor to execute commands from a file instead of from the system terminal.

Procedure:

```
*PERFORM comfile
```

or

```
> EDIT infile outfile comfile
```

Parameters:

comfile—The filespec of the editor command file.

infile—The filespec of the primary input file.

outfile—The filespec of the primary output file.

Comments: The editor command PERFORM causes the editor to begin executing commands from the specified file. When the end of the file is reached or an error is detected, the editor prompts for a command from the system terminal.

The initial command file (specified by the third parameter of the EDIT command) is automatically executed at the beginning of the editing session.

You may use the editor to create a command file in the same way you create any other text file.

Examples: The following editing session creates a command file called TOUCHDOW. TOUCHDOW moves the first line containing the string "FOOTBALL" to the end of the workspace.

```
> EDIT TOUCHDOW
 ** EDIT VERSION 3.0
 ** NEW FILE
*INPUT
INPUT:
COMMENT -- HIKE!
BEGIN.:FIND/FOOTBALL/
MOVE 1 E
COMMENT -- YAY!
<CR>
*FILE
 ** END OF TEXT

>
```

TOUCHDOW may be executed in any subsequent editing session. For example:

```
*TYPE B-E
END ZONE
THE FOOTBALL
LINEBACKER
SAFETY
END ZONE
*PERFORM TOUCHDOW
*COMMENT -- HIKE!
*BEGIN.:FIND/FOOTBALL/
THE FOOTBALL
*MOVE 1 E
*COMMENT -- YAY!
*TYPE B-E
END ZONE
LINEBACKER
SAFETY
END ZONE
THE FOOTBALL
```

Assume that file STARTUP contains the following text:

```
COMMENT -- Command file STARTUP
COMMENT -- This command file begins a standard editing session
COMMENT --    for I.  M.  DeProgrammer.
COMMENT -- Set selected editor flags:
BRIEF ON:ECHO OFF:NUMBER ON:XSEARCH ON:XTABS ON
COMMENT -- Select the tab stops and the editor tab character:
TABS 3 5 7 9 11 13 15 60
TAB @
COMMENT -- Define standard macros:
  COMMENT -- Macro 1 displays the first 5 lines and last 5
  COMMENT --    lines in the workspace.
  MACRO 1 = BEGIN.:TYPE 5:END:UP.5:TYPE 5
  COMMENT -- Macro 2 displays the workspace map.
  MACRO 2 = DEBUG ON:DEBUG OFF
  COMMENT -- Macro 3 displays the editor status, the workspace
  COMMENT --    map, and all currently defined macros.
  MACRO 3 = STATUS:MACRO 2:MACRO
COMMENT -- End of STARTUP.
```

The following command begins an editing session that creates or modifies a file called PROG:

```
> EDIT PROG,,STARTUP
```

The commands in STARTUP are executed, then the editor prompts for a command from the system terminal. At that point, the desired editor features have been turned ON or OFF, the tab stops and tab character have been selected, and three macros have been defined.

Section 3
COMMAND DICTIONARY

COMMAND INDEX

Page

Pointer Movement Commands
BEGIN—Moves the pointer to the first line in the workspace ... 3-7
DOWN—Moves the pointer toward the end of the workspace ... 3-17
END—Moves the pointer to the end of the workspace ... 3-24
UP—Moves the pointer toward the beginning of the workspace ... 3-68

Character String Editing Commands
FIND—Searches the workspace for a string ... 3-27
FNEXT—Searches the primary input file for a string ... 3-29
FSUBSTITUTE—Finds and replaces a string ... 3-32
SUBSTITUTE—Replaces a string in the current line ... 3-58
XSEARCH—Controls the wildcard search feature ... 3-72

Line Editing Commands
INPUT—Inputs text from the system terminal ... 3-36
INSERT—Inserts a line of text from the system terminal ... 3-38
KILL—Deletes lines of text from the workspace ... 3-39
MOVE—Moves lines of text within the workspace ... 3-44
REPLACE—Replaces the current line ... 3-54
SAVE—Saves lines of text in the save area ... 3-55
UNSAVE—Retrieves the text in the save area ... 3-67

File Manipulation Commands
COPY—Copies text from a file or device to a file or device ... 3-11
EDIT—Invokes the editor ... 3-19
FILE—Closes the editing session ... 3-26
GET—Copies text from a file into the workspace ... 3-34
NEXT—Brings the next section of text into the workspace ... 3-45
PUT—Copies text from the workspace to a file or device ... 3-51
PUTK—Moves text from the workspace to a file or device ... 3-52
XTABS—Controls expansion of tab characters on output ... 3-74

Display Commands
BRIEF—Controls display of the current line ... 3-8
COMMENT—Precedes a comment in a command file ... 3-10
ECHO—Controls display of command file commands ... 3-18
ERROR—Controls display of the error pointer ... 3-25

Page

Display Commands (cont.)
LIST—Lists text on the line printer ... 3-40
NUMBER—Controls display of lines numbers ... 3-47
TYPE—Displays text on the system terminal ... 3-66

Utility Commands
AGAIN—Repeats the last repeatable command ... 3-6
CRT—Controls the function of the RUBOUT key ... 3-13
DEBUG—Controls display of the workspace map ... 3-14
LN—Displays the current line number and workspace length ... 3-41
MACRO—Defines or executes a command macro ... 3-42
PERFROM—Executes commands from a command file ... 3-49
QUIT—Aborts the editing session ... 3-53
STATUS—Displays information about the status of the editor ... 3-56
SUSPEND—Suspends the editor ... 3-60
TAB—Defines the editor tab character ... 3-61
TABS—Sets new tab stops ... 3-64
UPARROW—Controls the representation of control characters ... 3-69

Illustrations

Fig. No.

3-1 The workspace buffer ... 3-3

Section 3

COMMAND DICTIONARY

INTRODUCTION

This Command Dictionary alphabetically lists and describes in detail the DOS/50 Editor commands. EDIT, the operating system command that invokes the editor, is also described in this dictionary. EDIT is listed in alphabetical order between editor commands ECHO and END.

Dictionary Page Format

Each command entry contains a syntax block, parameter definitions, a general explanation, and examples.

This Command Dictionary uses the same syntax block conventions as the Command Dictionary in the 8550 Microcomputer Development Lab System Users Manual:

- The underlined part of a command name is the shortest form of the command recognized by the editor.
- Parameters enclosed in braces { } are required.
- Parameters enclosed in brackets [ ] may be omitted.
- Parameters stacked one above the other are alternate forms of the same parameter.

The PARAMETERS section of each entry outlines the functions and limitations of the parameters. The uses and functions of the command are summarized in the EXPLANATION section and demonstrated in the EXAMPLES section.

Terminology

The **primary input file** and **primary output file** are the default files for those editor commands that copy text to or from files: COPY, GET, PUT, and PUTK. The editor commands FILE, FNEXT, and NEXT operate on the primary input file and primary output file exclusively. When a file is edited, the old version of the file is the primary input file and the edited version is the primary output file.

Any other file that provides or receives text through one of the above commands is termed an **alternate input file** or **alternate output file**.

The **initial command file** contains editor commands that are executed before the editor accepts commands from the system terminal. The initial command file might contain a few initializing commands or a whole editing session.

Primary files, alternate files, and the initial command file are discussed in more detail under the EDIT command.

The **workspace** is a section of program memory that holds the text currently being edited. The **workspace pointer** always points to one line of text in the workspace, called the **current line**. Many editor commands operate on the current line; other commands move the workspace pointer, designating a new current line.

A text line may be referenced by its **line number**, which indicates the line's position in the workspace. The top line of text is designated line 1, the next line down is line 2, etc. With certain editor commands, the following three letters may be used as line numbers: B (beginning of workspace), C (current line), and E (end of workspace).

The Workspace Buffer

The workspace buffer is a section of program memory used by the editor to store text. The workspace buffer contains the four areas listed below. Figure 3-1 shows how these four areas are arranged in the workspace buffer.

- The **workspace** contains the text being edited. As you add text to the workspace, the workspace grows toward the end of memory.
- The **save area** contains the text set aside by the latest SAVE command. The size of the save area corresponds to the size of the block of text most recently saved.
- The **macro area** contains any macros defined with the MACRO command. As you define new macros, the macro area expands, pushing the save area toward the beginning of memory.
- The **free area** is the section of available memory between the workspace and the save area.

The editor program occupies about 9K bytes of program memory. The rest of program memory constitutes the workspace buffer. In a system with 64K of program memory, 55K bytes of text (about 1400 lines averaging 40 characters each) may be stored in the workspace buffer.

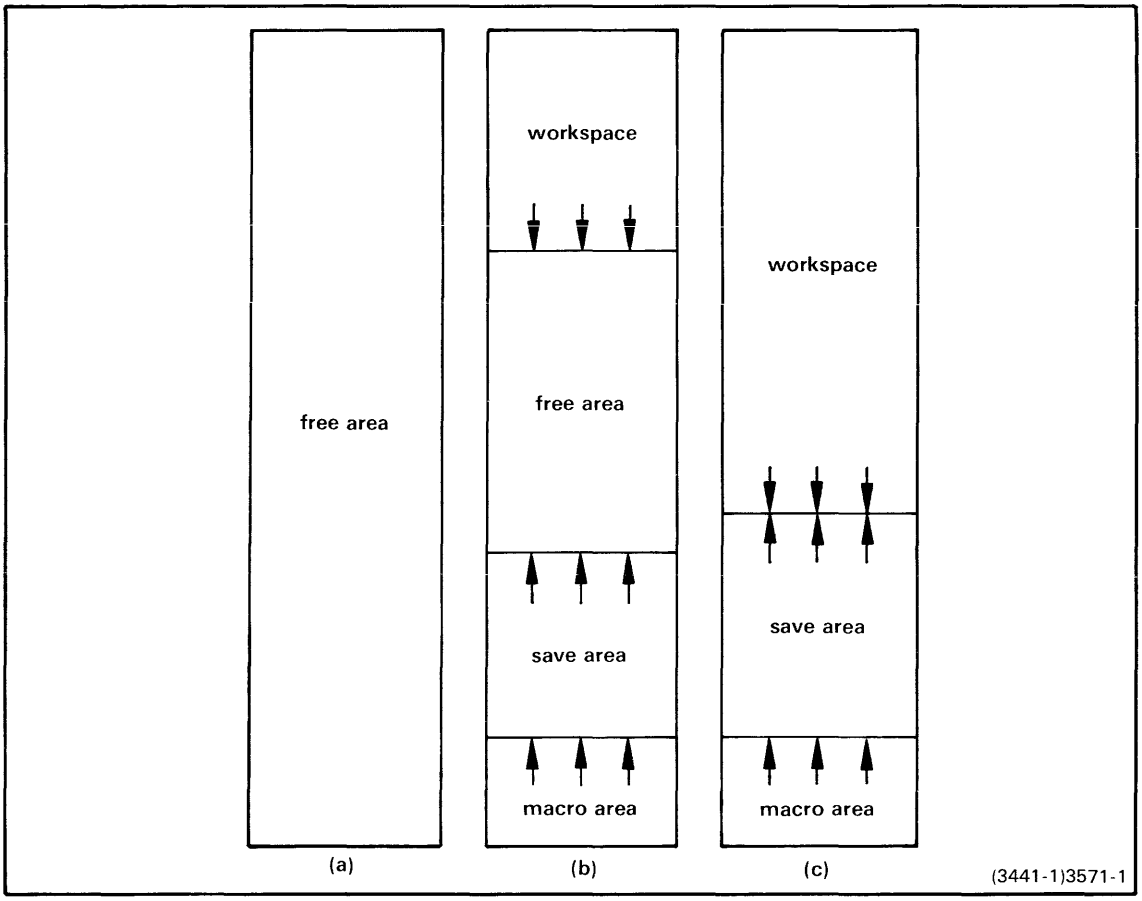


Fig. 3-1. The workspace buffer.

When you invoke the editor, the workspace, save area, and macro area are empty, and all of the workspace buffer is free (Fig. 3-1a). As text is stored into the workspace and the save area, the two areas grow toward each other, and the free area diminishes. The macro area grows with each new macro defined, pushing the save area toward the workspace (Fig. 3-1b). When no free area remains (Fig. 3-1c) the editor responds WORKSPACE FULL. You may recover some free space by moving or deleting lines from the workspace, by deleting macros, or by replacing the contents of the save area with a smaller block of text.

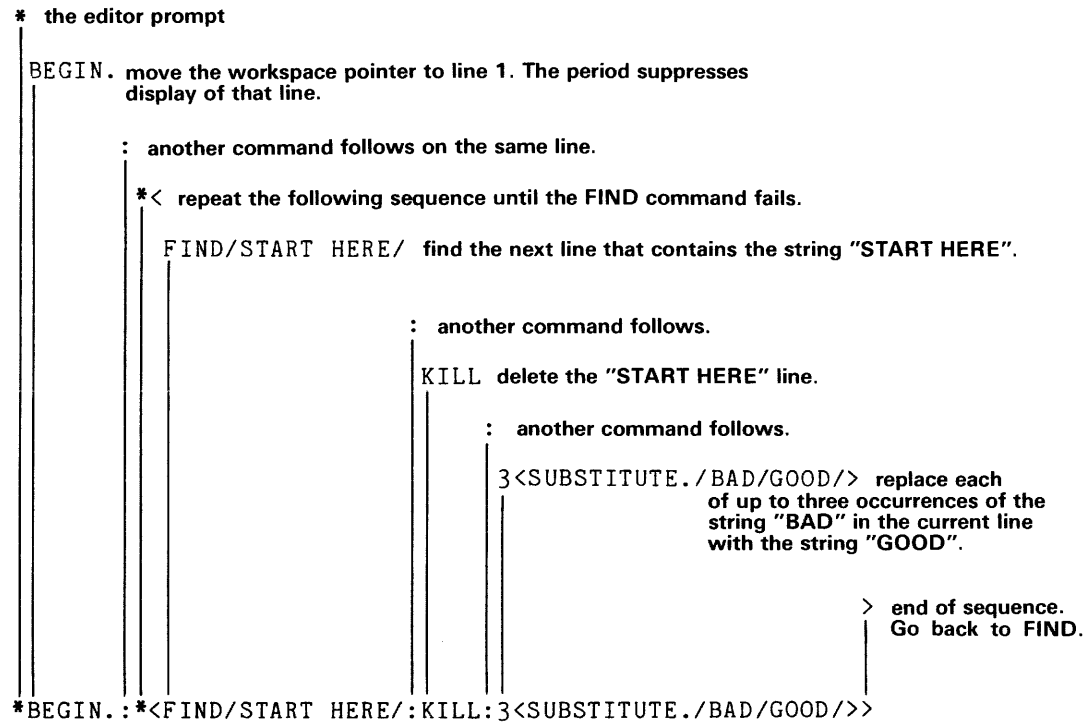
Command Lines

Several characters have special meanings in editor command lines:

- The asterisk ("*") is the editor prompt character.
- The colon (":") separates commands in the same line.
- The period (".") immediately following a command name suppresses display of the new current line after the command executes. (See the BRIEF command.)
- The angle brackets ("<" and ">") enclose a command sequence that is to be repeated a specified number of times (or until a command in the sequence fails to execute). The "<" must be preceded by an integer that indicates the number of times the sequence is to be repeated; or by an asterisk, which says to repeat until a command fails.

Execution of a command line aborts if the editor detects an error.

The following example demonstrates how these special characters are used.



Special Keys

ESC (Escape)—Erases Line; Halts Command Execution.
Pressing the ESC key once deletes the contents of the line being entered. If the editor is awaiting a command line, another prompt is supplied.

If the editor is executing a command when you press ESC, the editor finishes the command being executed but does not continue to the next command in the command line. The editor displays the message "BREAK" and prompts for a command.

If any of the following commands is executing when ESC is pressed, the editor finishes transferring only the line being copied.

COPY FILE FNEXT GET NEXT PUT PUTK

If you want to exit temporarily to DOS/50, use the SUSPEND command. Pressing the ESC key twice rapidly is a less reliable way of accomplishing the same thing.

BACKSPACE or RUBOUT—Deletes Character.
The BACKSPACE key and the RUBOUT key have similar functions; either key tells the editor to discard the last character typed. The RUBOUT key is discussed more fully under the CRT command.

CRTL-S—Halts Display Output

CTRL-S (press the S key while holding down the CTRL key) halts the output to the terminal so you can view the display.

CTRL-Q—Resumes Display Output

CTRL-Q resumes the output to the terminal.

CTRL-R—View the Type-ahead Buffer

CTRL-R displays the contents of the type-ahead buffer on the terminal.

CTRL-U—Erase the Type-ahead Buffer

CTRL-U erases the contents of the type-ahead buffer.

TAB or CTRL—Skips to Next Tab Stop.

The TAB key sends the standard tab character (CTRL-I, ASCII code 09) to the editor. The cursor moves to the next tab stop. If the last tab stop has been passed, the cursor advances one space.

SYNTAX

AGAIN

EXPLANATION

The AGAIN command repeats execution of the latest repeatable editor command, regardless whether that command was typed in, contained in a macro, or input from a command file, and regardless whether the command was successfully executed.

The following commands are repeatable with the AGAIN command:

BEGIN	FNEXT	KILL	PUT	SUBSTITUTE
COPY	FSUBSTITUTE	LIST	PUTK	SUSPEND
DOWN	GET	LN	REPLACE	TYPE
END	INPUT	NEXT	STATUS	UP
FIND	INSERT			

EXAMPLES

Assume that the following text is in the workspace:

```
-->A LINE
   ANOTHER LINE
   YET ANOTHER
   WILL THE LAST LINE EVER COME?
   YES, HERE IT IS AT LAST:
   *** THE LAST LINE ***
```

Enter the following command line:

```
*DOWN.1:KILL 2
```

The editor deletes the second and third lines of text.

The workspace now looks like this:

```
   A LINE
-->WILL THE LAST LINE EVER COME?
   YES, HERE IT IS AT LAST:
   *** THE LAST LINE ***
```

Enter the AGAIN command:

```
*AGAIN
```

The editor repeats only the KILL 2 command, not the whole command line.

The workspace now looks like this:

```
   A LINE
-->*** THE LAST LINE ***
```

SYNTAX

BEGIN

EXPLANATION

The BEGIN command moves the workspace pointer to the first line of text and displays the line. To suppress the display, enter a period after the command.

EXAMPLES

Assume that the following text is in the workspace:

```
      A LINE
      ANOTHER LINE
----> THE NEXT LINE IS THE
      END
```

Enter the following command. The new current line is displayed.

```
*BEGIN
A LINE
```

The workspace now looks like this:

```
----> A LINE
      ANOTHER LINE
      THE NEXT LINE IS THE
      END
```

SYNTAX

BRIEF [ON / OFF]

PARAMETERS

ON	suppresses the current-line display.
OFF	reinstates the current-line display.

EXPLANATION

The BRIEF command sets the value of the BRIEF flag. Normally (BRIEF flag OFF), after each of the following editor commands has finished executing, the new current line is displayed.

BEGIN	END	FNEXT	NEXT	SUBSTITUTE
DOWN	FIND	FSUBSTITUTE	REPLACE	UP

Turning the BRIEF flag ON causes this current-line display to be suppressed. Turning the flag OFF reinstates the display.

Entering the BRIEF command without a parameter reverses the value of the BRIEF flag.

When any of the above commands is followed immediately by a period, the action of the BRIEF flag is reversed for the execution of that command only.

EXAMPLES

Assume that the following text is in the workspace:

```
——► TRUSTWORTHY
   LOYAL
   HELPFUL
   FRIENDLY
   COURTEOUS
```

Enter the following command line:

```
*DOWN:BRIEF ON:DOWN:BRIEF OFF:DOWN:DOWN.
```

The first DOWN moves the workspace pointer to the second line and displays it:

```
LOYAL
```

The second DOWN moves the pointer to the third line, but does not display it because the BRIEF flag has been turned ON. The third DOWN moves the pointer to the fourth line and displays it because BRIEF is OFF again:

```
FRIENDLY
```

The last DOWN moves the pointer to the fifth line, but the current-line display is suppressed by the period following DOWN. This period turns the BRIEF flag back ON for that command only.

SYNTAX

COMMENT [text]

PARAMETERS

text	any line of text

EXPLANATION

The word COMMENT identifies a comment in command input. The rest of the line following a COMMENT command is ignored by the editor. The editor displays the COMMENT line on the system terminal (if the ECHO flag is ON) and proceeds to the next command.

EXAMPLES

Assume that disc file EXAMPLE contains the following text:

```
COM THIS COMMAND FILE MOVES THE POINTER TO THE END OF THE WORKSPACE.
END
COM *** END OF EXAMPLE ***
```

Enter the following command:

```
*PERFORM EXAMPLE
```

The editor executes the command file, moving the pointer to the end of the workspace and displaying each command as it is executed. The editor responds as follows:

```
*COM  THIS COMMAND FILE MOVES THE POINTER TO THE END OF THE WORKSPACE.
*END
 ** EOF
*COM  *** END OF EXAMPLE ***
```

SYNTAX

COPY [**number of lines** | **range of lines**] [{**input filespec**}] [**output filespec**]]

PARAMETERS

number of lines	number of lines to be copied.
range of lines	range of lines to be copied, for example 2-6. B, C, and E may **not** be used as line numbers in a COPY command.
input filespec	the file or device from which text is copied.
output filespec	the file or device to which text is copied. To specify the primary output file, omit this parameter.

EXPLANATION

The COPY command copies text from a file or device to another file or device without affecting the workspace contents or pointer. Copying ends when the specified lines have been copied or when the end of the input file is reached.

If neither a number of lines nor a range of lines is specified, one line is copied.

If an alternate input file is specified, copying begins at the beginning of that file or at the first line in the specified range. If an alternate output file is specified, its contents are replaced with the copied text. If the alternate output file specified did not exist previously, it is created.

If you do not specify an alternate input file or device, or if you specify the primary input file by name, text is copied from the primary input file.

- If you specify a number of lines, copying begins at the primary input file pointer.
- If you specify a range of lines, those lines are copied regardless of the position of the pointer.
- In any case, after copying the primary input file pointer is at the line following the last line copied.

You may not copy **to** the primary input file or **from** the primary output file.

If you specify an output file or device, you must also specify an input file or device.

EXAMPLES

```
*COPY
```

appends one line from the primary input file to the primary output file. The primary input file pointer advances one line.

```
*COPY 10-20 FILE1
```

skips over the first 9 lines of FILE1 and appends lines 10–20 to the primary output file.

```
*COPY 15 FILE2 FILE3
```

replaces the contents of FILE3 with the first 15 lines of FILE2.

```
*COPY 1000 FILE4 CONO
```

displays on the system terminal (CONO) the first 1000 lines of FILE4. If FILE4 contains fewer than 1000 lines, the message ** EOF is displayed after copying ends.

SYNTAX

CRT [ON / OFF]

PARAMETERS

ON	causes RUBOUT to backspace over the deleted character.
OFF	causes RUBOUT to echo the deleted character.

EXPLANATION

The CRT command sets the value of the CRT flag.

If your system terminal is a TV-type device, you will probably want to leave the CRT flag at its initial value (ON). Users of hard-copy terminals may find it easier to keep track of deleted characters if they turn the CRT flag OFF.

The RUBOUT key is used to delete the last character typed from the input text or command line. Normally (CRT flag ON) RUBOUT also erases the deleted character from the screen and backspaces the cursor. Turning the CRT flag OFF causes RUBOUT to echo the deleted character on the terminal instead.

Entering the CRT command without a parameter reverses the value of the CRT flag.

EXAMPLES

Enter the following command line. (Each @ represents a RUBOUT.)

```
*INSERT 123456@@@
```

If you are using a hard-copy terminal, what you see depends on the type of terminal. A TV terminal would display:

```
*INSERT 123
```

Now turn the CRT flag OFF and enter the same command line again:

```
*CRT OFF
*INSERT 123456@@@
```

What you see is different...

```
*INSERT 123456654
```

but the result is the same. Display the two lines entered:

```
*UP.2:TYPE 2
```

Both lines are the same:

```
123
123
```

SYNTAX

DEBUG [ON / OFF]

PARAMETERS

ON	causes the workspace map to be displayed after every command.
OFF	suppresses display of the workspace map.

EXPLANATION

The DEBUG command sets the value of the DEBUG flag.

Normally (DEBUG flag OFF) the workspace map is not displayed. Turning the DEBUG flag ON causes the workspace map to be displayed every time a command is executed. Turning the DEBUG flag OFF suppresses the display.

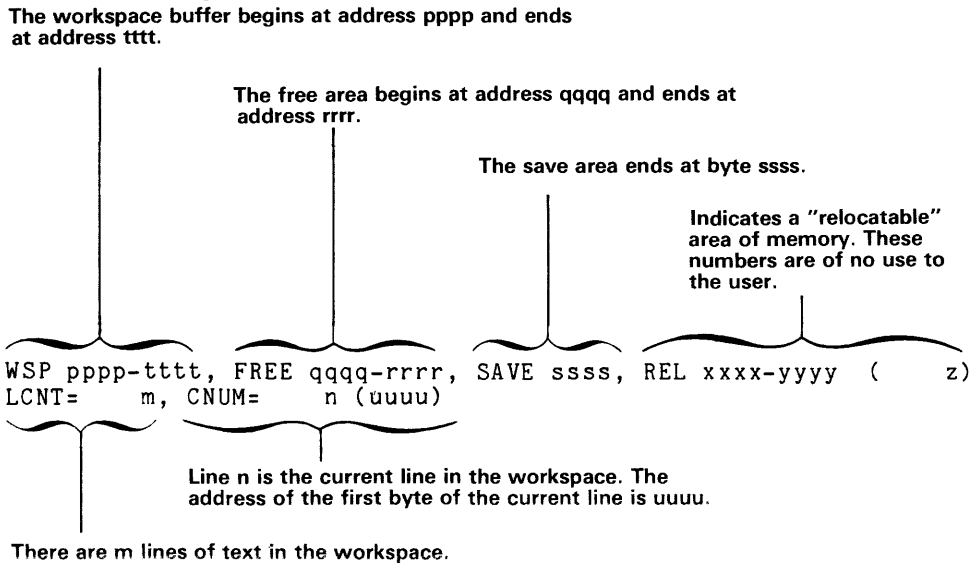
Entering the DEBUG command without a parameter reverses the value of the DEBUG flag.

The workspace map shows the locations of the four areas of the workspace buffer: the workspace, the free area, the save area, and the macro area. You may want to check the map whenever you suspect that the workspace buffer is full or nearly full of text.

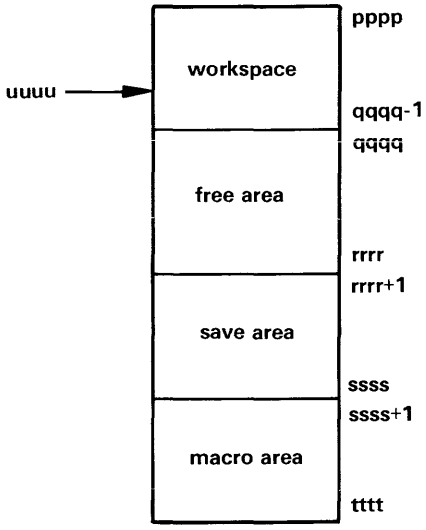
The map has the following format:

```
WSP pppp-tttt, FREE qqqq-rrrr, SAVE ssss, REL xxxx-yyyy  (     z)
LCNT=     m, CNUM=     n (uuuu)
```

You may interpret the workspace map as follows:



The values pppp through uuuu are hexadecimal addresses in program memory; the locations they represent are shown on the diagram below.



EXAMPLES

Assume that the workspace contains the following text:

```
      THE USUAL NONSENSE
      SOME UNUSUAL NONSENSE
----> SOME NONUSUAL UNSENSE
      SOME UNUSABLE INCENSE
```

Enter the following command to turn the DEBUG flag ON:

```
*DEBUG ON
```

Before prompting for another command, the editor checks the DEBUG flag. Because the DEBUG flag is ON, the editor displays the workspace map:

```
WSP 2324-9EB9, FREE 2379-9EB9, SAVE 9EB9, REL 2379-9EB9  (    0)
LCNT=    4, CNUM=    3 (234D)
```

From this map you can determine that the workspace occupies bytes 2324-2378 of program memory. Also, because the free area extends all the way to the end of the workspace buffer, you know that the save and macro areas are empty. There are four lines in the workspace, and the current line, line 3, begins at byte 234D.

Enter the following command to copy the workspace contents into the save area:

```
*SAVE B-E
WSP 2324-9EB9, FREE 2379-9E64, SAVE 9EB9, REL 2379-9EB9  (    0)
LCNT=    4, CNUM=    3 (234D)
```

Now the save area occupies bytes 9E65-9EB9. No other information has changed.

When you define a macro, the command line of the macro is stored in the macro area.

```
*MACRO1=TYPE B-E
WSP 2324-9EB9, FREE 2379-9E5A, SAVE 9EAF, REL 2379-9EB9  (    0)
LCNT=    4, CNUM=    3 (234D)
```

The macro is saved in bytes 9EB0-9EB9, pushing the save area into bytes 9E5B-9EAF. The workspace is unaffected.

Note that as long as the DEBUG flag is ON, the workspace map appears every time a command executes. Enter the following command line:

```
*BEGIN:DOWN
THE USUAL NONSENSE
WSP 2324-9EB9, FREE 2379-9E5A, SAVE 9EAF, REL 2379-9EB9  (    0)
LCNT=    4, CNUM=    1 (2324)
SOME UNUSUAL NONSENSE
WSP 2324-9EB9, FREE 2379-9E5A, SAVE 9EAF, REL 2379-9EB9  (    0)
LCNT=    4, CNUM=    2 (2337)
```

The BEGIN command moves the workspace pointer to the first line, displays the new current line, and displays the workspace map. The DOWN command moves the pointer down to line 2, displays the line, and displays the workspace map.

Enter the following command line to turn the DEBUG flag OFF and display the current line:

```
*DEBUG OFF:TYPE
SOME UNUSUAL NONSENSE
```

SYNTAX

DOWN [number of lines]

PARAMETERS

number of lines number of lines the pointer moves down.

EXPLANATION

The DOWN command moves the workspace pointer the specified number of lines toward the end of the workspace.

If you do not specify a number of lines, the pointer moves down one line.

The new current line is displayed. To suppress the display, enter a period immediately after the command, before any space or number.

If moving the pointer the specified number of lines would place the pointer past the end of the workspace, the pointer stops one line below the last text line and the message ** END OF TEXT is displayed.

EXAMPLES

Assume that the workspace contains the following text:

```
      A LINE
----> A SECOND LINE
      HOW MANY LINES
      DO I HAVE?
      FIVE LINES
```

Enter the following command. The new current line is displayed.

```
*DOWN 2
DO I HAVE?
```

The workspace now looks like this:

```
      A LINE
      A SECOND LINE
      HOW MANY LINES
----> DO I HAVE?
      FIVE LINES
```

SYNTAX

```
        [ON ]
ECHO    [OFF]
```

PARAMETERS

ON	causes command file lines to be echoed on the system terminal.
OFF	suppresses the command-line display.

EXPLANATION

The ECHO command sets the value of the ECHO flag.

When an editor command file is performed, each command line is first displayed on the system terminal and then executed. Turning the ECHO flag OFF causes the command-line display to be suppressed. Turning the flag ON reinstates the display.

Entering the ECHO command without a parameter reverses the value of the ECHO flag.

The ECHO flag is initially ON.

EXAMPLES

Assume that file SAMPLE contains the following text:

```
COM  THIS LINE WILL BE PRINTED.
ECHO OFF
COM  THIS LINE WON'T BE.
ECHO:COM  REVERSE THE ECHO FLAG FROM OFF TO ON.
COM  THIS LINE WILL ALSO BE PRINTED.
```

Execute the command file:

```
*PERFORM SAMPLE
```

The following lines will be displayed on the system terminal:

```
*COM  THIS LINE WILL BE PRINTED.
*ECHO OFF
*COM  THIS LINE WILL ALSO BE PRINTED.
```

SYNTAX

EDIT [infile] [outfile] [comfile]

PARAMETERS

infile	primary input filespec.
output	primary output filespec.
comfile	initial command filespec.

EXPLANATION

The DOS/50 command EDIT invokes the editor and specifies:

- the file that contains the text to be edited (the **primary input file**);
- the file on which the edited text will be saved (the **primary output file**);
- the file that contains a series of editor commands to be executed at the beginning of the editing session (the **initial command file**).

Editing Options

A typical editing session proceeds as follows:

1. Text is copied from the primary input file to the workspace with the commands GET, NEXT, or FNEXT.
2. Text in the workspace is modified with various editor commands.
3. Text is copied from the workspace to the primary output file with the commands FILE, NEXT, or FNEXT.

You may add text to the workspace from sources other than the primary input file. The INPUT command allows you to enter text from the system terminal. The GET command is used to copy text from other files.

You may use the commands PUT or PUTK to save text from the workspace onto files other than the primary output file.

Primary and Alternate Files

Any file, other than the primary input file and the primary output file, may serve as an **alternate input file** (a file that provides text) or an **alternate output file** (a file that receives text).

The primary input file differs from alternate input files in the following ways:

- Copying from an alternate input file always begins at the beginning of the file. Copying from the primary input file generally begins at the primary input file pointer, which points to the line following the last line copied.
- An alternate input file must be specified by its filespec in a GET or COPY command. The primary input file is the default input file for GET and COPY.
- The commands FILE, NEXT, and FNEXT can copy text only from the primary input file.

The primary output file differs from alternate output files in similar ways:

- Text copied to an alternate output file always replaces the previous contents of the file. (If the file did not exist, it is created.) Text copied to the primary output file is generally appended to the end of the file.
- An alternate output file must be specified by its filespec in a PUT, PUTK, or COPY command. The primary output file is the default output file for PUT, PUTK, and COPY.
- The commands FILE, NEXT, and FNEXT can copy text only to the primary output file.

The Initial Command File

An editor command file contains a series of editor commands; these commands are identical to any that you may enter from the system terminal. You may specify an editor command file as the third parameter in the EDIT command line. The editor executes the commands from that file (called the **initial command file**) before accepting commands from the system terminal. If the editor encounters a FILE or QUIT command in the initial command file, the editing session ends.

Useful Invocations

The four useful forms of the EDIT command are described here. Each form may include or omit the optional third parameter. In the following discussion:

- **oldfile** represents any existing filespec;
- **newfile** represents a non-existent filespec;
- **comfile** represents the initial command filespec.

The two most commonly used forms of the EDIT command are **EDIT newfile** and **EDIT oldfile**. Two other invocations you may find useful are **EDIT** (no parameters) and **EDIT oldfile newfile**.

1. **EDIT newfile** or **EDIT newfile,,comfile**

   This invocation begins the editing session that creates file **newfile**. There is no primary input file. You may enter text from the system terminal or add text from alternate input files. The FILE command copies the text in the workspace to the primary output file, saves the primary output file in the file **newfile**, and ends the editing session.

2. **EDIT oldfile** or **EDIT oldfile,,comfile**

   This invocation begins an editing session that modifies file **oldfile**, which becomes the primary input file. A temporary primary output file is created. You may use the commands GET, NEXT, or FNEXT to bring text from **oldfile** into the workspace. You may modify the text in the workspace, bring in additional text from alternate input files, or copy text to alternate output files.

   When you close the editing session with a FILE command, the edited version of **oldfile** is copied to the primary output file. The primary output file is saved in the file **oldfile**, and the primary input file is given a backup name, **oldfile#**. If the file is already called **oldfile#**, the backup file name will be **oldfile##**.

   If you abort the editing session with a QUIT command, the primary output file is lost and the primary input file retains the name [oldfile].

3. **EDIT** or **EDIT,,,comfile**

   You may use this invocation when you have no use for a primary input file or a primary output file (for example, when you are sorting out text among several files). All text must be entered from the system terminal or copied from alternate input files. The text must be saved on alternate output files. You must use the QUIT command to end the editing session.

4. **EDIT oldfile newfile** or **EDIT oldfile newfile comfile**

   This invocation allows you to create a modified version of **oldfile** on **newfile**. **Oldfile** is not affected. **Oldfile** is the primary input file and **newfile** is the primary output file. When you close the editing session with a FILE command, the edited version of **oldfile** is copied to the primary output file, **newfile**. If you abort the editing session with a QUIT command, **newfile** is lost.

Avoid any invocations other than these four.

Invocations to Avoid

If you specify an existing file as the primary output file, the editor responds ** SUPERSEDING EXISTING FILE. The previous contents of the file are lost unless you exit with a QUIT command before copying any text to the primary output file.

If you specify a non-existent file as the primary input file, the editor responds

```
** CANNOT READ NEW FILE
** PRIMARY INPUT
```

and designates no primary input file.

If you specify the same file as both the primary input file and the primary output file:

- If the file exists, the invocation is equivalent to **EDIT oldfile**.
- If the file does not exist, the editor responds

  ```
  ** CANNOT READ NEW FILE
  ** PRIMARY INPUT
  ** SUPERSEDING EXISTING FILE
  ```

  and designates neither a primary input file nor a primary output file.

EXAMPLES

```
> EDIT PROG
```

begins the editing session that creates or modifies file PROG in the current directory. If PROG already exists, it serves as the primary input file. When you close the editing session with a FILE command, the edited version (the primary output file) takes the name PROG and the old version (the primary input file) takes the name PROG#. If PROG did not exist previously, there is no primary input file. When you enter the FILE command, the primary output file is saved under the name PROG.

```
> EDIT DIRTY CLEAN
```

begins the editing session that modifies text from file DIRTY and stores the text in file CLEAN. DIRTY is the primary input file and CLEAN is the primary output file. If CLEAN did not previously exist, it is created. If CLEAN already exists, the editor responds **SUPERSEDING EXISTING FILE, and the current contents of CLEAN are lost unless you exit with a QUIT command before copying any text to the primary output file.

SYNTAX

END

EXPLANATION

The END command moves the workspace pointer one line beyond the last line of text and displays the message ** END OF TEXT. To suppress the display, enter a period after the command.

EXAMPLES

Assume that the workspace contains the following text:

```
      ONE
----> TWO
      I THINK
      I'M THROUGH
```

Enter the following command to move the pointer one line past the last line of text. The editor displays the message ** END OF TEXT.

```
*END
** END OF TEXT
```

The workspace now looks like this:

```
      ONE
      TWO
      I THINK
      I'M THROUGH
---->
```

SYNTAX

ERROR [ON / OFF]

PARAMETERS

ON	reinstates display of the error pointer.
OFF	suppresses display of the error pointer.

EXPLANATION

The ERROR command sets the value of the ERROR flag.

Whenever an error is found during the interpretation or execution of a command, the editor displays an appropriate error message. Normally (ERROR flag ON) the command line is also displayed, with a pointer (^ or ↑) below the last character of the illegal command or parameter. When the ERROR flag is turned OFF, display of the command line and pointer is suppressed. Turning the flag ON reinstates the display.

Entering the ERROR command without a parameter reverses the value of the ERROR flag.

The ERROR flag is initially ON.

EXAMPLES

Enter the following command line:

```
*HELP ME!
```

The editor displays an error message, lists the command line, and points to the the last character of the illegal command HELP.

```
** UNKNOWN COMMAND
** HELP ME!
**     ^
```

Turn the ERROR flag OFF and enter the illegal command again:

```
*ERROR OFF
*HELP ME!
```

Only the error message is displayed.

```
** UNKNOWN COMMAND
```

SYNTAX

FILE

EXPLANATION

The FILE command ends a normal editing session, causing the following to happen:

1. The contents of the workspace are appended to the primary output file.
2. Any remaining text in the primary input file is appended to the primary output file.
3. If the name of the primary output file was not specified in the EDIT command line or was the same as the name of the primary input file:
   a. The primary output file (to which the editor has given the temporary name ###.EDIT.TMP) is given the name of the primary input file.
   b. The primary input file is renamed to a backup file name, replacing the existing backup file, if any. The backup file name is formed by appending "#" to the name of the primary input file.

If neither a primary input file nor a primary output file was specified in the EDIT command line, then you must use the QUIT command to exit from the editor. If you were to enter a FILE command, the editor would respond ** NO OUTPUT FILE SPECIFIED and do nothing. The editor thus guards against accidental loss of the workspace contents.

EXAMPLES

Use the editor to modify system file PROG:

```
> EDIT PROG
 ** EDIT VERSION 3.0
*
   .
   .       Edit the contents of PROG.
   .
*FILE
 ** END OF TEXT
 ** EOF

>
```

Two versions of PROG now exist: unmodified (PROG#) and modified (PROG).

SYNTAX

FIND {/**string**/}

PARAMETERS

string delimiter: any character except a space. The delimiter character must not occur in the string. Both delimiters must be the same character. If the delimiter character is a letter or period, a space must separate the command from the first delimiter. The final delimiter may be omitted if it is the last character in the command line.

string — the string of characters to be found.

EXPLANATION

The FIND command searches for the specified string, beginning at the current line and proceeding toward the end of the workspace.

If the string is found, the workspace pointer moves to the line containing the string and the line is displayed. To suppress the display, enter a period between the command and the first delimiter.

If the string is not found, the workspace pointer does not move and the message ** NOT FOUND is displayed.

If you enter the AGAIN command to repeat a FIND command, the search begins one line down from the current line, so that the same line is not found again.

If a FIND command inside repeat brackets ("<" and ">") fails to find a match, execution resumes with the next command outside the brackets.

If you are using tab stops, remember that while each tab character in your text is displayed as one or more spaces, it is stored as a single character. To include a standard tab character in your search string, press the TAB key at the appropriate place in the command line.

See the XSEARCH command for an explanation of special search characters.

EXAMPLES

Assume that the workspace contains the following text:

```
——► I THINK
   I SEE
   A LINE
   FOR ME
```

Enter the following command:

```
*FIND/A/
```

The workspace pointer moves to line 3. That line is displayed.

```
A LINE
```

The workspace now looks like this:

```
   I THINK
   I SEE
——► A LINE
   FOR ME
```

The FIND command searches the text beginning at the current line. The search will fail if the search string does not occur in or after the current line.

```
*FIND/THINK/
** NOT FOUND
```

SYNTAX

FNEXT {/**string**/}

PARAMETERS

/	string delimiter: any character except a space. The delimiter character must not occur in the string. Both delimiters must be the same character. If the delimiter character is a letter or period, a space must separate the command from the first delimiter. The final delimiter may be omitted if it is the last character in the command line.
string	the string of characters to be found.

EXPLANATION

The FNEXT command searches from the current line through the workspace, then through the rest of the primary input file, until the string is found. When no match is found in the current contents of the workspace, they are appended to the primary output file and the workspace is filled to three-fourths of its capacity with text from the primary input file. The search then continues from the first line of the new workspace contents.

When a match is found, the line containing the specified string becomes the current line. That line may occur anywhere in the workspace.

If no match is found, the command is terminated. At that time, the workspace is empty and the primary input file is at its end.

If you enter the AGAIN command to repeat the FNEXT command, the search begins one line after the current line, so that the same line is not found again.

If an FNEXT command inside repeat brackets ("<" and ">") fails to find a match, execution resumes with the next command outside the brackets.

If you are using tab stops, remember that while each tab character in your text is displayed as one or more spaces, it is stored as a single character. To include a standard tab character in your search string, press the TAB key at the appropriate place in the command line.

See the XSEARCH command for an explanation of special search characters.

EXAMPLES

If the search string occurs in the workspace at or after the current line, FNEXT acts the same as FIND. Assume that the following text is in the workspace:

```
--> NOT ME
    NOPE
    HERE I AM
```

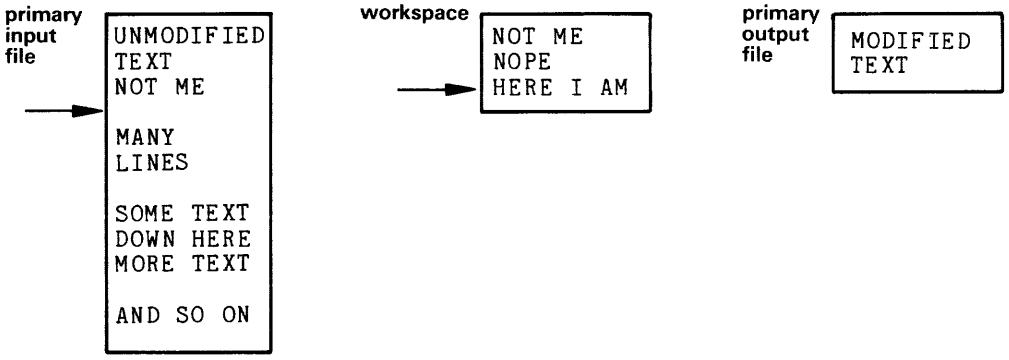
Enter the following command:

```
*FNEXT/HERE/
```

The line containing the string "HERE" becomes the current line.

```
    NOT ME
    NOPE
--> HERE I AM
```

Now consider a case in which the text you want to edit is not in the workspace. Assume that the next line to be edited contains the string "DOWN HERE" and occurs toward the end of the primary input file. The diagram below shows the contents of the workspace and the primary files. Note the positions of the primary input file pointer and the workspace pointer.

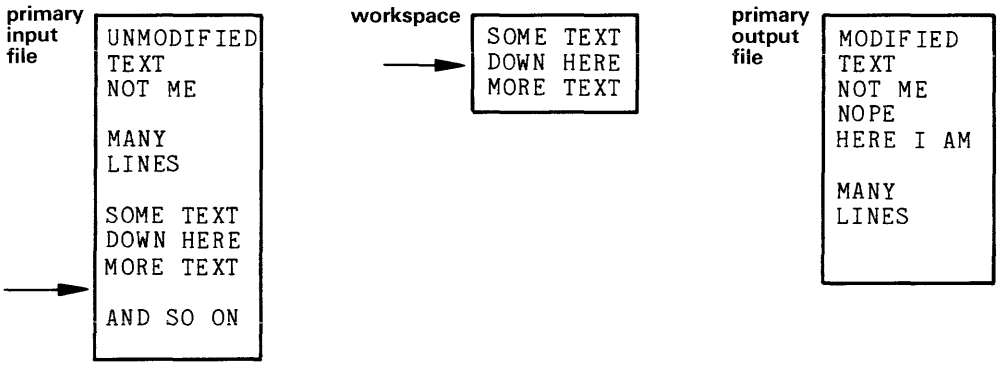


Enter the following command:

```
*FNEXT/DOWN HERE/
```

When the editor fails to find the string in the current workspace, it appends the workspace contents to the primary output file. Then it copies sections of text from the primary input file, through the workspace, to the primary output file, until the section that contains the string is brought into the workspace.

After execution of the command, this is what you have:



SYNTAX

FSUBSTITUTE {/**search string**/**new string**/}

PARAMETERS

string delimiter: any character except a space. The delimiter character must not occur in either string. All three delimiters must be the same character. If the delimiter character is a letter or period, a space must separate the command from the first delimiter. The final delimiter may be omitted if it is the last character in the command line.

search string — the string of characters you want to replace.

new string — the string of characters that replaces the search string.

EXPLANATION

The FSUBSTITUTE command combines the functions of the commands FIND and SUBSTITUTE. Beginning at the current line, the editor searches the workspace for the search string. If the search string is found, the line that contains it becomes the current line and the new string replaces the search string. If no match is found, the message ** NOT FOUND is displayed and the workspace pointer remains at the same line.

The strings may be of different lengths. If the new string is empty, the search string is deleted.

After the substitution, the modified line is displayed. To suppress the display, enter a period between the command and the first delimiter.

If the modified line exceeds 127 characters, the line is truncated to 127 characters and the editor displays the message ** TRUNCATED along with the truncated line.

If an FSUBSTITUTE command inside repeat brackets ("<" and ">") fails to find a match, command execution resumes with the next command outside the brackets.

If you are using tab stops, remember that while each tab character in your text is displayed as one or more spaces, it is stored as a single character. To include a standard tab character in your search string or new string, press the TAB key at the appropriate place in the command line.

See the XSEARCH command for an explanation of special search characters.

EXAMPLES

The following text is in the workspace:

```
→ yes yes yes
  yes yes
  NO NO NO
  yes yes
  NO
```

Enter the following command:

```
*FS/NO/yes/
```

The first occurrence of the string "NO" is on line 3. That line becomes the current line and is displayed as modified:

```
yes NO NO
```

You may enclose an FSUBSTITUTE command in repeat brackets to make the same text substitution any number of times; however, be sure that the new string does not contain the search string. For example, the command line

```
5<FS/MAN/WOMAN/>
```

replaces the first occurrence of "MAN" with the string "WOWOWOWOWOMAN".

SYNTAX

GET [**number of lines** | **range of lines**] **[input filespec]**

PARAMETERS

number of lines	number of lines to be read.
range of lines	range of lines to be read, for example 2-6. B, C, and E may **not** be used as line numbers in a GET command.
input filespec	file or device from which text is copied. To specify the primary input file, omit this parameter.

EXPLANATION

The GET command reads text from a file into the editor workspace. The text is inserted before the current line. The workspace pointer does not move.

Copying ends when the specified lines have been copied or when the end of the input file is reached.

If neither a number of lines nor a range of lines is specified, one line is copied.

If an input file is specified, copying begins at the beginning of the file or at the first line in the specified range. You may **not** specify the primary input file by name, and you may not specify the primary output file at all.

If an input file is not specified, text is copied from the primary input file.

- If you specify a number of lines, copying begins at the primary input file pointer.
- If you specify a range of lines, those lines are copied, regardless of the position of the primary input file pointer.
- Afterward, in either case, the primary input file pointer is at the line following the last line copied.

EXAMPLES

```
*GET 30 FILE1
```

copies the first 30 lines of file FILE1 into the workspace, inserting them before the current line.

```
*GET 10-20
```

copies lines 10-20 from the primary input file into the workspace, inserting them before the current line. Afterward, the primary input file pointer is at line 21.

```
*GET
```

inserts the next line from the primary input file into the workspace. The primary input file pointer advances one line.

SYNTAX

INPUT

EXPLANATION

The INPUT command allows you to type any number of text lines into the workspace. INPUT places the editor in input mode. All text entered in input mode, up to (but excluding) the first empty line, is inserted into the workspace before the current line.

An empty line (carriage return only) terminates input mode, but a blank line (one or more spaces followed by a carriage return) is treated like any other text line.

The current line remains the same.

There is no prompt character in input mode.

If a text line exceeds 127 characters, the editor displays the message ** TRUNCATED. The line you entered is replaced with the text line ** TRUNCATED.

If an INPUT command is executed from a command file, the editor accepts text lines from the command file rather than from the keyboard. The command file must contain an empty line at the end of the input text. Although you cannot use INPUT or INSERT to create an empty line, you may use the following command line to insert an empty line after the current line:

```
*XSEARCH ON:FS/>/>>/
```

EXAMPLES

To create text in an empty workspace, enter:

```
*INPUT
```

The editor responds:

```
INPUT:
```

Enter the following lines:

```
DICE 3 POTATOES INTO SAUCEPAN.
ADD 1 CUP CHOCOLATE SYRUP.
CHILL 15 MIN. BEFORE SERVING.
```


@

After terminating the third line with a carriage return, enter a second carriage return to terminate input mode.

The workspace now looks like this:

```
        DICE 3 POTATOES INTO SAUCEPAN.
        ADD 1 CUP CHOCOLATE SYRUP.
        CHILL 15 MIN. BEFORE SERVING.
------>
```

To insert text between lines 2 and 3, move the workspace pointer to line 3:

```
*UP
```

The new current line is displayed.

```
CHILL 15 MIN. BEFORE SERVING.
```

Add the following text:

```
*INPUT
INPUT:
STIR BRISKLY 30 SEC.
DRAIN OFF EXCESS SYRUP.
```

The workspace now looks like this:

```
        DICE 3 POTATOES INTO SAUCEPAN.
        ADD 1 CUP CHOCOLATE SYRUP.
        STIR BRISKLY 30 SEC.
        DRAIN OFF EXCESS SYRUP.
------> CHILL 15 MIN. BEFORE SERVING.
```

SYNTAX

INSERT [text]

PARAMETERS

text	any line of text. An INSERT command without text is interpreted as an INPUT command.

EXPLANATION

The INSERT command inserts one line of text into the workspace immediately preceding the current line.

One or more spaces must separate the INSERT command from the text. The text may not begin with a space.

If the command line, including the text, exceeds 127 characters, the editor displays the message ** COMMAND INPUT ABORTED and the command line is lost.

EXAMPLES

Assume that the workspace contains the following text:

```
      THERE WAS A LITTLE HEN
      AND SHE HAD A WOODEN LEG
----> SHE COULD LAY MORE WOODEN EGGS THAN ANY HEN ON THE FARM
      ANOTHER LITTLE DRINK WON'T DO US ANY HARM
```

Enter the following command:

```
*INSERT THE BEST LITTLE HEN THAT EVER LAID A WOODEN EGG
```

The workspace now looks like this:

```
      THERE WAS A LITTLE HEN
      AND SHE HAD A WOODEN LEG
      THE BEST LITTLE HEN THAT EVER LAID A WOODEN EGG
----> SHE COULD LAY MORE WOODEN EGGS THAN ANY HEN ON THE FARM
      ANOTHER LITTLE DRINK WON'T DO US ANY HARM
```

SYNTAX

```
KILL  [number of lines]
      [range of lines ]
```

PARAMETERS

number of lines — number of lines to be deleted, beginning with the current line.

range of lines — range of lines to be written, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.

EXPLANATION

The KILL command deletes the indicated lines of text from the workspace.

If you do not specify a number of lines or a range of lines, only the current line is deleted.

The workspace pointer does not move unless the current line is deleted, in which case the first line after the deleted text becomes the current line.

EXAMPLES

```
*KILL
```

deletes the current line.

```
*KILL 10
```

deletes 10 lines beginning with the current line.

```
*KILL 20-C
```

deletes all text between line 20 and the current line, inclusive.

SYNTAX

```
LIST [number of lines]
     [range of lines ]
```

PARAMETERS

number of lines	number of lines to be listed, beginning with the current line.
range of lines	range of lines to be saved, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.

EXPLANATION

The LIST command lists the specified lines of workspace text on the line printer.

The workspace contents and pointer are not affected.

If neither a number of lines nor a range of lines is specified, only the current line is listed.

EXAMPLES

```
*LIST
```

lists the current line.

```
*LIST 10
```

lists 10 lines, beginning with the current line.

```
*LIST 5-15
```

lists workspace lines 5 through 15.

SYNTAX

LN

EXPLANATION

The LN command displays the line number of the current line and the number of lines in the workspace.

EXAMPLES

Assume that the workspace contains the following text:

```
        AS I WAS STANDING IN THE STREET,
        AS QUIET AS CAN BE,
------> A GREAT BIG UGLY MAN CAME UP,
        AND TIED HIS HORSE TO ME.
```

Enter the following command:

```
*LN
```

The editor reports the current line number and line count.

```
00003  (00004)
```

SYNTAX

MACRO [number [=[command line]]]

PARAMETERS

number	macro identification number: any integer from 1 to 127.
command line	the command line represented by this macro.

EXPLANATION

The MACRO command stores command lines that will be used repeatedly. MACRO is also used to execute those command lines, to list any presently defined macros, and to delete macros.

To define a macro, enter MACRO, the number, the equals sign, and the command line. Any macro definition previously associated with that number is lost. No spaces are necessary except those that are required by the commands in the command line.

To delete a macro, enter MACRO, the identifying number, and the equals sign.

To execute a macro, enter MACRO and the identifying number.

To list the macros you have defined, enter MACRO with no parameters.

A macro may invoke other macros, but not itself. A macro may not contain a macro definition.

A macro may not delete a macro or contain a PERFORM command.

EXAMPLES

Enter the following command to define a macro that displays 10 lines and then advances the pointer 10 lines:

```
*MACRO5=TYPE10:DOWN.10
```

If the workspace pointer is on line 14 and you enter MACRO5, lines 14 to 23 will be displayed. Since the TYPE command does not move the pointer, the DOWN command is used to move the pointer down 10 lines to the new current line, number 24. The current line is not displayed, since the period after the DOWN command suppresses the display.

To execute the macro, enter:

```
*MACRO5
```

To display all the macros you have defined, enter:

```
*MACRO
```

To delete macro 5, enter:

```
*MACRO5=
```

SYNTAX

MOVE { **number of lines** / **range of lines** } **[line number]**

PARAMETERS

number of lines	number of lines to be moved, beginning with the current line.
range of lines	range of lines to be deleted, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.
line number	line number of the destination line. B, C, or E may also be used if the text is to be inserted before the first line, current line, or end of workspace. E is the default destination line.

EXPLANATION

The MOVE command deletes the specified lines from their current position in the workspace and inserts them before the destination line.

If the destination line is not specified or does not exist, the lines are moved to the end of the workspace.

The workspace pointer remains pointing to the same line, even if that line moves.

EXAMPLES

Assume that the workspace contains the following text:

```
      DO
      RE
      SO
      LA
--->  TI
      MI
      FA
      DO
```

Enter the following command:

```
*MOVE 3-5 8
```

Lines 3, 4, and 5 are inserted before line 8. The rearranged workspace is shown below. Notice that "TI" is still the current line, even though its line number has changed from 5 to 7.

```
      DO
      RE
      MI
      FA
      SO
      LA
--->  TI
      DO
```

SYNTAX

NEXT [number of lines]

PARAMETERS

number of lines number of new lines to be brought into the workspace.

EXPLANATION

The NEXT command appends the workspace contents (if any) to the primary output file, clears the workspace, and copies the specified number of lines from the primary input file into the workspace.

If a number of lines is not specified, the workspace is filled to three-fourths of its capacity with text from the primary input file.

After the new lines have been brought into the workspace, the first line in the workspace becomes the current line and is displayed.

Any of the following messages may be displayed during normal execution of a NEXT command:

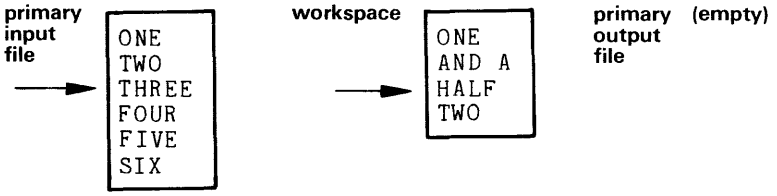
END OF TEXT	The workspace contents have been appended to the primary output file.
WORKSPACE FULL	Text from the primary input file has filled the workspace to full capacity, if a number of lines was specified, or to three-fourths of its capacity otherwise.
EOF	The primary input file has been read to its end.

EXAMPLES

The following two command lines are equivalent:

```
*NEXT 50
*PUTK B-E:GET 50:BEGIN
```

Consider the following situation. (Note the position of the primary input file pointer.)



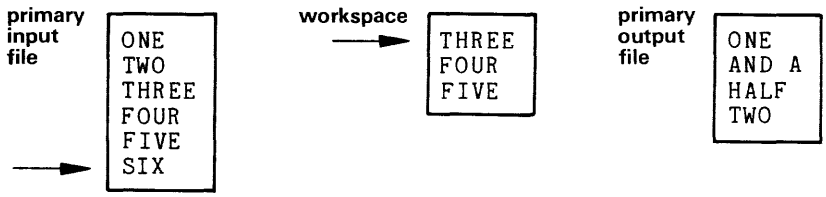
Enter the command:

```
*NEXT 3
```

The workspace contents are moved to the primary output file; then the next three lines from the primary input file are brought into the workspace. The editor reports when the workspace has been cleared, then displays the new current line when copying is finished.

```
** END OF TEXT
THREE
```

After execution of the command, the files and workspace look like this:



SYNTAX

NUMBER [ON / OFF]

PARAMETERS

ON	causes line numbers to accompany the displayed text.
OFF	suppresses the display of line numbers.

EXPLANATION

The NUMBER command sets the value of the NUMBER flag.

The editor can display workspace text with or without line numbers. Normally (NUMBER flag OFF) line numbers are not displayed. Turning the NUMBER flag ON causes each line displayed to be accompanied by the number indicating that line's position in the workspace. Turning the NUMBER flag OFF suppresses the display of line numbers.

Entering the NUMBER command without a parameter reverses the value of the NUMBER flag.

EXAMPLES

Assume that the following text is in the workspace:

```
      WHEN
      IN
      DOUBT,
----► CLEAR
      THE
      ACCUMULATOR.
```

Enter the following command:

```
*TYPE 3
```

The editor displays three lines, beginning with the current line.

```
CLEAR
THE
ACCUMULATOR.
```

Now turn on line numbering and display the same three lines:

```
*NUMBER ON
*TYPE 3
```

Line numbers are displayed.

```
4: CLEAR
5: THE
6: ACCUMULATOR.
```

SYNTAX

PERFORM {**command filespec**}

PARAMETERS

command filespec — a file containing editor commands.

EXPLANATION

The PERFORM command causes the editor to begin executing commands from the specified file. When the end of the command file is reached or an error is detected, the editor prompts for a command from the system terminal. The editing session will terminate if a FILE or QUIT command is executed from the command file.

All the usual editor responses (prompt character, current-line display, text display, informative messages, error messages) are displayed on the system terminal. In addition, if the ECHO flag is ON, each command line is displayed before it is executed.

Commands following PERFORM on the same command line will not be executed.

A command file may contain any editor command except PERFORM.

The initial command file (specified by the third parameter of the EDIT command) is treated as if it has been started by a PERFORM command.

EXAMPLES

Assume that the BRIEF flag is OFF, the ECHO flag is ON, and the following text is in the workspace:

```
      I DO NOT UNDERSTAND COMMAND FILES.
      I DO NOT LIKE COMMAND FILES.
----> I DO NOT USE COMMAND FILES.
```

Disc file PEPTALK contains the following text:

```
BEGIN.:*<FS/DO NOT //>
END.:INSERT I'M OK; COMMAND FILES ARE OK.
```

Execute the command file:

```
*PERFORM PEPTALK
```

The first line of the command file moves the workspace pointer to the first line, then deletes all occurrences of the string "DO NOT ". The second command line inserts a line at the end of the workspace. The editor responds as follows as it executes the command file:

```
*BEGIN.:*<FS/DO NOT //>          first command line echoed
I UNDERSTAND COMMAND FILES.      revised current line
I LIKE COMMAND FILES.            new current line
I USE COMMAND FILES.             new current line
 ** NOT FOUND                    no more occurrences of the string
*END.:INSERT I'M OK; COMMAND FILES ARE OK.
                                 second command line echoed
*                                ready for a command from the keyboard
```

The workspace now looks like this:

```
      I UNDERSTAND COMMAND FILES.
      I LIKE COMMAND FILES.
      I USE COMMAND FILES.
      I'M OK; COMMAND FILES ARE OK.
```

SYNTAX

```
PUT [number of lines] [output filespec]
    [range of lines]
```

PARAMETERS

number of lines	number of lines to be written, beginning with the current line.
range of lines	range of lines to be listed, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.
output filespec	the file or device to which text is written. To specify the primary output file, omit this parameter.

EXPLANATION

The PUT command writes the specified lines of workspace text onto the specified file or device. The workspace contents and pointer are not affected.

If neither a number of lines nor a range of lines is given, only the current line is copied.

If no output file or device is specified, text is appended to the primary output file.

If an alternate output file is specified, its contents are replaced by the copied text. If the specified file did not previously exist, it is created. You may **not** specify the primary output file by name, and you may not specify the primary input file at all.

EXAMPLES

```
*PUT
```

appends a copy of the current line to the primary output file.

```
*PUT 1-20 FILEA
```

replaces the contents of FILEA with a copy of the first 20 lines in the workspace.

```
*PUT 30 LPT
```

lists 30 lines on the line printer, beginning with the current line.

SYNTAX

```
        [number of lines]
PUTK    [range of lines ]  [output filespec]
```

PARAMETERS

number of lines	number of lines to be written, beginning with the current line.
range of lines	range of lines to be moved, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.
output filespec	the file or device to which text is written. To specify the primary output file, omit this parameter.

EXPLANATION

PUTK combines the functions of the commands PUT and KILL.

The PUTK command writes the specified lines of workspace text onto the specified file or device, then deletes those lines from the workspace.

The workspace pointer does not move unless the current line is deleted, in which case the first line after the deleted text becomes the current line.

If neither a number of lines nor a range of lines is given, only the current line is copied and deleted.

If no output file or device is specified, text is appended to the primary output file.

If an alternate output file is specified, its contents are replaced by the copied text. If the specified file did not previously exist, it is created. You may **not** specify the primary output file by name, and you may not specify the primary input file at all.

EXAMPLES

```
*PUTK
```
appends the current line to the primary output file and deletes the line from the workspace. The next line becomes the current line.

```
*PUTK 1-20 FILEA
```
replaces the contents of FILEA with the first 20 lines in the workspace and deletes those lines from the workspace. If the current line is among those deleted, the new first line (formerly line 21) becomes the current line.

```
*PUTK 30 LPT
```
lists 30 lines on the line printer, beginning with the current line, and deletes those lines from the workspace. The first line after the deleted text becomes the current line.

SYNTAX

QUIT

EXPLANATION

The QUIT command aborts the editing session.

If the primary output file is a new file or a new version of the primary input file, the primary output file is deleted.

If the primary output file existed before the editing session began:

- If no text has been written to it in the current editing session (through commands such as COPY, FNEXT, NEXT, PUT and PUTK), it retains its original contents.
- Otherwise it contains only the text written to it during this editing session.

All changes made to alternate output files during this session are retained.

The primary input file remains unchanged, and its backup file (if any) is not deleted.

SYNTAX

REPLACE {**text**}

PARAMETERS

text	any line of text.

EXPLANATION

The REPLACE command replaces the whole current line with the new text line specified. The workspace pointer does not move.

If the command line, including the text, exceeds 127 characters, the editor displays the message ** COMMAND INPUT ABORTED and the command line is lost.

One or more spaces must separate the command from the text. The text may not begin with a space.

The new line is displayed. To suppress the display, enter a period between the command and the space preceding the text.

EXAMPLES

Assume that the current line is:

```
--> A LINE
```

Enter the following command:

```
*REPLACE A NEW LINE
```

The new line is displayed:

```
A NEW LINE
```

SYNTAX

SAVE [number of lines / range of lines]

PARAMETERS

number of lines — number of lines to be saved, beginning with the current line.

range of lines — range of lines to be displayed, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.

EXPLANATION

The SAVE command replaces the current contents of the save area (if any) with the specified lines of workspace text. The workspace contents and pointer are not affected.

If neither a number of lines nor a range of lines is specified, only the current line is saved.

The UNSAVE command retrieves saved text.

EXAMPLES

```
*SAVE
```

replaces the contents of the save area with a copy of the current line.

```
*SAVE 30-40
```

replaces the contents of the save area with a copy of workspace lines 30 through 40.

NOTE

The workspace and the save area share the same section of memory in the editor. Storing a large section of text in the save aea may noticeably reduce the capacity of the workspace.

If you receive the message WORKSPACE FULL in response to a SAVE command, there is not enough space in the save area for the text you are storing. You can store the text on a file with a PUT command and retrieve it with a GET command instead of using SAVE and UNSAVE.

SYNTAX

STATUS

EXPLANATION

The STATUS command displays information about the current status of the editor.

EXAMPLES

Enter the command:

```
*STATUS
```

The editor displays the following information:

```
STATUS
PI = FILE1
     LINE   101
PO = FILE1
     LINE    50
LAST AI = FILE2
LAST AO =
COMMAND FILE =
TAB CHARACTER = &
TAB STOPS =   8  16  24  32  40  48  56  64
BRIEF    false
CRT      true
DEBUG    false
ECHO     true
ERROR    true
NUMBER   false
UPARROW  false
XSEARCH  false
XTABS    false
CURRENT LINE:
   15 (   60)
```

This information can be interpreted as follows:

The primary input file (P1) is the current version of file FILE1. One hundred lines have been read from FILE1: the primary input file pointer is at line 101.

The primary output file (PO) is the new version of FILE1. Fifty lines have been written to the primary output file.

The last alternate input file (AI) read from (by a GET or COPY command) is FILE2.

No alternate output files have been written on.

There was no initial command file specified in the EDIT command line.

The tab character is &. The default tab stops are in effect.

All editor flags are at their default values. (TRUE is synonymous with ON; FALSE is synonymous with OFF.)

Workspace line 15 is the current line. There are 60 lines in the workspace.

SYNTAX

SUBSTITUTE {/**old string**/**new string**/}

PARAMETERS

string delimiter: any character except a space. The delimiter character must not occur in either string. All three delimiters must be the same character. If the delimiter character is a letter or period, a space must separate the command from the first delimiter. The final delimiter may be omitted if it is the last character in the command line.

old string — the string of characters you want to replace.

new string — the string of characters that replaces the old string.

EXPLANATION

The SUBSTITUTE command searches the current line for the old string. If the old string is found, the new string replaces it. If the old string does not occur in the current line, the message ** NOT FOUND is displayed.

The workspace pointer does not move.

The strings may be of different lengths. If the new string is empty, the old string is deleted.

After the substitution, the modified line is displayed. To suppress the display, enter a period between the command and the first delimiter.

If the modified line exceeds 127 characters, the line is truncated to 127 characters and the editor displays the message ** TRUNCATED along with the truncated line.

If a SUBSTITUTE command inside repeat brackets ("<" and ">") fails to find a match, command execution resumes with the next command outside the brackets.

If you are using tab stops, remember that while each tab character in the text is displayed as one or more spaces, it is stored as a single character. To include a standard tab character in the old string or the new string, press the TAB key at the appropriate place in the command line.

See the XSEARCH command for an explanation of special search characters.

EXAMPLES

Assume that the workspace contains the following text:

```
——►THIS IS A LINE.
   HERE IS ANOTHER.
```

Enter the following command:

```
*SUBSTITUTE/A/A CHANGED/
```

The modified line is displayed:

```
THIS IS A CHANGED LINE.
```

Enter the following command:

```
*SUBSTITUTE/HERE/THIS/
```

The editor responds:

```
** NOT FOUND
```

To replace a string on a different line, you must use the FSUBSTITUTE command, described earlier in this dictionary.

SYNTAX

SUSPEND

EXPLANATION

The SUSPEND command suspends the editor. Control passes to DOS/50 and the prompt > is displayed. When you enter the DOS/50 command CONT * or CONT EDIT, control returns to the editor. The editor prompts for another command or resumes execution of the command file in progress.

If you enter the DOS/50 command ABORT * or ABORT EDIT while the editor is suspended, the editing session is aborted as with a QUIT command.

SYNTAX

TAB {character}

PARAMETERS

character — the character to be used as the editor tab character. You may not specify the colon (:), less-than (<), greater-than (>), space, or carriage return.

EXPLANATION

The **standard tab character** is CTRL-I (ASCII code 09), which may be entered in one of two ways:

- Press the TAB key.
- Press the I key while holding down the CTRL (control) key.

The TAB command defines the **editor tab character**, which you may use as an alternative to the standard tab character. When you invoke the editor, the editor tab character is undefined. Each TAB command removes the significance of any previous editor tab character.

The editor tab character is like the standard tab character in most respects:

- When you type either tab character, that character is inserted into the line being entered and the cursor advances to the next tab stop.
- When text containing tab characters is displayed, spaces are inserted to align the text to the current tab stops.
- When text containing tab characters is saved on a file:

  If the XTABS flag is ON, each tab character copied is expanded to the appropriate number of spaces.

  If the XTABS flag is OFF, tab characters are copied unchanged.

Unlike the standard tab character, the editor tab character has no meaning outside the current editing session. DOS/50 routines such as ASM and MDL recognize only the standard tab character, and the editor does not remember the editor tab character from a previous editing session.

Since the standard tab character is not printable, you may find it easier to keep track of tabs in your text if you use a printable character as the editor tab character.

EXAMPLES

Create three lines of assembly language source code using the percent sign (%) as the tab character:

```
*TAB %
*INPUT
INPUT:
DEMO%LXI%H,500%; SET TABLE POINTER
%MVI%B,5%; SET PASS COUNTER
%XRA%A%; CLEAR ACCUMULATOR
```

Display the text you have created:

```
*TYPE B-E
DEMO    LXI     H,500    ; SET TABLE POINTER
        MVI     B,5      ; SET PASS COUNTER
        XRA     A        ; CLEAR ACCUMULATOR
```

In order to save the text with spaces instead of percent signs, you must turn the XTABS flag ON:

```
*XTABS ON
*FILE
 ** END OF TEXT

>
```

The following example shows how redefining the editor tab character affects your text. In this example, "t" denotes the standard tab character.

Assume that the workspace is empty and the editor tab character is undefined. Enter the following text:

```
*INPUT
INPUT:
THIStIStAtLINEtOFtTEXT
HERE%IS%ANOTHER
A=BIG=WORD=OVERRIDES=TABS
```

Display the text you have entered:

```
*TYPE B-E
THIS    IS      A       LINE    OF      TEXT
HERE%IS%ANOTHER
A=BIG=WORD=OVERRIDES=TABS
```

Set the editor tab character to % and display the text again:

```
*TAB %
*TYPE B-E
THIS    IS      A       LINE    OF      TEXT
HERE    IS      ANOTHER
A=BIG=WORD=OVERRIDES=TABS
```

Each % in the text is treated as a tab character.

Change the editor tab character to = and display the text again:

```
*TAB =
*TYPE B-E
THIS    IS      A       LINE    OF      TEXT
HERE%IS%ANOTHER
A       BIG     WORD    OVERRIDES       TABS
```

The % is no longer a tab character.

SYNTAX

TABS {column} [column] . . .

PARAMETERS

column — column number of the tab stop. Up to eight tab stops may be specified.

EXPLANATION

The TABS command eliminates all existing tab stops and defines up to eight new tab stops.

A column number may not be less than 1 or greater than 127.

Column numbers must be specified in ascending order.

When the editor is invoked, there are eight default tab stops set at 8, 16, 24, 32, 40, 48, 56, and 64 spaces from the left margin.

Tab stops are not retained from one editing session to the next.

Text is aligned according to the current tab stops, even if the text was created with different tab settings. If the original tab characters have been replaced by spaces, no realignment is possible. See the XTABS command.

EXAMPLES

Set the editor tab character to % and enter the following text:

```
*TAB %
*INPUT
INPUT:
THIS%IS%A LINE%OF TEXT
THIS%%IS%ONE%MORE
%HERE%IS%ANOTHER%ONE
```

Display the text using the default tab stops:

```
*UP.3:TYPE 3
THIS    IS      A LINE  OF TEXT
THIS            IS      ONE     MORE
        HERE    IS      ANOTHER ONE
```

If you specify new tab stops, all the old stops are lost. Enter the following command line:

```
*TABS 10:TYPE 3
```

With only one tab stop, the text looks like this:

```
THIS      IS A LINE OF TEXT
THIS       IS ONE MORE
          HERE IS ANOTHER ONE
```

Note that any tab character beyond the last tab stop is displayed as a space, but remains unchanged in the text. Display the same text with some new tab stops:

```
*TABS 7 14 23 32:TYPE 3
THIS  IS      A LINE    OF TEXT
THIS          IS        ONE       MORE
      HERE    IS        ANOTHER   ONE
```

SYNTAX

```
TYPE [number of lines]
     [range of lines ]
```

PARAMETERS

number of lines	number of lines to be displayed, beginning with the current line.
range of lines	range of lines to be saved, for example 2-6. B, C, and E may be used to represent the first line, current line, and end of workspace, respectively.

EXPLANATION

The TYPE command displays the specified lines of workspace text on the system terminal.

The workspace contents and pointer are not affected.

If neither a number of lines nor a range of lines is specified, only the current line is displayed.

EXAMPLES

```
*TYPE
```

displays the current line.

```
*TYPE 10
```

displays 10 lines beginning with the current line.

```
*TYPE 5-15
```

displays workspace lines 5 through 15.

SYNTAX

UNSAVE

EXPLANATION

The UNSAVE command copies the contents of the save area into the workspace, inserting them before the current line.

The save area is not affected.

The current line remains the same.

EXAMPLES

Assume that the following text is in the workspace:

```
      WELL,
----> HE RAN
      AND
      HE GOT
      VERY TIRED.
```

Enter the following command:

```
*SAVE 2
```

Two lines of text, beginning with the current line, are copied into the save area. The workspace remains the same. The save area contains the following text:

```
HE RAN
AND
```

Enter the following command line:

```
*UNSAVE:UNSAVE
```

Each UNSAVE command inserts a copy of the two saved lines in front of the current line. The workspace now looks like this:

```
      WELL,
      HE RAN
      AND
      HE RAN
      AND
----> HE RAN
      AND
      HE GOT
      VERY TIRED.
```

SYNTAX

UP [number of lines]

PARAMETERS

number of lines number of lines the pointer moves up.

EXPLANATION

The UP command moves the workspace pointer the specified number of lines toward the beginning of the workspace.

The new current line is displayed. To suppress the display, enter a period immediately after the command, before any space or number.

If you do not specify a number of lines, the pointer moves up one line.

If moving the pointer the specified number of lines would cause the pointer to go past the beginning of the workspace, the pointer stops at line 1.

EXAMPLES

Assume that the workspace contains the following text:

```
      A LINE
      B LINE
      C D LINE
----> E LINE IS THE NEXT TO
      LAST LINE
```

Enter the following command:

```
*UP 2
```

The pointer moves up two lines and displays the new current line.

```
B LINE
```

The workspace now looks like this:

```
      A LINE
----> B LINE
      C D LINE
      E LINE IS THE NEXT TO
      LAST LINE
```

SYNTAX

UPARROW [ON / OFF]

PARAMETERS

ON	assigns the uparrow character to special use in representing control characters.
OFF	removes the special significance of the uparrow character.

EXPLANATION

The UPARROW command sets the value of the UPARROW flag. Turn the UPARROW flag ON when you want to display text containing control characters.

Normally (UPARROW flag OFF) the uparrow character (^ or ↑) has no special significance to the editor. Turning the UPARROW flag ON allows any control character to be entered or displayed as an uparrow followed by a printable character. For example, the control characters NULL, BACKSPACE, and ESCAPE are represented as ^@, ^H, and ^ [. The ASCII code of the printable character (64 to 95) is equal to 64 plus the ASCII code of the control character (0 to 31).

All control characters except the carriage return and the current tab character can be represented using the uparrow.

When the UPARROW flag is OFF, the editor displays any control character as itself, and does not treat the uparrow character specially. How a control character is displayed depends on the display device and the control character.

Entering the UPARROW command without a parameter reverses the value of the UPARROW flag.

You may want to think of the uparrow as representing the CRTL (control) key. For example, ↑A represents the control character CTRL-A, ↑B represents CTRL-B, and so on. the 8550 Lab System Users Manual contains an ASCII-Binary-Decimal Conversion Table. Control characters occupy columns 1 and 2 of the table. Columns 5 and 6 contain the printable characters that may be used to represent the control characters.

EXAMPLES

The following table shows how the editor stores the text you enter.

	What the editor stores			
	UPARROW flag OFF		UPARROW flag ON	
What you type	characters	ASCII codes	characters	ASCII codes
^[	^[	94,91	(ESC)	27
^G	^G	94,71	(BELL)	7
^g	^g	94,103	^g	94,103
(CTRL-G)	(BELL)	7	(BELL)	7

The following table shows how the editor displays the text it has stored.

What the editor stores		What the editor displays	
characters	ASCII codes	UPARROW flag OFF	UPARROW flag ON
(ESC)	27	depends on terminal	^[
(BELL)	7	depends on terminal	^G
^G	94,71	^G	^G

Enter the following command:

```
*INSERT THIS BELL GOES ^G^G
```

With the UPARROW flag OFF, all characters in the inserted text are stored just as they are. The ASCII codes for the last four characters stored are 94, 71, 94, 71. Now turn the UPARROW flag ON and insert another line of text.

```
*UPARROW ON
*INSERT THAT BELL GOES ^G^G^G
```

This time, each ^G is stored as a single control character (BELL, ASCII code 7). Note that

ASCII(BELL) = ASCII (CTRL-G) = ASCII(G) — 64

Display both lines of text.

```
*UP.2:TYPE 2
```

With the UPARROW flag ON, each BELL character is displayed as ^G.

```
THIS BELL GOES ^G^G
THAT BELL GOES ^G^G^G
```

Now turn the UPARROW flag OFF again and display the two lines.

```
*UPARROW OFF:TYPE 2
```

This time the exact contents of the lines are sent to the system terminal:

```
THIS BELL GOES ^G^G
THAT BELL GOES (ding) (ding) (ding)
```

Each BELL character rings the bell on the system terminal.

SYNTAX

```
XSEARCH [ON ]
        [OFF]
```

PARAMETERS

ON	enables the wildcard search feature.
OFF	disables the wildcard search feature.

EXPLANATION

The XSEARCH command sets the value of the XSEARCH flag.

When the XSEARCH flag is ON, the characters "?" and ">" have special meanings in search and replacement strings:

- In a search string, "?" matches any single character. In a replacement string, "?" stands for whatever character it matched in the search string.
- The character ">" signifies a carriage return, which is the last character in any line of text. The carriage return separates one line of text from the next.

Additionally, when the XSEARCH flag is ON, a search or replacement string may begin on one line of text and end on another.

When the XSEARCH flag is OFF, "?" and ">" have no special meanings, and strings are not considered to cross or include line boundaries.

Entering the XSEARCH command without a parameter reverses the value of the XSEARCH flag.

The XSEARCH flag is initially OFF.

EXAMPLES

All of the following examples assume that the XSEARCH flag is ON.

SEARCH STRING EXAMPLES

```
*FIND/A?C/
```

finds any three-character string beginning with "A" and ending with "C".

```
*FIND/END.>/
```

finds a line ending with "END.".

```
*FIND/>THE/
```

finds a line beginning with "THE". The line preceding that line becomes the current line, because the carriage return (>) belongs to the line preceding "THE".

```
*FIND/>ALONE>/
```

finds a line containing "ALONE" and nothing else. The line preceding "ALONE" becomes the current line.

REPLACEMENT STRING EXAMPLES

```
*SUB/A?/XYZ/
```

deletes the "A" and whatever character follows it and substitutes the string "XYZ".

```
*SUB/A?C/X?Z/
```

substitutes "X" for "A" and "Z" for "C" but leaves the character between them unchanged.

```
*SUB/ABC?/?ABC/
```

moves the character matched by "?" from behind the string "ABC" to in front of it.

```
*SUB/A?C/A????C/
```

inserts three additional copies of the character between "A" and "C".

```
*SUB/A??D/X?????Y/
```

acting on the string "ABCD" results in "XBCBCBY".

```
*SUB/AC/A?C/
```

inserts a question mark between "A" and "C".

```
*SUB/NEW LINE/NEW>LINE/
```

inserts a carriage return between the words "NEW" and "LINE", making one line into two.

```
*FSUB/NEW>LINE/NEW LINE/
```

combines two lines into a single line.

```
*SUB/NEW>LINE/NEW LINE/
```

produces the response ** NOT FOUND because the SUBSTITUTE command stops searching at the first carriage return.

```
*FSUB/>/>>/
```

inserts an empty line after the current line.

SYNTAX

XTABS [ON / OFF]

PARAMETERS

ON	specifies that tab characters are to be expanded to spaces on output.
OFF	specifies that text is to be output with tab characters.

EXPLANATION

The XTABS command sets the value of the XTABS flag.

When you use tab stops in creating text, each tab character you enter is stored in the text. When one of the following commands copies text from the workspace to a file or device, the editor checks the XTABS flag.

FILE FNEXT NEXT PUT PUTK

If the XTABS flag is ON, each tab character copied is replaced with the number of spaces appropriate to the current set of tab stops.

If the flag is OFF, tab characters are copied unchanged.

Entering the XTABS command without a parameter reverses the value of the XTABS flag.

The XTABS flag is initially OFF.

A file containing standard tab characters is correctly processed by such DOS/50 routines as ASM and MDL, but may produce unexpected results if used as a command file or data file.

Since each tab character generally replaces several spaces, a file created with XTABS OFF usually occupies less disc space than the same file created with XTABS ON.

A file containing non-standard tab characters (defined with the TAB command) is usually useless. Turn the XTABS flag ON before saving text aligned with non-standard tab characters.

EXAMPLES

In these examples "t" denotes the standard tab character.

Assume that the tab stops and XTABS flag are at their default values. The workspace contains four lines of assembly language:

```
tXRAtAt; CLEAR ACCUMULATOR
LOOPtADDtMt; ADD BYTE FROM TABLE
tINXtHt; POINT TO NEXT BYTE
tDCRtBt; DECREMENT PASS COUNTER
```

The following command writes an exact copy of the workspace contents into FILE1:

```
*PUT B-E FILE1
```

Now enter the following command line:

```
*XTABS ON:PUT B-E FILE2
```

The PUT command copies the workspace contents to FILE2. Because the XTABS flag is ON now, each tab character is replaced by spaces up to the next tab stop.

Counting carriage returns, the workspace and FILE1 each contain 120 characters. FILE2 contains 180 characters and looks like this:

```
        XRA     A       ; CLEAR ACCUMULATOR
LOOP    ADD     M       ; ADD BYTE FROM TABLE
        INX     H       ; POINT TO NEXT BYTE
        DCR     B       ; DECREMENT PASS COUNTER
```

Section 4

TECHNICAL NOTES

This section is reserved for technical information about the DOS/50 Editor. At the time of this writing, no technical notes are included. Technical notes will be incorporated into later versions of this manual as necessary.

Section 5

ERROR MESSAGES

ASSIGN PROBLEM (LIST command). The channel reserved for the line printer (LPT) is assigned to some other device or file (the ASSIGN PROBLEM message may also occur as a submessage under the DOS STAT = xx message.)

BOOLEAN? The parameter ON or OFF was entered improperly after one of the following commands: BRIEF, CRT, DEBUG, ECHO, ERROR, NUMBER, UPARROW, XSEARCH, or XTABS.

BREAK. The ESCAPE key was pressed to terminate an editor function.

COMMAND INPUT ABORTED. An attempt was made to enter a command line longer than 127 characters; the entire line is lost.

CANNOT NEST COMMAND FILES. The command file being executed contains a PERFORM command.

CANNOT READ. A write-only file or device was specified for input.

CANNOT READ NEW FILE. A nonexistent or incorrect filespec was specified for input.

CANNOT WRITE. A read-only file or device was specified for output.

DISK FULL. The disc specified to receive output is full and cannot accept any more text.

DOS STAT = xx. A DOS/50 service call error occurred; the Service Calls section in your 8550 System Users Manual explains the SRB status code (xx). This message is followed by submessages that indicate more precisely where the error occurred.

EMPTY SEARCH STRING. The search string was missing, contained no characters between the delimiters, or was not terminated by a second delimiter identical to the first.

END OF FILE. The specified range of lines lies completely past the end of the file. This message may also be displayed when no error has occurred to inform you that the editor has read to the end of a file.

END OF TEXT. The specified range of lines lies completely outside the actual range of lines in the workspace. This message may also be displayed when no error has occurred to inform you that an editor function has reached the end of the workspace.

ERROR (command line). Commands were not separated by colons or too many parameters were entered.

ERROR (MACRO command). The macro being executed contains a MACRO definition.

ERROR (REPLACE command). No text follows the REPLACE command.

ERROR (TABS command). A tab stop was specified out of ascending order, beyond the maximum line length (127), or with a number syntax error. The current tab settings are not altered.

FILE NAME TOO LONG. The specified file name contains more than fourteen characters.

ILLEGAL MACRO NUMBER. A macro number outside the range 1 to 127 was specified or a syntax error was made.

ILLEGAL TAB CHARACTER. An attempt was made to specify a colon (:), left repeat bracket (<), right repeat bracket (>), space, or carriage return as the editor tab character.

MISSING DELIMITER. A delimiter in an FSUBSTITUTE or SUBSTITUTE command was missing.

NEST. Repeat brackets (< >) were not entered in matching pairs.

NEW FILE. Informs you, upon invoking the editor, that you are beginning a new file; there is no primary input file to read data from. Also appears when you have created a new file with the COPY, PUT, or PUTK commands.

NO PI. No primary input file exists and no alternate input file was specified.

NO PO. No primary output file exists and no alternate output file was specified.

NO SAVED TEXT. An attempt was made to retrieve text from the save area but no text had been saved previously.

NOT FOUND. The string being searched for was not found within or beyond the current line. (The SUBSTITUTE command searches only the current line.)

NUMBER NOT ALLOWED. A numeric parameter was entered for a command that does not take a numeric parameter.

NUMBER NOT FOLLOWED BY <. A number was found that should have started a repeated command sequence, but was not followed by the left repeat bracket (<).

NUMBER? The line number parameter contained a syntax error.

PROCEDURE ERROR. A primary file was illegally specified as an alternate file.

RANGE? The range-of-lines parameter contained a syntax error.

SUPERSEDING EXISTING FILE. Warns that you have specified an existing file as the primary output file. The previous contents of the file will be lost if you do not terminate the editor with the QUIT command before copying any text to the primary output file.

TRUNCATED. An attempt was made to enter or create a text line longer than 127 characters. In an INPUT command, the line is lost and the message **TRUNCATED replaces the lost text. In an FSUBSTITUTE or SUBSTITUTE command, the line is truncated to 127 characters.

UNKNOWN COMMAND. A nonexistent or misspelled command was entered.

WORKSPACE FULL. There is an insufficient amount of free area left in the workspace buffer to execute the latest editor command. To gain more free area, KILL or PUTK some workspace lines, delete some macros, or SAVE a smaller block of text. This message may also be displayed after the NEXT or FNEXT commands to inform you that the workspace has been filled to three-fourths of its capacity.

Section 6

GLOSSARY

Alternate Input File (AI). A file that provides text to be edited. It may be any text file other than the primary input file or the primary output file.

Alternate Output File (AO). A file that receives edited text. It may be any text file other than the primary input file or the primary output file.

Command File. A file containing editor command lines as text. When a command file is invoked by the PERFORM command, the command lines it contains are executed.

Command Line. A line of one or more editor commands. Multiple commands must be separated by colons (:). The line may not contain more than 127 characters.

Current Directory. The directory that contains the files you are currently using. A filespec that does not begin with a slash specifies either a standard device (such as CONI) or a file or directory in the current directory.

Current Line. A line of text in the workspace that serves as a point of reference for many editor commands. The current line is pointed to by the workspace pointer, which may be moved to any line in the workspace.

Directory. A file that may contain only pointers to other files. See the 8550 System Users Manual for complete information on files and directories.

DOS/50. The **D**isc **O**perating **S**ystem of the 8**550** Microprocessor Lab.

Editor. The DOS/50 system program that allows a user to create and modify text files conveniently.

Editor Tab Character. See **Tab Character**.

Filespec. A sequence of names, separated by slashes, that defines a path to a file. A file that is pointed to by the current directory may be specified with a single name. The term **filespec** in a Command Dictionary syntax block may refer to a standard device name such as CONI.

Free Area. The area of the workspace buffer that is still available for use by the workspace, macro area, and save area.

Initial Command File. An editor command file that is automatically executed at the beginning of the edit session. The initial command file is an optional parameter of the EDIT command. See **Command File**.

Macro. A specially defined command line that may be executed by a short command rather than by typing in the whole line. Up to 127 macros may be stored in the macro area. Macros are defined and executed by the MACRO command.

Macro Area. The area of the workspace buffer that holds the macros defined by the user for the current editor session.

Primary Input File (PI). The default file from which text is read into the workspace to be edited. This file may be specified when the editor is invoked.

Primary Input File Pointer. A pointer maintained by the editor that points to the next line to be copied from the primary input file. The primary input file pointer points initially to the first line of the primary input file.

Primary Output File (PO). The default file into which edited text is stored from the workspace. This file may be specified when the editor is invoked.

Program Memory. The 8301 memory used as a substitute for prototype memory in the early stages of prototype development (emulation modes 0 and 1). User programs run in program memory, as does the editor and certain other system programs.

Repeat Brackets. The characters "<" and ">" used as brackets to enclose a command sequence that is to be repeated.

Save Area. The area in the workspace buffer that stores the text copied from the workspace by the most recent SAVE command.

Standard Tab Character. See **Tab Character**.

Tab Character. A character entered to instruct the editor to include spaces in a text line so that the next character of text is in a tab column. The standard tab character (CTRL-I, ASCII code 09) may be entered by pressing the TAB key, or by pressing the I key while holding down the CTRL key. An optional editor tab character, defined by the TAB command, may be used in place of the standard tab character, CTRL-I.

Workspace. The area of the workspace buffer that holds the text currently being edited. Most editor commands operate on the text currently in the workspace.

Workspace Buffer. The area of program memory used by the editor to store text. The editor program occupies about 9K bytes of program memory. The rest of program memory constitutes the workspace buffer. Each of four areas of the workspace buffer (workspace, free area, save area, and macro area) may vary in size, but the size of the workspace buffer remains constant.

Workspace Map. A two-line display produced by the editor that gives the size and location of each area of the workspace buffer. The command DEBUG ON causes the workspace map to be displayed after execution of each editor command.

Workspace Pointer. A pointer maintained by the editor that points to the current line in the workspace. See **Current Line**.

Section 7

INDEX

A

AI. *See* Alternate input file
AO. *See* Alternate output file
Aborting the editor, 3-53
AGAIN command, 3-6
Alternate input file, 3-1, 3-20
Alternate output file, 3-1, 3-20
Angle brackets, 3-3
    See also Repeat brackets
Asterisk:
    editor prompt, 1-4
    with repeat brackets, 3-3

B

BACKSPACE key, 1-4
Backup file, 3-21, 3-22
Batch editing. *See* Command file
BEGIN command, 3-7
Blank line, 3-36
BRIEF command, 3-8

C

Cancelling. *See* Deleting
Carriage return in a search string, 3-72
Character string. *See* String
Characters, special:
    asterisk, 1-4, 2-25, 3-3
    carriage return, 3-72
    colon, 3-3, 3-61
    greater-than, 3-3, 3-61, 3-72
    less-than, 3-3, 3-61
    period, 3-3
    question mark, 3-72
    tab character, 1-5, 3-61, 3-74, 6-2
    See also Control characters
Colon, 3-3, 3-61
Command file, 3-49
    creating, 2-27,
    executing, 2-27, 3-49
    initial, 2-28, 3-20
Command line, 3-3
    correcting mistakes in, 1-4
    special characters in, 3-3
    stringing together commands in, 2-24
COMMENT command, 3-10
Control characters, 3-5, 3-69
Control-I. *See* Tab character
Converting tab characters to spaces. *See* XTABS command
COPY command, 3-11
    examples, 2-8, 2-21
Copying text:
    between files, 3-11
    from a file, 2-22, 3-34, 3-45
    from the save area, 3-67
    to a file, 2-21, 3-26, 3-51, 3-52
    to the save area, 3-55
    See also Saving text
Correcting typing mistakes, 1-4
Creating a file, 2-4
Creating text, 1-5, 3-36, 3-38
CRT command, 3-13
CTRL-I. *See* Tab character
Current line, 3-2
    display of, 2-11, 3-8
    See also Workspace pointer

D

DEBUG command, 3-14
DELETE key, 1-4
Deleting a string, 3-32, 3-58
Deleting lines of text, 3-39, 3-52
Deleting tab characters. *See* XTABS command
Deleting the last character typed, 1-4
Deleting the line being typed, 1-5
Demonstration Run, 1-4
Displaying text:
    from a file, 2-8
    from the workspace, 1-6, 2-8
    on the line printer, 1-10, 2-8
    on the system terminal, 2-8
        interrupting display, 3-5
DOWN command, 3-17
Duplicating. *See* Repeating

E

ECHO command, 3-18
EDIT command, 3-19
Editing a large file, 2-18, 3-29, 3-45
Editing an existing file, 2-5

Editor:
aborting, 3-53
exiting, 2-3, 3-26, 3-53
interrupting, 2-3, 3-4, 3-60
invoking, 2-2, 3-19
Editor tab character. *See* Tab character
Empty line, 3-36
END command, 3-24
Entering text, 1-5, 3-36, 3-38
Erasing. *See* Deleting
ERROR command, 3-25
Error messages, 3-25, 5-1
ESC key, 3-4
Exiting the editor, 2-3, 3-26, 3-53
Expanding tab characters to spaces. *See* XTABS command

F

Features of the editor, 1-3
File:
creating, 2-4
displaying, 2-8
modifying, 1-7, 2-5
FILE command, 3-26
FIND command, 3-27
Finding a string, 2-14
FNEXT command, 3-29
Free area, 3-2, 3-14
FSUBSTITUTE command, 3-32

G

GET command, 3-34
Global string replacement, 2-16
Greater-than sign, 3-3, 3-61, 3-72

H

Halting Display, 3-5

I

Initial command file, 2-28, 3-20
INPUT command, 3-36, 3-38
INSERT command, 3-38
Interrupting display, 3-5
Interrupting the editor, 2-3, 3-4, 3-60
Invoking the editor, 2-2, 3-19

K

Keys, special, 3-4
BACKSPACE, 1-4
DELETE, 1-4
ESC, 3-4
RUBOUT, 1-4, 3-13
TAB, 3-5, 3-61
KILL command, 3-39

L

Less-than sign, 3-3, 3-61
Line numbers:
from beginning of file, 3-56
from beginning of workspace, 1-9, 3-2, 3-47
Line pointer. *See* Workspace pointer
Line printer, 2-8, 3-40
LIST command, 3-40
Listing. *See* Displaying
LN command, 3-41

M

Macro, 2-25, 3-42, 6-1
Macro area, 3-2, 3-14
MACRO command, 3-42
Modifying an existing file, 1-7, 2-5
MOVE command, 3-44
Moving text:
backward in the file, 2-21
forward in the file, 2-20
within the workspace, 1-9, 3-44
Moving the workspace pointer, 2-10
Multiple commands in a line, 2-24

N

NEXT command, 3-45
examples, 2-5, 2-18
NUMBER command, 3-47

P

PI. *See* primary input file
PO. *See* primary output file
PERFORM command, 3-49
example, 2-27
Period, 3-3, 3-8
Pointer. *See* Workspace pointer
Primary input file, 3-1, 3-19, 3-20

Primary input file pointer, 3-56, 6-2
Primary output file, 3-1, 3-19, 3-20
Printing text. *See* Displaying text
Program memory, 6-2
PUT command, 3-51
PUTK command, 3-52

Q

Question mark, 3-72
QUIT command, 3-53

R

Repeat brackets, 3-3, 6-2
Repeating a block of text, 2-12
Repeating a command sequence:
    using a macro, 3-42
    using repeat brackets, 2-24, 3-3
    See also AGAIN command
REPLACE command, 3-54
Replacing a line, 3-54
Replacing a string, 2-15
    repeatedly, 2-16
RUBOUT key, 1-4, 3-13

S

Save area, 3-2, 3-14, 3-55
SAVE command, 3-55
Saving text:
    in an alternate output file, 2-21, 3-51, 3-52
    in the primary input file, 1-6, 3-26, 3-45, 3-51, 3-52
    in the save area, 3-55
Special characters. *See* Characters, special
Special keys. *See* Keys, special
Special search feature, 3-72
Standard tab character. *See* tab character
STATUS command, 3-56
Status of editor, 3-14, 3-41, 3-56
String,
    deleting, 3-32, 3-58
    finding, 2-14
    replacing, 2-15
        repeatedly, 2-16
    special features, 3-72
SUBSTITUTE command, 3-58
SUSPEND command, 3-60
Syntax conventions, 3-1

T

Tab character, 1-5, 3-61, 3-74, 6-2
TAB command, 3-61
TAB key, 3-5, 3-61
Tab stops, 3-64
TABS command, 3-64
Terminating the editor. *See* Exiting the editor
Text. *See* Copying text, Displaying text, Entering text, Moving text, Saving text
TYPE command, 3-66

U

Underlined characters:
    in procedures, 2-1
    in syntax blocks, 3-1
UNSAVE command, 3-67
UP command, 3-68
UPARROW command, 3-69
Uses of the editor, 1-2

W

Wildcard search feature, 3-72
Workspace, 3-2, 3-14
Workspace buffer, 3-2, 3-14
    size of, 3-2
Workspace map, 3-14
Workspace pointer, 3-2
    moving, 1-8, 2-10

X

XSEARCH command, 3-72
XTABS command, 3-74
    example, 2-6