File No. S370-36
Order No. GC20-1805-3

Systems

IBM Virtual Machine Facility/370: EDIT Guide

| Release 2

File No. S370-36
Order No. GC20-1805-3

Systems

IBM Virtual Machine Facility/370: EDIT Guide

| Release 2 PLC 13

This publication explains, for users of the Conversational Monitor System (CMS), how to use the CMS Editor to create, examine, and modify files at the terminal. It contains descriptions of:

- The EDIT command.
- The EDIT subcommands.
- The EDIT macros.

Examples are also included.

The *IBM Virtual Machine Facility/370: Command Language Guide for General Users,* Order No. GC20-1804, is a prerequisite for thorough understanding and effective use of this publication.

IBM

Preface

This publication is intended for system programmers, application programmers, and other terminal users. It describes how to use the CMS EDIT program to create and change files at the terminal.

It is divided into six sections, plus two appendixes.

"Section 1: Introduction" describes the facilities of the CMS Editor and includes an introduction to editing with an IBM 3277 Display Station.

"Section 2: Command Modes" describes how to change from one command mode to another.

"Section 3: EDIT Subcommands" describes each subcommand in detail. If a subcommand operates differently at a typewriter terminal than it does at a display terminal, the differences are discussed.

"Section 4: EDIT Macros" discusses macro conventions and use.

"Section 5: Operational Characteristics" discusses unique CMS Editor operations and functions.

"Section 6: Error Conditions and Recovery Procedures" provides information on error messages and conditions.

"Appendix A: Summary of EDIT Subcommands" lists the subcommands with their operands, for quick reference.

"Appendix B: User-Written EDIT Macros" illustrates some EDIT macros that you can write for your own use.

PREREQUISITE PUBLICATION

IBM Virtual Machine Facility/370: _Command Language Guide for General Users_, Order No. GC20-1804.

Fourth Edition (January 1975)

This is a major revision of GC20-1805-3, and makes obsolete that edition
| and Technical GN20-2641. This edition, together with Technical
| Newsletter GN20-2660, dated March 31, 1975, corresponds to _Release 2 PLC_
| _13_ (Program Level Change) of the IBM Virtual Machine Facility/370 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest _IBM System/360 and System/370 Bibliography_, Order No. GA22-6822, and its _Virtual Storage Supplement_, Order No. GC20-0001, for the editions that are applicable and current.

Technical changes and additions to text and illustrations are indicated by a vertical bar to the left of the change.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, VM/370 Publications, 24 New England Executive Park, Burlington, Massachusetts 01803. Comments become the property of IBM.

Summary of Amendments
for GC20-1805-3
as updated by TNL GN20-2660
VM/370 Release 2 PLC 13

3270 DISPLAY SYSTEM AS A REMOTE VIRTUAL MACHINE CONSOLE

New: Program Feature

The IBM 3270 Display System is now supported as a remote virtual machine console when attached to a 2701 Data Adapter Unit or 2703 Transmission Control Unit via a nonswitched line.

Remote 3270s controlled by an IBM 3704 or 3705 Communications Controller must be controlled by the Emulation Program (EP) or the Partitioned Emulation Program (PEP) in EP mode.

Using this support, 3270 Display System users can have copies of 3277 screen displays printed on 3284, 3286, or 3288 printers at their remote locations.

The section "Editing with a Display Terminal" has been updated to reflect this support.

CMS EDITOR CHANGES

New: Program Features

New EDIT Command Option: NODISP

The new NODISP option of the EDIT command places a 3270 display terminal in LINE (typewriter terminal) mode for the entire EDIT session.

New EDIT Subcommands: DSTRING and FORMAT

The new DSTRING subcommand allows you to delete any number of lines beginning at the current line and ending at the line containing a specified character string.

The new FORMAT subcommand allows you to change the mode of your 3270 display terminal from DISPLAY (display terminal) mode to LINE (typewriter terminal) mode, or from LINE mode to DISPLAY mode.

Changed: Program Features

Changes to the VERIFY, SCROLL and AUTOSAVE Subcommands

The VERIFY subcommand has been changed to allow you to specify the verification start column as well as the verification end column.

The SCROLL subcommand syntax has been changed: SCROLL specifies a forward scroll and SCROLLUP specifies a backward scroll. The F and B operands are eliminated with this change.

The AUTOSAVE subcommand has been changed so that it now treats multiline changes caused by a single CHANGE or OVERLAY subcommand as one update.

160-Character Records Edited

The maximum length of a record that can be edited has been changed to 160 characters. Although you cannot create 160-byte records using the CMS Editor, you can edit an existing file that contains 160-byte records.

DOCUMENTATION CHANGES

The changes that appear in this technical newsletter reflect the programming changes listed above and minor technical and editorial changes.

DOCUMENTATION CHANGES

This manual has been revised for this release. The following changes have been made:

- The discussion "Editing with a 3270 Display Terminal" in "Section 1: Introduction" has been rewritten.
- "Figure 2. Sample Settings of 3270 Program Function Keys" has been included to show one way you could define a 3270's program function keys to make them particularly useful for editing.
- "Figure 4. Access Modes for CMS Files" has been revised.
- In the discussion "Serialization of Records," the list of serialization defaults for various filetypes has been corrected.
- The discussion "Subcommand and Macro Notation Conventions" has been deleted. This information is found in the _VM/370: Command Language Guide for General Users_.
- Two examples of editing a file at a typewriter terminal in "Line Number Editing" have been corrected.
- Information has been added to the description of the CHANGE subcommand. Use of the CHANGE subcommand without operands causes unprintable characters to be removed from the current line.
- Information about FREEFORT and ASSEMBLE files has been added to the discussion of the LINEMODE subcommand.
- The full record length of BASIC and VSBASIC files is now the default when the TRUNC, VERIFY, and ZONE subcommands are issued for these files.
- The section "3270 Display Terminal Subcommands" has been deleted. This information is contained in "Section 3: EDIT Subcommands."
- The discussion "Writing EDIT Macros" has been rewritten to clarify the information about stacking EDIT subcommands.

CMS EDITOR LINE RENUMBERING FEATURE

<u>New</u>: Program Feature

A new CMS Editor subcommand, RENUM, recomputes the line numbers for VSBASIC and FREEFORT source files. The following changes were made:

- The "Line Number Editing" section of "Section 1: Introduction" summarizes the RENUM subcommand.

- "Section 3: EDIT Subcommands" describes the new RENUM subcommand in detail and gives additional information for the GETFILE subcommand.

- The "Editor Messages" section of "Section 6: Error Conditions and Recovery Procedures" lists and describes the messages related to the line renumbering feature.

- "Appendix A: Summary of EDIT Subcommands" is updated to include the RENUM subcommand.

CMS EDITOR SUPPORTS IBM 3270 DISPLAY TERMINAL

New: Program Feature

The CMS Editor now supports the IBM 3270 Display terminal. The following changes reflect this support:

- "Section 1: Introduction" describes editing with a display terminal.
- "Section 3: EDIT Subcommands" has a new section, "3270 Display Terminal Subcommands," which describes the EDIT subcommands that are supported only for the 3270 display terminal or work somewhat differently with the 3270. Also, the following new subcommands are described in "Section 3: EDIT Subcommands:"
  - AUTOSAVE allows automatic saving of the current file.
  - BACKWARD moves records down the screen and moves the current line pointer backward in the file.
  - FORWARD moves records up the screen and moves the current line pointer forward in the file.
  - SCROLL allows automatic display of the entire file in increments of 20 lines, for one-minute intervals.

MISCELLANEOUS CHANGES

Changed: Documentation Only

The *VM/370: EDIT Guide* contains many small editorial and technical changes too numerous to list.

Contents

FIGURES

Section 1: Introduction

CMS EDITOR

The CMS EDIT facility (hereinafter referred to as the Editor) allows you to:

- Create sequential files containing fixed- or variable-length records.
- Use standard default values, with certain filetypes, for record length, format, tab settings, serialization, lowercase to uppercase translation, and other file attributes.
- Add, delete, or change any part of a file.
- Move one or more lines from one place in the file to another.
- Search and change a file using context or line editing.
- Receive prompting with line numbers, if desired.
- Display all or part of the file.
- Use the CMS EXEC interpreter to execute user-written EDIT macros.

| EDITOR MODES: EDIT AND INPUT MODE

The Editor has two modes of operation, EDIT mode and INPUT mode. When you issue an EDIT command at the terminal, or within an EXEC procedure, EDIT mode is always entered, whether or not the specified file exists.

If the file already exists, CMS locates it and you can begin working on it. If the file does not exist, CMS creates a new file having the filename and filetype you specified in the EDIT command.

If you want to start putting new data into the file, type INPUT to get into INPUT mode. The Editor considers all information keyed in from this point on to be input data until you enter a null line (that is, a line containing no data). This null line causes the Editor to return to EDIT mode so that you can invoke EDIT subcommands, if you wish, before storing the new file on disk. On the other hand, if you do not want to enter INPUT mode immediately, you can enter EDIT subcommands to add to, examine, alter and rearrange the contents of an existing file.

EDITING WITH A 3270 DISPLAY TERMINAL

The IBM 3277 Display Station (hereinafter referred to as the 3270) allows you to edit files and obtain the same results as you would with a typewriter terminal.

This section describes the screen layout of the 3270, outlines some considerations for editing files at a 3270, and describes how to define and use the 3270 program function keys. If you do not have a 3270 display terminal, go on to the section entitled "EDIT Command."

3270 Display Terminal Modes: DISPLAY and LINE Mode

A local or remote 3270 display terminal can operate in either DISPLAY mode or LINE mode. In DISPLAY mode, the screen appears as shown in Figure 1; the file that is being edited is displayed in the output display area. In LINE mode, the screen appears as if it were a typewriter terminal's console sheet, except that the screen status (for example, VM READ) appears in the lower right-hand corner of the screen. The commands and subcommands you issue appear in the output display area in the order in which you issued them.

A remote 3270 terminal is in LINE mode unless you issue the FORMAT DISPLAY subcommand, which puts the terminal into DISPLAY mode for the remainder of the EDIT session. (Each time you enter INPUT mode, however, the terminal temporarily returns to LINE mode.)

A local 3270 terminal is in DISPLAY mode unless you:

1. Begin an EDIT session by issuing the EDIT command with the NODISP option, or

2. Issue the FORMAT LINE subcommand.

If you use the NODISP option, the 3270 goes into LINE mode for the entire EDIT session. You cannot change it to DISPLAY mode by issuing the FORMAT DISPLAY subcommand. If you have issued the FORMAT LINE subcommand, however, the 3270 goes into LINE mode, but you can return to DISPLAY mode by issuing FORMAT DISPLAY.

The NODISP option is useful if you want your previous console output to remain on the screen when you enter EDIT mode (for example, during execution of an EXEC procedure).

The terminal's response is faster in LINE mode than in DISPLAY mode, because each subcommand you issue in LINE mode causes fewer changes to the screen than it would in DISPLAY mode. For example, issuing a subcommand that changes the current line pointer (such as UP, DOWN, LOCATE, or FIND), in DISPLAY mode causes the entire output display area to be rewritten (up to 1600 bytes). One of these subcommands issued in LINE mode, however, causes only one line to be written (up to 160 bytes).

In DISPLAY mode, therefore, terminal response is slower but you are able to see how the file you are editing looks and how it is changed by each EDIT subcommand you issue. In addition, subcommands that are unique to display terminals, such as SCROLL and CHANGE with no operands, can only be used when the terminal is in DISPLAY mode.

Except for speed of response, a local terminal in LINE mode and a remote terminal in LINE mode function alike. A local terminal in DISPLAY mode and a remote terminal in DISPLAY mode function alike except when the Editor goes into INPUT mode. A local terminal in DISPLAY mode remains in DISPLAY mode when it enters INPUT mode. A remote terminal, however, returns to LINE mode when it enters INPUT mode. After you have finished entering input and return to EDIT mode, the remote terminal returns to DISPLAY mode. For more information about display terminals in INPUT mode, see the description of the INPUT subcommand.

3270 Screen Layout for Editing

The screen of a 3270 display terminal in DISPLAY mode contains the following data during an editing session, in the format shown in Figure 1:

- Edit Session Status--NEWFILE, EDIT, or INPUT. NEWFILE appears if you edit a new file, EDIT or INPUT appears when you enter the first subcommand, and INPUT appears when you enter input mode.
- File ID--fn ft fm F/V lrecl (filename, filetype, filemode, fixed or variable length, logical record length).
- Message--Any VM/370 message to the user, preceded by five "greater than" (>>>>>) characters.
- Output Display Area--up to 20 lines of data. You cannot key data into this area of the screen.
- Current Line Pointer (CLP)--Positioned at line 9, indicates the current input or editing line. Line 9 is the line that the 3270 System Available indicator appears on.
- ¤ (Lozenge)--Indicates an unusable position occupied by a 3270 attribute control character. Thus, line 21 (the last line of the output display area) can display only the first 79 characters of a line since the screen attribute character occupies position 80. The screen attribute characters appear as blanks on the screen.
- User Input Area--80 characters on line 22 and 56 characters on line 23; a total of 136 characters.
- Screen Status--The screen status conditions are: RUNNING, MORE..., HOLDING, CP READ, VM READ, and NOT ACCEPTED. See the _VM/370: Terminal User's Guide_ for details about screen status conditions.

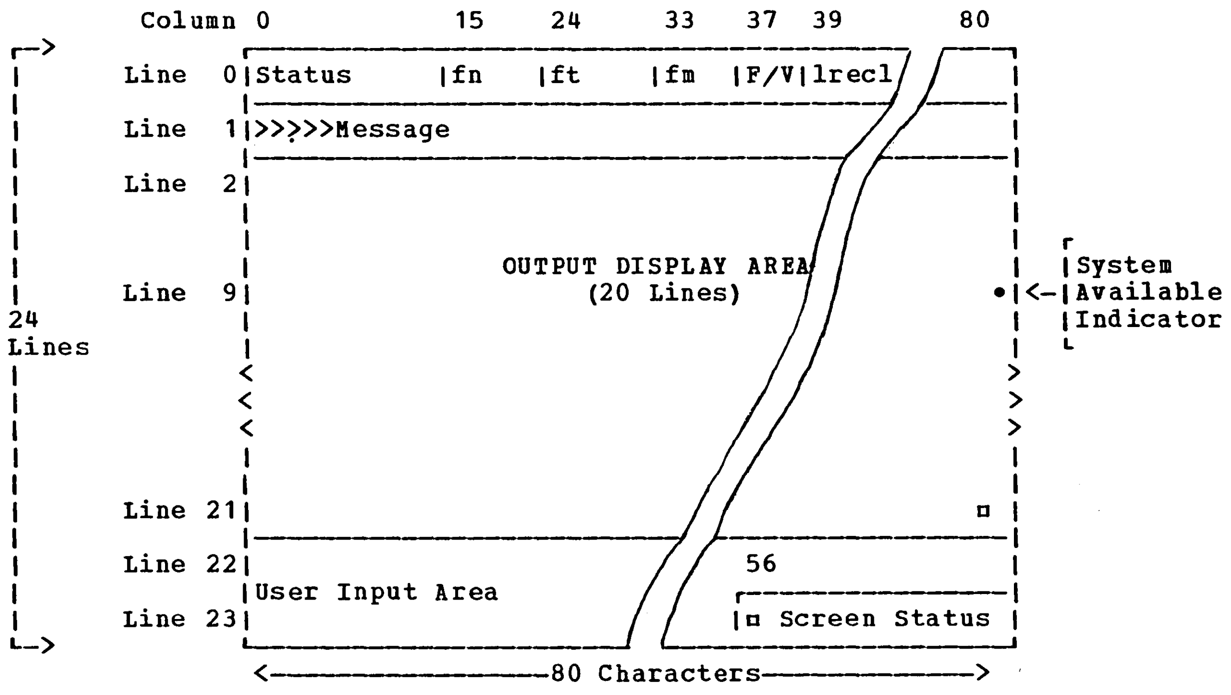


Figure 1. 3270 Display Terminal Screen Layout

Editing a File with a 3270 Display Terminal

Some EDIT subcommands work differently on a 3270 terminal in DISPLAY mode than they do on a typewriter terminal or a 3270 in LINE mode. The following briefly notes some characteristics of the 3270 in DISPLAY mode that are important in editing.

- The output display area, lines 2 through 21, displays up to 20 file records. The current line (that is, the line being edited) always appears on line 9.
- You can use the SCROLL subcommand to display up to 20 lines of the file at once, allowing a better visualization of format and editing changes. This results in faster file editing because it eliminates unnecessary checking and searching for lines, and replaces slow-speed typing with high-speed displays.
- The CHANGE subcommand allows you to change the current line in the user input area by using the 3270 INSERT and DELETE keys, the cursor controls, and the keyboard. You can use the cursor controls to move the cursor around in the user input area and then retype, insert, or delete characters in the current line. For more information about using the cursor, see the *VM/370: Terminal User's Guide*.
- When VERIFY is set to ON, the EDIT subcommands that change the file you are editing cause a new display reflecting the change to replace the current display.
- The FORWARD and BACKWARD subcommands allow you to move the file display any number of lines in either direction.
- The Editor does not redisplay the EDIT command line itself in the output display area unless the terminal is in LINE mode. In DISPLAY mode, only data lines are displayed in the output display area. You can write a chronological history of your EDIT commands and subcommands on the virtual console spool file when using the 3270, by issuing the CP command SPOOL CONSOLE START when you begin. However, output display area information written by the Editor is not spooled in DISPLAY mode, but it is spooled in LINE mode.
- The "?" (query) subcommand causes the last EDIT subcommand you entered to be redisplayed in the user input area. You can now repeat this subcommand by pressing the 3270 ENTER key.

Using the 3270 Program Function Keys

The 3270's program function keys can be very useful for editing purposes if you set them to execute EDIT subcommands when they are pressed. Then, whenever you want to execute one of these subcommands, you need not key it in; you simply press the program function key. Thus you can avoid repeatedly keying in frequently-used subcommands.

To set a program function key to a subcommand, enter the CP command SET PFnn IMMED, followed by the subcommand you want the PF key to perform. For example, you could issue:

```
SET PF02 IMMED UP 5
```

Now, whenever you wish to move up in the file five lines, press the PF02 key; the subcommand UP 5 will be executed immediately.

The SET command cannot be issued in EDIT mode; you must set your PF keys before issuing the EDIT command, or press the PA1 key to bring you into the CP command environment.

Figure 2 shows the layout of the 3270 program function keys with sample settings. To see how to set a program function key for logical tab settings (PF10 in Figure 2), see "Defining a 3270 Program Function Key for Tab Settings" under "Tab Settings."

PF1 SCROLLUP	PF2 UP 5 (BACKWARD 5)	PF3 UP 1 (BACKWARD 1)
PF4 SCROLL	PF5 NEXT 5 (FORWARD 5)	PF6 NEXT 1 (FORWARD 1)
PF7 TABSET 1 10 16 31 36 72 80	PF8 CMS	PF9 RETURN
PF10 CP TAB (values equal to those of PF7)	PF11 TYPE	PF12 INPUT

Figure 2. Sample Settings of 3270 Program Function Keys

Remote 3270 COPY Function

If you are using a remotely-connected 3270 display terminal, you can copy the full screen display currently appearing on the screen. The SET PFnn COPY command allows you to assign a COPY function to a specified PF key. Pressing the PF key copies the current display on the screen by printing the display on a 3284, 3286, or 3288 printer that is attached to the same control unit as the display terminal.

Figure 2.1 is an example of a remote 3270 screen display that could be copied on the printer.

The user identification in Figure 2.1 is an identifying name that you can give the sheet; this is desirable if more than one remote terminal is using the printer. To enter this user identification, type it into the screen's user input area just before you press the PF key that is set to execute the COPY function.

If you use the COPY function frequently, you can set a PF key with a user identification as follows:

```
SET PFnn yourname...dept no.
```

Press this PF key just before you press the PF key that you have set to execute the COPY function.

```
DEFINE STORAGE 16384K
STORAGE = 16384K
IPL 190
CMS VERSION n.n mm/dd/yy  hh:mm
  .
  .
  .
  .
  .
  .
  .
user identification
                                                RUNNING
```

Figure 2.1 Remote 3270 Screen Display

See the description of the SET command in the _VM/370: Command Language Guide for General Users_ for detailed information about printing screen displays.

EDIT COMMAND

The format of the EDIT command is:

```
Edit     | filename filetype [filemode] [(options...[)]]
         |
         |                               options:
         |                              [LRECL nnn]
         |                              [NODISP]
```

where:

filename
  Indicates the name of the file to be processed. You must specify a one- to eight-character alphameric filename. If no filename is entered, an error message is displayed.

filetype
  Indicates the type of the file to be processed. You must specify a one- to eight-character alphameric filetype. If no filetype is entered, an error message is displayed.

filemode
  Indicates the filemode of the file, which identifies the disk where the file resides, and the mode of access to the file. The filemode can be that of the parent disk if the file resides on an extension of that disk.

Options:

LRECL nnn
Indicates the record length of the file, where nnn is one to three decimal digits. The maximum record length (nnn) you can specify is 160. See the discussion on "Record Length" in this section.

NODISP
Places a 3270 display terminal in LINE mode (that is, operating as a typewriter terminal) for the entire EDIT session. Display terminal subcommands, such as SCROLL, are not valid. If you specify NODISP, you cannot change to DISPLAY mode by issuing the FORMAT DISPLAY subcommand.

If a filemode is specified, the Editor searches only that disk and its extensions for the file. If the filemode is not specified (that is, left blank), the Editor searches your A-disk and any extensions. If the filemode is specified as an asterisk (*), then the Editor searches all of your CMS disks for the file.

If it finds the file, the Editor saves the filemode of the disk on which it found the file. If the Editor finds the file on an extension, it saves the filemode of the parent disk.

When the file is to be written out (due to a FILE, SAVE, or AUTOSAVE subcommand), the Editor tries to write the file on the disk whose filemode was saved. If this disk is read-only, however, the file cannot be written on it. The following error message is issued:

```
SET NEW FILEMODE AND RETRY
```

You must then reissue the FILE or SAVE subcommand, specifying the filemode of one of your read-write disks. Or, if future AUTOSAVEs will be issued for the file, change the filemode with the FMODE sucommand.

If the Editor does not find the file on any disk when you issue the EDIT command, it allows you to create a new file with the file identifications you specified. When the newly-created file is to be written out, the Editor writes it on the disk specified by filemode, or if you did not specify filemode, on your A-disk.

Record Length

If you do not specify a record length in the EDIT Comand line, the Editor assumes the following defaults:

- Editing Existing Files: The existing record length is used for all filetypes regardless of format (fixed or variable).
- Creating New Files:

Filetype	Record Length	Format
LISTING	121 characters maximum	Variable
SCRIPT	132 characters maximum	Variable
FREEFORT	81 characters maximum	Variable
All others	80	Fixed

The largest record you can edit with the Editor is 160 characters. A file with record length up to 160 bytes (for example, a listing file created by a DOS program) can be displayed and edited.

The largest record you can create with the Editor, however, is 130 characters. If you key in more than 130 characters of input, the record is truncated to 130 characters when you press the ENTER or RETURN key.

For most purposes, you will not need to create records longer than 130 characters. If it is necessary, however, you can expand a record that you have entered. You do this by issuing the CHANGE subcommand with operands, to add more characters to the record (for example, by changing a one-character string to a 31-character string). However, if you later use the CHANGE subcommand without operands to bring a record down into the user input area, anything in that record beyond 136 characters is truncated.

You can specify any filetype and assign it any record length up to 160 bytes. For example, you could begin to create a new file with a filetype of X and a record length of 145 by issuing:

```
    edit payday x (lrecl 145
```

The default record lengths of the standard filetypes, like LISTING and SCRIPT, can be overrideen by the LRECL option of the EDIT command. For example, you could issue either of these commands:

```
    edit newfile script (lrecl 150
    edit newfile script (lrecl  80
```

You cannot create a record that is longer than the record length of the file. For example, if the file you are editing has a default record length of 80, or if you specified LRECL 80 when you created the file, the Editor truncates all records to 80 characters.

Using the LRECL option, you can override the record length of an existing file and make it larger, but not smaller.

If you try to override the record length of an existing file and make it smaller, the Editor displays an error message, and you must issue the EDIT command again with a larger record length. For example, suppose you have on your B-disk a file named MYFILE FREEFORT, which was created with the default record length of 81. If you try to edit that file by issuing:

```
    edit myfile freefort b (lrecl 72
```

the Editor displays the message:

```
    GIVE A LARGER RECORD LENGTH.
```

You must then issue the EDIT command again and either specify a length of 81 or more, or allow it to default to the current record length of the file.

LINE POINTER

- A Current Line Pointer (CLP) is associated with each file being edited.
- The current line is the line that is being created or edited in a file.
- The current line pointer identifies the line in the file that is the current line.

Various EDIT subcommands and macros exist for moving the current line pointer. The line pointer may be moved to:

1. A record specified by its relative displacement (in number of records) forward or backward from the current position of the pointer.

2. A record containing a specified string of characters or having a specified label. A string of characters can be any group of alphameric characters. If the EDIT subcommand used requires delimiters such as a slash (/) around the string, blanks may be included in the string.

3. A specific record (indicated by the record number).

The ability to search for strings allows you to be concerned with only the specific data to be manipulated, freeing you from the task of keeping track of the locations of specific records or the counts of inserted and deleted records. The Editor allows you to change one, several, or all occurrences of a specific string that exist in a file. This global change capability eliminates many tedious and repetitious editing chores.

When you change from INPUT to EDIT mode, the pointer remains positioned at the last line entered from the terminal; when you change from EDIT mode to INPUT mode, the pointer is positioned so that the next line entered will follow the last line edited.

After issuing the EDIT command and entering EDIT mode, the pointer is positioned at a special blank (null) line that the Editor creates and places in virtual storage in front of the first line of the file. If, during the processing of the file, a TOP subcommand is issued, the pointer is positioned at this null line. This is to allow you to insert records in front of the first data record in the file. The null line is not stored on your disk when you issue the FILE, SAVE, or AUTOSAVE subcommand. Issuing a TYPE subcommand when the pointer is positioned at the null line causes a blank line to be displayed.

LOGICAL LINE EDITING CHARACTERS

The following paragraphs describe the VM/370 logical line editing characters. In summary, the normal default values for the line editing characters are:

<u>Character</u>	<u>Meaning</u>
@	Logical character delete
#	Logical line end
¢	Logical line delete
"	Logical escape

The logical line editing characters are defined for each virtual machine during VM/370 system generation. If your terminal's keyboard lacks any of these special characters, your installation will define other special characters for logical line editing. Check with your system administrator to see which logical line editing characters have been defined for your virtual machine. Also, you can use the CP TERMINAL command to change the logical line editing characters for your virtual machine. (See the <u>VM/370</u>: <u>Command Language Guide for General Users</u>.)

If you enter three or more line editing symbols together, you may get unpredictable results.

Logical Character Delete

Should you discover that you have made a minor typing error while entering commands or data, the logical character delete symbol (@) allows you to delete one or more of the previous characters entered. The @ deletes one character per @ entered, including the ¢ and # logical editing characters. For example:

```
ABC#@@ results in AB
ABC@D results in ABD
¢@DEF results in DEF
ABC@@@ deletes the entire string
```

Logical Line End

The logical line end symbol (#) allows you to key in more than one command on the same line, to minimize the amount of time you have to wait between entering commands. You type the # at the end of each logical command line, and follow it with the next logical command line. The Editor stacks the commands and executes them in sequence. For example, the entry

```
down 1#type 1#top
```

is executed in the same way as the entries:

```
down 1
type 1
top
```

Logical Line Delete

The logical line delete symbol (¢) (or [for Teletype[1] Mod 33/35 terminals) deletes the entire previous physical line, or the last logical line back to (and including) the previous logical line end (#). It can be used to delete a line containing many or serious errors. If a # sign (logical line end) immediately precedes the ¢ sign, only the # sign is deleted. For example:

- Logical Line Delete:

  ```
  ABC#DEF¢ deletes the #DEF and results in ABC
  ABC#¢ results in ABC
  ABC#DEF¢#GHI results in ABC#GHI
  ABC#DEF¢GHI results in ABCGHI
  ```

- Physical Line Delete:

  ```
  ABC¢ deletes the whole line
  ```

Logical Escape

The logical escape symbol (") causes VM/370 to consider the next character entered to be a data character, even if it is normally one of the logical line editing symbols (@, ¢, ", or #). For example:

```
ABC"¢D results in ABC¢D
""ABC"" results in "ABC"
```

The appearance of a single logical escape symbol (") as the last character of a line is ignored.

----
[1]Trademark of the Teletype Corporation, Skokie, Illinois.

SAVING INTERMEDIATE RESULTS

When you are changing a file extensively or entering a large file, it is good practice to make a few additions or changes, issue the SAVE subcommand, and then continue entering or editing. Use of this subcommand ensures that a minimum of work would need to be redone in case of a CP or CMS system failure or a major user editing error such as globally changing the wrong word or string. Alternatively, you can invoke the AUTOSAVE subcommand at any time during the editing session to automatically save the entire file after every "n" updates.

A file must consist of at least one line of data to be written permanently on disk. A file consisting only of a null line cannot be saved, and results in an error message being displayed.

DATA FILES

The Editor examines the filetype component of the file identifier to see if it is one of the reserved filetypes whose default attributes are known to the Editor. Examples of such attributes are:

- Record format: F (Fixed) or V (Variable)
- Logical record length (LRECL)
- Case mode (uppercase or lowercase)
- Truncation columns
- Zone columns
- Serialization conditions
- Line editing conditions (LINEMODE setting)
- Verification columns
- Line image settings
- Tab settings

In general, the Editor prepares fixed-length (logical) records of 80 characters, except for SCRIPT files, which are variable-length records up to 132 characters long; LISTING files, which are 121-character variable-length records; and FREEPORT files, which are 81-character variable-length records. For all the reserved filetypes except SCRIPT and MEMO, all characters are translated to uppercase. all characters are translated to uppercase. information on reserved filetypes, refer to the _VM/370_: _Command Language Guide for General Users_.

For example, if a filetype of MEMO is specified, the file consists of 80-character fixed-length records containing both uppercase and lowercase letters, with no truncation or serialization columns.

If the filetype is SCRIPT, all input lines entered in INPUT mode and strings entered in EDIT mode are interpreted without converting lowercase characters to uppercase. The record format is set to V (variable-length records). In addition, input lines containing underscored information are internally rearranged when CANON was specified in the IMAGE command (this is the default for SCRIPT files). CANON means that regardless of how the characters are keyed in (characters, backspaces, underscores), the Editor stores the information in the file as: character-backspace-underscore, character-backspace-underscore, and so on. If, for example, you want an input line to look like:

ABC

You could enter it as:

```
     ABC, 3 backspaces, 3 underscores

               - or -

     3 underscores, 3 backspaces, ABC
```

A typewriter types out the line in the following order:

```
     A backspace, underscore
     B backspace, underscore
     C backspace, underscore, which results in:
```
<u>ABC</u>

| To enter backspace characters (X'16') at a 3270, which has no backspace
| key, you must enter some other character in place of the backspace, and
| then use the ALTER subcommand to change that character to a X'16'.

| For example, if you enter:

```
|    input ABC???___
|    alter ? 16 1 G
```

| all occurrences of the question mark on that line are changed to X'16'.
| When the changed line is redisplayed in the output display area, it
| looks like:

```
|    _ A_ B_ C
```

The backspace character is indicated by a blank, as the 3270 cannot place compound characters in a single screen position.

If you try to use a subcommand with column settings greater than the record length specified in the EDIT command (LRECL n), the Editor rejects the command. For example, a MEMO file has a fixed record length of 80, and cannot have a truncation column set beyond this limit:

```
     edit newfile memo
     NEW FILE:
     EDIT:
     trunc                the current setting is requested
      80
     trunc 129            truncation at position 129 is requested
     ?EDIT: trunc 129     the subcommand is not accepted
     trunc 81             truncation at position 81 is requested
     ?EDIT: trunc 81      the subcommand is not accepted
     trunc 60             the subcommand is accepted
     trunc                the current setting is requested
      60
```

DATA TRUNCATION

The truncation columns that are used by EDIT subcommands for reserved filetypes (and any other filetypes) are shown in Figure 3.

Truncation Column[1]	Filetype	Subcommand
71	For ASSEMBLE, UPDATE, and UPDTxxxx files	INPUT REPLACE
72	For FORTRAN, COBOL, and PLIOPT files	
81	For FREEFORT files	
Record length	All other filetypes	INPUT
End Zone	For all filetypes, the string is displayed up to the VERIFY column.	ALTER LOCATE CHANGE
[1]The truncation column may be altered at any time during the terminal session by entering a TRUNC subcommand.		

Figure 3. Truncation Columns for EDIT Subcommands

TAB SETTINGS

Logical tab settings indicate the column positions where fields within a record begin. These logical tab settings need not correspond to the physical tab settings on a typewriter terminal. Tab operations using a display terminal are discussed under "Inserting Tab Characters at a 3270." The default logical tab settings depend on the filetype of the file. Changing the default logical tab settings is discussed below.

What happens when you press the TAB key on a typewriter terminal depends on whether IMAGE is on or off. (The default for all filetypes except SCRIPT is IMAGE ON. You can change the default by issuing the EDIT subcommand IMAGE ON or IMAGE OFF.)

If IMAGE is on when you press the TAB key, the Editor inserts one or more blanks into the record, starting at the column where you pressed the TAB key, and ending at the last column before the next logical tab setting. The next character entered after the tab becomes the first character of the next field.

If IMAGE is off when you press the TAB key, the tab character, X'05', is inserted in the record, just as any other data character is inserted. No blanks are inserted.

Default Tab Settings

The Editor places data entered under the following filetypes in the proper column when the tabbing is performed.

Filetype	Default Tab Settings
ASSEMBLE, MACRO, UPDATE, UPDTxxxx, ASM3705	1, 10, 16, 31, 36, 41, 46, 69, 72, 80
FORTRAN	1, 7, 10, 15, 20, 25, 30, 80
FREEFORT	9, 15, 18, 23, 28, 33, 38, 81
BASIC, VSBASIC	7, 10, 15, 20, 25, 30, 80
PLIOPT, PLI	2, 4, 7, 10, 13, 16, 19, 22, 25, 31, 37, 43, 49, 55, 79, 80
COBOL	1, 8, 12, 20, 28, 36, 44, 68, 72, 80
All Others	1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 61, 71, 81, 91, 101, 111, 121, 131

To change the default settings, use the EDIT subcommand TABSET.

Inserting Tab Characters at a Typewriter Terminal

If you want to insert a tab character (X'05') into a record and IMAGE is on, you can do one of the following:

1. Set IMAGE OFF before you enter or edit the record, and then press the TAB key at the appropriate place in the record; or

2. Press some other character key at the appropriate place in the record, and then use the ALTER subcommand to alter that character to a X'05' (tab character). You should use a seldom-used character, such as ? or ¬ (not) for this purpose.

Inserting Tab Characters at a 3270

To enter a tab character at a 3270, which does not have a TAB key, you can either:

1. Enter another character at the appropriate place in the record, and then use the ALTER subcommand to alter that character to a X'05', or

2. Before you begin to create the file, use the CMS command SET INPUT char 05 to define some other character as the tab character. CMS will then translate all occurrences of that character to X'05' and redisplay the input line in the output display area. If IMAGE is on, the line will appear padded with an appropriate number of blanks.

3. Define a 3270 program function key to indicate tab settings.

Defining a 3270 Program Function Key for Tab Settings

Use the CP "SET PFnn TAB" command in the following manner:

```
SET PFnn TAB n1 n2 . . . nn
```

where:

nn	is any valid function key number from 1 to 12.
n1 n2 . . . nn	are the logical tab settings desired, expressed as decimal numbers. Valid numbers are 1 to 136. Invalid tab settings are ignored. You can specify the setting values in any order, but they are normally specified in ascending order.

You can define different PF keys with different tab settings for different filetypes. The operation is always executed immediately regardless of the specification of the IMMED or DELAY operand with the operation.

The operation works in the following manner when you press the PF key. CP examines the tab settings and sets them in ascending order. CP ignores invalid values and examines the data at the current cursor position on the screen. If it is "null" it is replaced in storage by a tab character, but on the screen the cursor is positioned at the next tab setting in the user input area. CP does not replace valid characters in the input data. Tabbing beyond the last defined tab position repositions the cursor at the beginning of the user input area.

When you end the data line by pressing the ENTER key, the data with the imbedded tab character (X'05') is presented to the Editor for processing.

OPERANDS CONTAINING CHARACTER STRINGS

Some EDIT subcommands require character strings as operands. A string begins with a delimiter (any valid character that does not appear in the string) and continues as a sequence of valid characters until the delimiter is again specified. Thus, the delimiter indicates only the beginning and end of the string. The delimiter is redefined in each new subcommand by its appearance at the beginning of a string. If two strings exist in one subcommand, the same delimiting character must be used for each string. In this publication, the slash (/) is used to denote the delimiter, although any valid character that does not appear in the string can be used.

Note: Unless you turn off logical editing via the CP TERMINAL command, VM/370 edits the input line for character delete, line delete, logical escape, and logical line end symbols before the Editor receives the character string. For a discussion of input line editing, refer to the previous section entitled "Logical Line Editing Characters."

SERIALIZATION OF RECORDS

If record serialization is the default for the filetype being edited, or if you specify serialization by issuing a SERIAL subcommand, an identifier is normally placed in columns 73-80 of each record. (For BASIC and VSBASIC files, a 5-digit sequence number is placed in position 1-5 of each record.) The identifier normally consists of a three-character identification followed by a five-digit number.

The identifier is taken from the first three characters of the filename, or from the first operand of the SERIAL subcommand. The sequence numbering normally begins at 00010 and is increased by 10 for each record unless it is changed by the PROMPT or SERIAL subcommand. A PROMPT subcommand simply changes the prompting increment. The line identifier in columns 73-80 can be changed by issuing the SERIAL subcommand when in EDIT mode.

Serialization is the default for the following filetypes: ASSEMBLE, MACRO, FORTRAN, COBOL, PLIOPT, UPDATE, and UPDTxxxx. Serialization is suppressed, unless requested by the SERIAL subcommand, for all other filetypes. A file with a filetype of MEMO, LISTING, or EXEC must not be serialized, since each column in records in these types of files may contain data. Serialization can only be used on fixed format, 80-character record length files.

CONTEXT EDITING

You would normally use context editing on files that are not line number oriented or if changes are to be made globally throughout the file. It allows you to create and update files by content rather than by line number. The following discussion introduces the EDIT subcommands that are useful in "Section 3: EDIT Subcommands."

EDIT Subcommands Used for Context Editing

You can use the following EDIT subcommands when context editing:

AUTOsave [n|OFF]

| Tells the Editor to automatically save the file after every _n_
| updates to the file.

BAckward [_1_|n]

Moves records down the screen n lines. This subcommand is functionally equivalent to the UP subcommand. The current line pointer moves within the file, but the current line remains positioned at line 9.

Bottom

Moves the line pointer to the last record of the file.

Change /string1/string2[/[_1_|n|*[G|*]]]

Changes the old data (string1) to the new data (string2).

DELete [1|n|*]

Deletes a single record, a group of consecutive records, or all records in a file from the current line pointer to the end of the file.

DOwn [1|n]

Moves the line pointer down the specified number of lines. This subcommand is functionally equivalent to FORWARD and NEXT.

| DString /string[/]

| Deletes the lines from the current line down to, but not including,
| the first line containing the specified character string.

FILE [filename [filetype [filemode]]]

Stores the current file on your disk and returns to CMS, terminating the EDIT session.

Find [line]

Searches the beginning of each line in the file, between the current line and the end of the file, for the first occurrence of the specified data line. If no line is specified, the search is successful on the first line examined.

FOrward [1|n]

Moves records up the screen n lines. This subcommand is functionally equivalent to DOWN and NEXT.

Input [line]

Places the specified line in the file while in EDIT mode. If no data line is specified, INPUT mode is entered.

Locate /string[/]

Scans each record in the file, between the current line and the end of the file, for the first occurrence of the specified string.

Next [1|n]

Moves the line pointer down the specified number of lines. This subcommand is functionally equivalent to FORWARD and DOWN.

QUIT

Terminates the EDIT session and returns to CMS, but does not save the current file contents on your disk.

Replace [line]

Replaces the current line with the specified line. If no data line is specified, INPUT mode is entered and the current line is deleted.

SAVE [filename [filetype [filemode]]]

Saves the current file on your disk. If a file with the same filename already exists on your disk, it is replaced by the current file.

S[croll][Up][*|n|1]

Displays a file in increments of 10 or 20 records at a time. SCROLL causes forward scrolling; SCROLLUP causes backward scrolling. This subcommand is valid only for a display terminal in DISPLAY mode.

TOP

Moves the line pointer to the null line at the top of the file.

Type [1|m|*[n|*]]

Displays the specified lines of the current file at the terminal.

Up [1|n]

Moves the line pointer up the specified number of lines. This subcommand is functionally equivalent to BACKWARD.

Creating and Changing a File by Context

The following example shows the creation of a new file at a typewriter terminal, then its alterations, and finally the replacement of the original file by the updated file. The subcommands used are illustrative only and are not intended to show the optimum way to achieve a particular result. User entries are in lowercase; computer responses are in uppercase.

ipl cms	Although not shown, each line is terminated with pressing the Return key, which causes the end of input line. (See "Section 2: Command Modes," for a discussion on ending an input line.)
CMS VERSION n LEVEL n	
edit dalow memo NEW FILE: EDIT: input INPUT:	EDIT command is issued, and a system file search reveals that no CMS file with that name and type exists. To enter INPUT mode, input is typed.
This file has been created to shooow of the edit command. But it is not	These four lines were created by keyed input. This is the file data.
	A null line (press RETURN) entered here causes the system to leave INPUT mode and enter EDIT mode.
EDIT:	The EDIT mode allows the use of the EDIT subcommands.
file	The 4-line file named DALOW MEMO, created above, is stored on disk.

R; T=0.12/0.33 16:30:42	Acknowledgment by CMS that the FILE subcommand completed successfully.
edit dalow memo	The file is requested again so that corrections can be made.
EDIT:	The file exists, thus the Editor knows that subsequent actions will involve alterations via the EDIT subcommands.
type 4	The TYPE subcommand is invoked to type four lines.
This file has been created to shooow of the edit command.	The Editor responds by typing the null line at the beginning of the file, plus the next 3 data lines. <u>Note</u>: The pointer now points to the last line typed (that is, the "of the edit command" line).
up 2	Move the pointer up 2 lines.
This file has been	The Editor responds by typing the desired line.
down	Point to next line. Since no operand is specified, 1 is assumed.
created to shooow	The Editor responds by typing the next line.
change /shooow/show/	The CHANGE subcommand is issued to correct the spelling of shooow.
created to show	The Editor responds by typing the corrected line.
input the power and versatility	The INPUT subcommand is used to insert a new line following the line just changed.
bottom	The BOTTOM subcommand is issued.
But it is not	The Editor responds by typing the last line of the file.
delete	DELETE subcommand deletes the line.
EOF:	The Editor indicates that the current line pointer is at the end of the file.
top	The TOP subcommand requests the line pointer to be positioned at the beginning of the file.
TOF:	The Editor indicates that the pointer is at the null line at the top of the file.
type 10	The TYPE subcommand requests the Editor to type 10 lines (or all that are in the file if less than 10).

```
This file has been
created to show
the power and versatility
of the edit command.
EOF:
file



R; T=0.22/0.96  16:30:55
```

You see that all the changes have been made.

When you have verified the changes, use the FILE subcommand to store the edited file on your disk. (The old DALOW MEMO file is erased, and the new one replaces it.)

LINE NUMBER EDITING

You may find it more convenient to use line numbers when creating or updating certain types of data files, such as those containing BASIC or VSBASIC source statements. Line number references are used to insert, replace, or call out data records explicitly by number, rather than by relative displacement from the current position of the line pointer.

EDIT Subcommands Used for Line Number Editing

You can use the following EDIT subcommands when editing by line numbers:

Input

You cannot specify a data line operand with the INPUT subcommand when LINEMODE LEFT or RIGHT has been specified. The nnnnn [text] subcommand must be used instead. Also, when in INPUT mode, line numbers are displayed at the beginning of each line as the Editor prompts for the next input line. On a display terminal, however, the prompting numbers are displayed in the message line near the top of the screen.

LINEmode [Left|Right|OFF]

The LEFT or RIGHT operands place line numbers at the beginning (LEFT) or end (RIGHT) of each data record.

nnnnn [text]

Use this subcommand while in EDIT mode in conjunction with or instead of the INPUT subcommand when LINEMODE LEFT or RIGHT has been specified.

PROMPT [n]

Changes the prompting increment by which line numbers are incremented. It is only applicable in INPUT mode when LINEMODE LEFT or RIGHT has been specified.

RENum [strtno|<u>10</u> [incrno|<u>strtno</u>]]

RENUM recomputes all the line numbers in a VSBASIC or FREEFORT file. Also, for all VSBASIC statements that have statement numbers for operands, those operands are recomputed so that they correspond to the new line numbers.

Creating and Changing a File by Line Numbers

The following example illustrates the creation of a BASIC language source file at a typewriter terminal. User entries are lowercase; system responses are uppercase. Although not shown, each line is terminated by pressing the RETURN key, causing the end of input. (See "Section 2: Command Modes," for a discussion of ending an input line.)

```
ipl cms
CMS VERSION n LEVEL n
```

The user loads CMS.

```
e total basic
NEW FILE:
EDIT:
```

The minimum acceptable form of the EDIT command is issued, and a file search reveals that no CMS file with that filename and filetype exists. Because the filetype is BASIC, LINEMODE LEFT is automatically set by the Editor.

```
input
INPUT:
```

The user enters INPUT mode by typing "input".

```
   10 a=0
   20 c=3
   30 d= b+a*c
   40

EDIT:
25 b=10
```

The Editor prompts each new data line with line numbers (10, 20, etc.), and the user enters data (a=0, c=3, etc.). Then the user issues a null line to get out of INPUT mode. The user enters EDIT mode, then inserts line 25, which he had forgotten to enter previously.

```
5
```

He then checks to see if line 5 exists.

```
LINE NOT FOUND
TOF:
```

It does not; the pointer is at top of file.

```
5 rem  sample basic program

30
   30 D= B+A*C
change /c/c+1/
   30 D= B+A*C+1
```

The user then inserts the forgotten line at line 5, and calls out line 30. The Editor types line 30 as it appears in the file. The user changes the line and the Editor types the line as it now appears.

```
input
INPUT:
   40 t=d+t
```

The user enters INPUT mode to continue inserting lines in the file.

```
   50
EDIT:
```

A null line response returns the user to EDIT mode.

```
26 t=0
```

The user remembers to initialize a value and inserts the line.

```
bottom
   40 T=D+T
50 end
```

He then goes to the bottom of the file to insert the last line in the file. The Editor types out the last record in the file, and the user types in line 50.

```
top
TOF:
type *

    5 REM  SAMPLE BASIC PROGRAM
   10 A=0
   20 C=3
   25 B=10
   26 T=0
   30 D= B+A*C+1
   40 T= D+T
   50 END
EOF:
```

The TOP subcommand positions the pointer at the top of the file. At this point, the user requests that all lines of the file be typed.

```
file
R; T=0.34/1.37  16:33:34
```

Since the results are satisfactory, he stores the program by issuing the FILE subcommand.

Creating and Changing an Assembler File by Line Numbers

The following example illustrates the creation of an assembler language file that uses CMS macros. (Note that the program created does not do any useful work.)

The example as shown assumes that the user has set physical tabs to positions 16 and 22. Since the file being created has a filetype of ASSEMBLE, the Editor automatically sets the logical tabs needed for an assembler language file (1, 10, 16, etc.) This is discussed under "Tab Settings."

```
e test assemble
NEW FILE:
EDIT:
```

After the user loads CMS, he enters the EDIT command. Because the specified file did not previously exist, the Editor types out NEW FILE:.

```
linemode right
```

The user issues the LINEMODE RIGHT subcommand to number the new file.

```
input
INPUT:
00010 * sample of linemode right
00020 test     start x'0'
00030          balr  12,0
00040          using *,12
00050          st    14,sav14
00060          type  'testing...'
00070          l     14,sav14
00080          br    14
00090          end
```

The user then enters INPUT mode to insert lines into the file. Note: The line numbers (with leading zeros) are typed out on the left, even though LINEMODE RIGHT has been specified. The Editor performs this transposition to save output typing time and to make it easier to refer to the line numbers.

```
00100
```

A null line keyed in causes a transfer to EDIT mode.

```
EDIT:
20
00020 TEST     START X'0'
change /0/20000/
00020 TEST     START X'20000'
80
00080          BR    14
input
```

The user calls out line 20 by number, and the Editor types out line 20. The modification is made and then typed out. The user then requests that line 80 be typed, then goes to INPUT mode to insert the program data areas after line 80.

```
INPUT:
00083 sav14    ds    f
00085 wkarea   ds    3d
00087 flag     ds    x
00088 runon    equ   x'80'
00089 runoff   equ   x'40'
```

The Editor uses a predetermined algorithm to insert new lines between the old lines.

```
RENUMBER LINES
```

At this point, the Editor runs out of available line numbers since record number 00090 already exists.

```
EDIT:
line off
serial all 10
save
EDIT:
line right
```

The user then follows this procedure (valid for LINEMODE RIGHT only) to reserialize the line numbers. The '10' could be any reasonable increment. The line numbers cannot be greater than five digits long. A FILE or SAVE subcommand must be invoked to cause reserialization to occur in storage and on disk. After serialization, the LINEMODE RIGHT subcommand is reissued.

```
type
```

A typeout of the last record entered is requested.

```
00130 RUNOFF   EQU   X'40'
```

(Line 130 was previously numbered line 89.)

```
| 135 runmix     equ    x'20'        The nnnnn subcommand is  used to enter
| 50                                 line 135 and to move  the line pointer
| 00050          ST     14,SAV14     to line 50.
| input                              The user issues  the INPUT  subcommand
| INPUT:                             and the Editor  prompts him  for lines
  00053          tm     flag,runon   53, 55, and 57.
  00055          bcr    1,14
  00057                              A null line  response is  used to  get
  EDIT:                              out of INPUT mode.
  top                                The user  repositions the  pointer  to
  TOF:                               the beginning of the file and requests
  type *                             a typeout of the file.
  00010 * SAMPLE OF LINEMODE RIGHT
  00020 TEST     START X'20000'
  00030          BALR  12,0
  00040          USING *,12
  00050          ST     14,SAV14
  00053          TM     FLAG,RUNON
  00055          BCR    1,14
  00060          TYPE  'TESTING...'
  00070          L      14,SAV14
  00080          BR     14
  00090 SAV14    DS     F
  00100 WKAREA   DS     3D
  00110 FLAG     DS     X
  00120 RUNON    EQU    X'80'
  00130 RUNOFF   EQU    X'40'
  00135 RUNMIX   EQU    X'20'
  00140          END
  EOF:
  file                               Issuing the FILE subcommand causes the
                                     file to be stored on the user's disk.
  RESERIALIZATION SUPPRESSED         The Editor  did  not  reserialize  the
  R;                                 file again; it remains as it was since
                                     the file was previously serialized due
                                     to the previous SAVE subcommand.
```

The user can obtain a typeout of this file as it appears on disk under CMS by keying in:

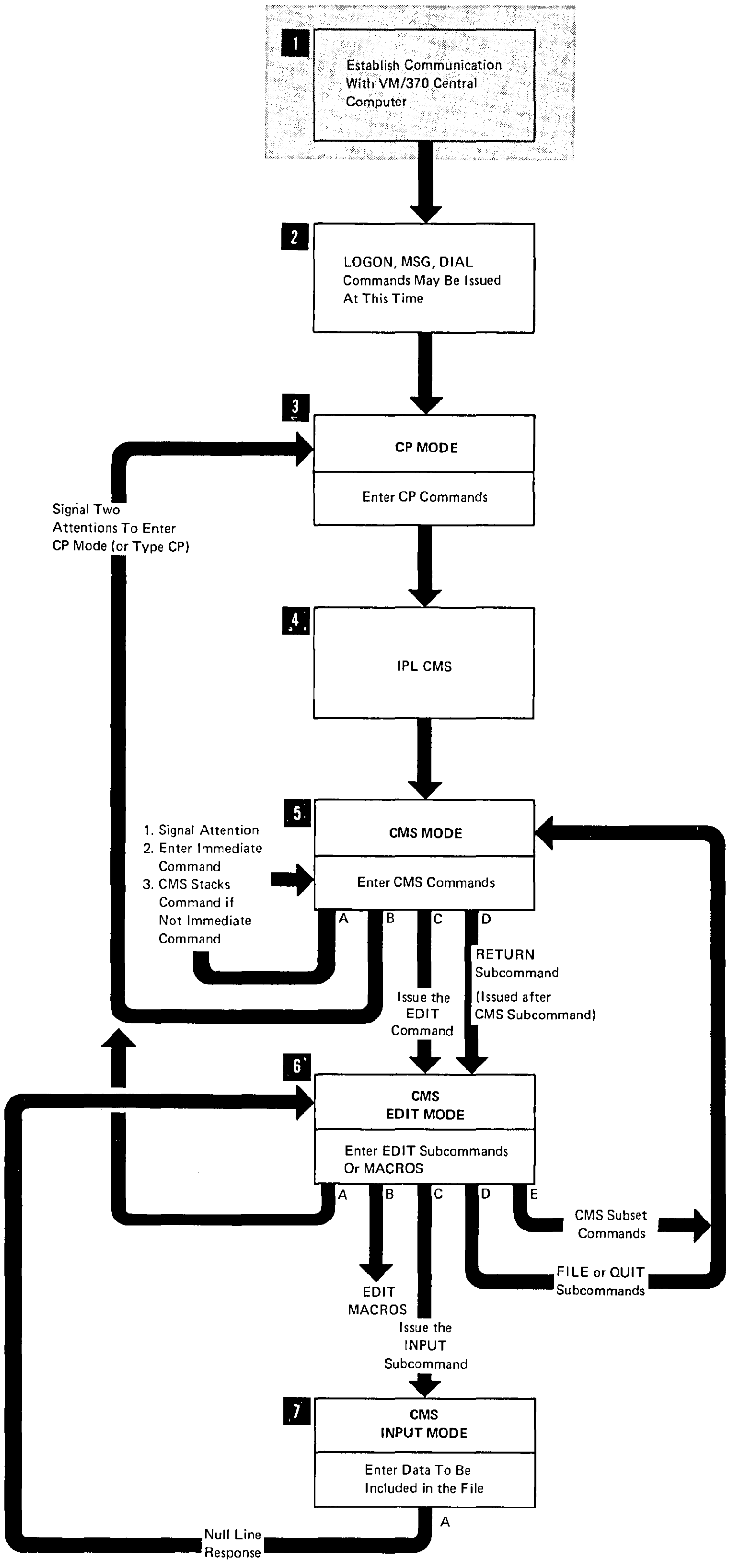
```
type test assemble

* SAMPLE OF LINEMODE RIGHT                                           00000010
TEST     START X'20000'                                              00000020
         BALR  12,0                                                  00000030
         USING *,12                                                  00000040
         ST    14,SAV14                                              00000050
         TM    FLAG,RUNON                                            00000053
         BCR   1,14                                                  00000055
         TYPE  'TESTING...'                                          00000060
         L     14,SAV14                                              00000070
         BR    14                                                    00000080
SAV14    DS    F                                                     00000090
WKAREA   DS    3D                                                    00000100
FLAG     DS    X                                                     00000110
RUNON    EQU   X'80'                                                 00000120
RUNOFF   EQU   X'40'                                                 00000130
RUNMIX   EQU   X'20'                                                 00000135
         END                                                         00000140

R;
```

Observe that the record numbers now appear where they actually occur in the records--in positions 73 through 80.

Section 2: Command Modes

This section shows you how to enter into and exit from the various VM/370 command modes.

Throughout this section, a diagram guides you in the discussion of each command. The diagram faces each page of text, with the topics discussed highlighted on the diagram. The numbered and lettered portions of the diagram correspond to the numbered and lettered portions of the text.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
4
IPL CMS
5
CMS MODE
Enter CMS Commands
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand
(Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
EDIT MACROS
Issue the INPUT Subcommand
CMS Subset Commands
FILE or QUIT Subcommands
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

1 VM/370 LOGON PROCEDURES

Depending on the type of terminal you are using, you need to turn the terminal on and either:

- Perform the LOGON procedure if your terminal is connected to the central computer by a leased line or local attachment.
- Dial the central computer's telephone number, if the terminal is connected to the central computer by a switched line. Then perform the LOGON procedure.

For additional information on establishing communication with the VM/370 system, consult the *VM/370: Terminal User's Guide*.

The terminal used in this example is a typewriter terminal.

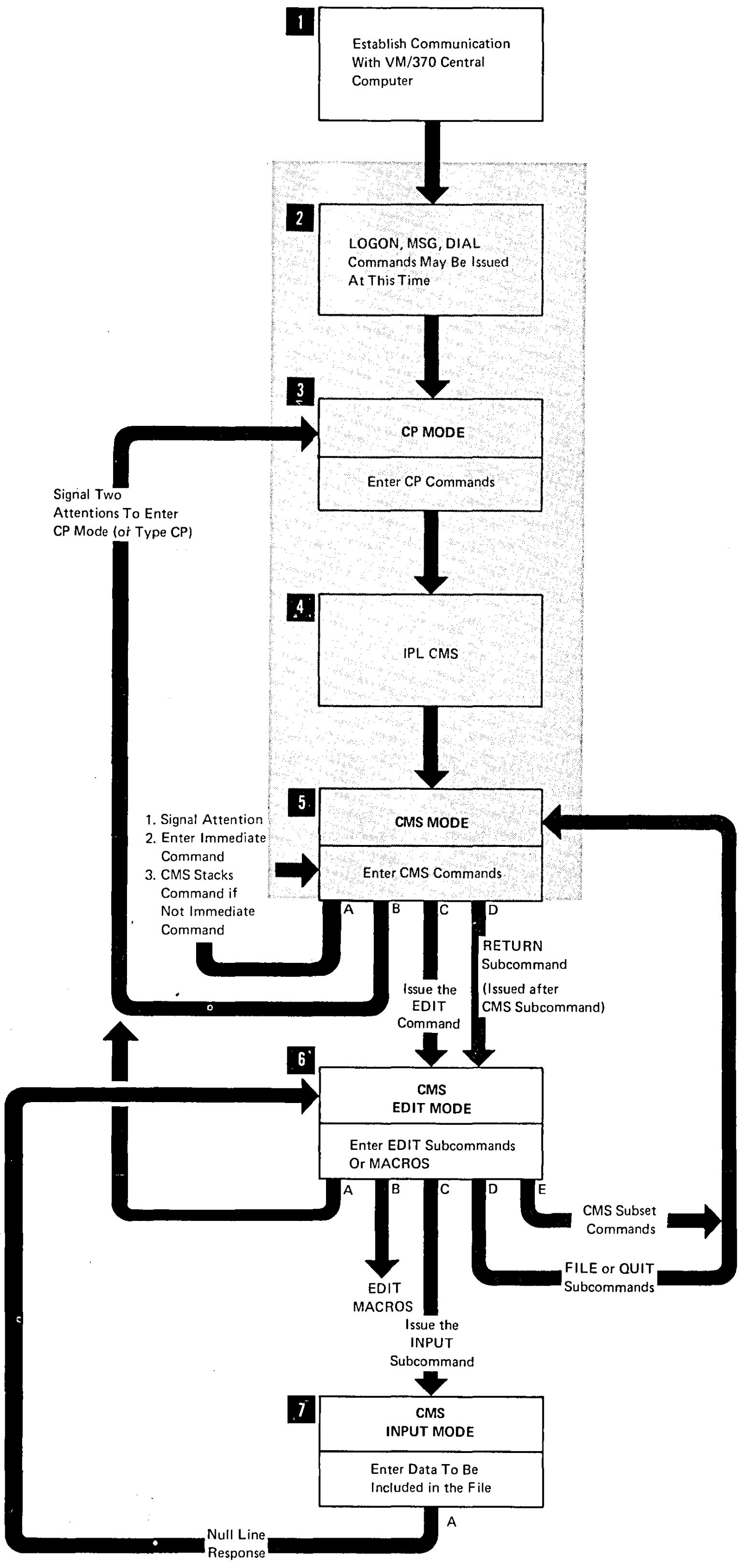
SIGNALLING ATTENTION

Machine Type	Key Used
2741,3767	ATTN
1052	RESET LINE or ATTN[1]
Teletype Models 33,35	BREAK
3210,3215	REQUEST
3158 Console	ENTER
3270	ENTER, PA1
3066 Console	ENTER

ENDING AN INPUT LINE

Machine Type	Key Used
2741,3767	RETURN
1052	RETURN[2] or ALTN CODING plus numeral 5 (EOB)
Teletype Models 33,35	CTRL plus X-OFF (or with the appropriate terminal control unit feature on the computer, RETURN)
3210,3215	END
3158 Console	ENTER
3270	ENTER
3066 Console	ENTER

----
[1]Will be ATTN if feature 1600 is present.
[2]Requires an RPQ Feature.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
4
IPL CMS
5
CMS MODE
Enter CMS Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand
(Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
CMS Subset Commands
FILE or QUIT Subcommands
EDIT MACROS
Issue the INPUT Subcommand
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

2 LOGON, MSG COMMANDS

Once communication with the VM/370 system has been established, the normal procedure is to log on via the LOGON command. If unable to log on, use the MSG command to communicate with the VM/370 computer operator or another user who is logged on with your userid.

3 CP MODE

On completion of the logon procedure, the CP command environment is entered as indicated by the message:

```
LOGON AT 10:57:54 EST FRIDAY 02/09/73
```

4 IPL CMS

At this point, you can load CMS or any other supported operating system
| into the virtual machine by issuing the CP command IPL, if this was not
| automatically done for you.

5 CMS MODE

When CMS is loaded into the virtual machine, a message similar to the following types:

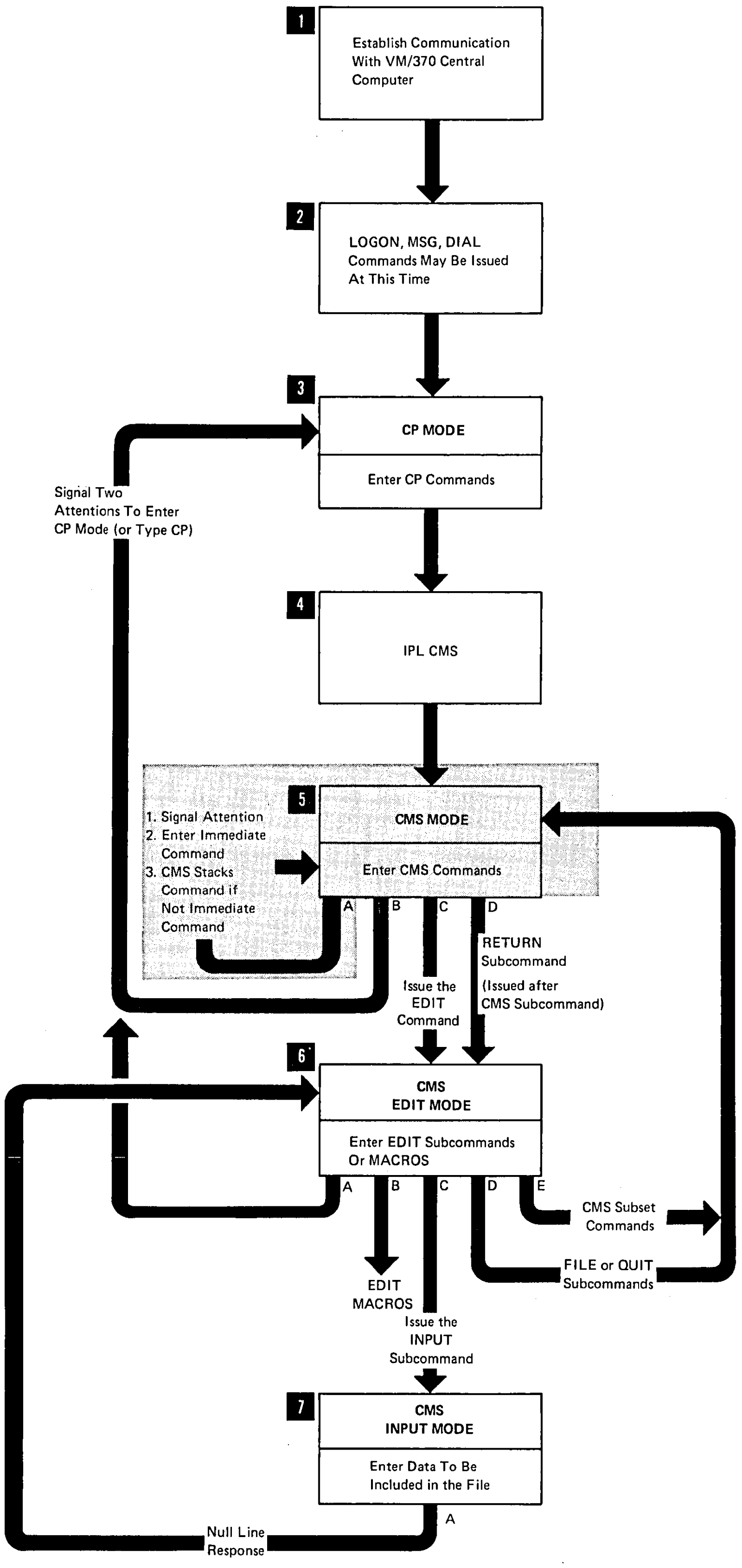
```
CMS 02/02/73 FRI 08.12.38
```

FILE IDENTIFIERS

Each file created under CMS control, or created for use via CMS commands, is represented on disk by a file identifier. The file identifier consists of three components--filename, filetype, and filemode, in the following format:

filename	filetype	filemode

filename	is a one- to eight-character alphameric name assigned (in most cases) by you.
filetype	is a one- to eight-character alphameric name used as a descriptor or as a qualifier of the filename. Refer to the _VM/370: Command Language Guide for General Users_ for a list of the assumed or default attributes of reserved filetypes.
filemode	is a two-character field. The first character, the mode letter, is alphabetic and denotes the name of the file directory of the disk on which the file resides. The second character, the mode number, is numeric and denotes the mode by which the file is accessed. Valid mode numbers are shown in Figure 4.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
4
IPL CMS
5
CMS MODE
Enter CMS Commands
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand
(Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
CMS Subset Commands
FILE or QUIT Subcommands
EDIT MACROS
Issue the INPUT Subcommand
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

Filemode Number	Read/Write Status	Meaning
0	R/W	The file specified is a private file; you cannot access a file with the 0 filemode unless you have read/write privileges for the virtual disk on which the file resides.
1	R/W	You can read from and write on this file, depending on how the disk is accessed.
2	R/W	You can read from and write on this file, depending on how the disk is accessed. Certain files on the S-disk are mode 2; you can access these files. You can also use mode 2 to describe files on disks other than the system disk.
3	R/E	The file is to be erased after it is read. Usually, this filemode is used for temporary work files created by the language processors and some CMS commands.
4	OS	This file is created using OS macros. It may be blocked and, if in OS variable format, may contain Block Descriptor Words (BDWs) and Record Descriptor Words (RDWs).
5	R/W	Has the same meaning as filemode 1.
6–9		Reserved for IBM use.

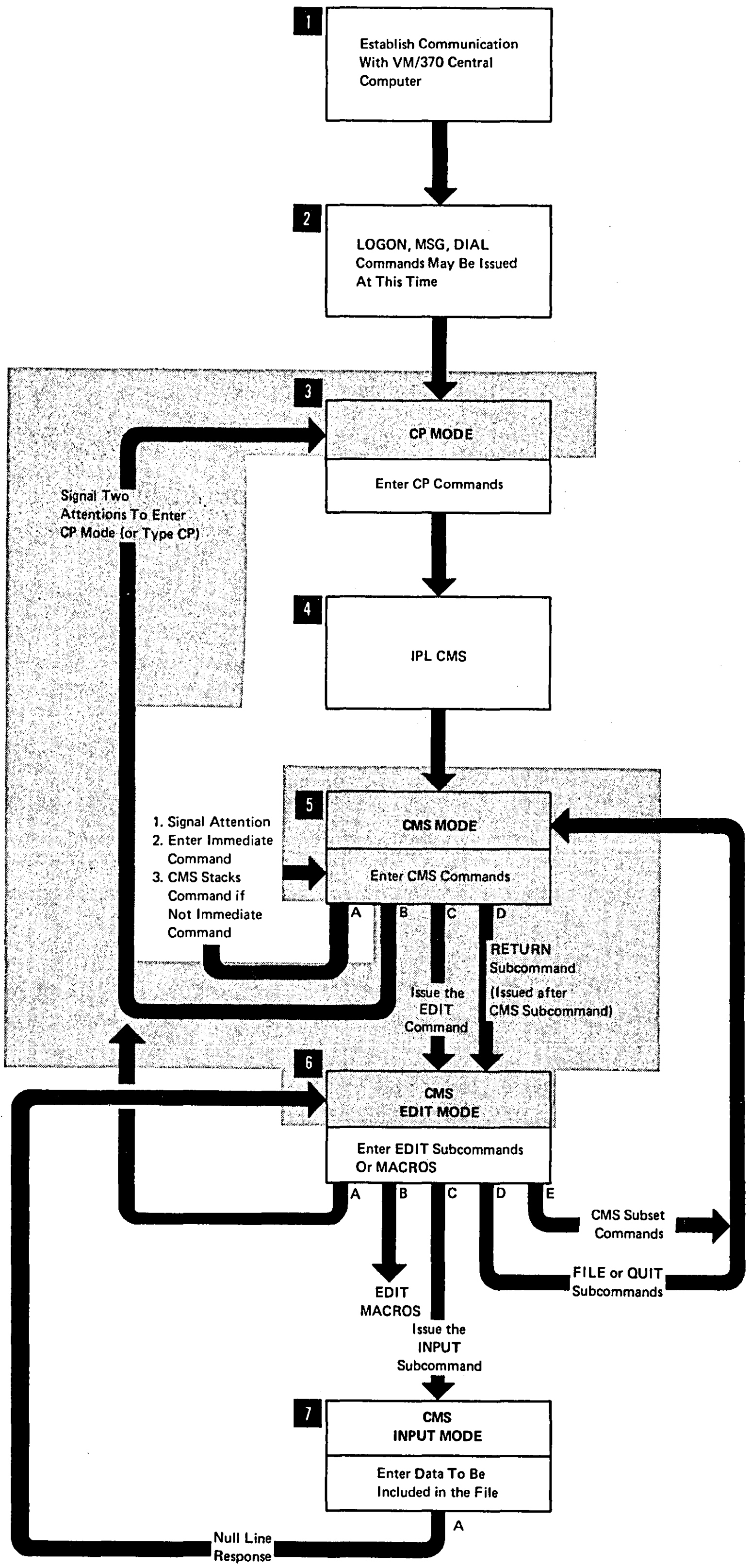
Figure 4. Access Modes for CMS Files

5A CMS IMMEDIATE COMMANDS

You can enter one of the following CMS commands only after signalling an Attention interruption (pressing the ATTN key, ENTER key, or equivalent) during program execution. (Refer to the description of the TERMINAL MODE command in the _VM/370: Terminal User's Guide_ to see how Attention handling can be defined differently for different users.) CMS reads and acts on the command immediately if it is one of the following:

Command	Action
HB	Halt Batch. This stops the execution of a CMS Batch virtual machine after the completion of the current job.
SO	Suspend Tracing. This causes the recording of trace information to be temporarily suspended.
RO	Resume Tracing. This causes tracing that was suspended by the SO command to resume.
HO	Halt Tracing. This causes the recording of trace information to stop.
HT	Halt Typing. This causes the output typing or displaying to stop, but the program continues execution.
RT	Restore Typing. This causes typing or displaying that was stopped by an HT command to resume at a later point in the program.
HX	Halt Execution. This causes execution of the previous CMS command or user program to terminate immediately.

If the CMS command is not one of the above, CMS "stacks" the command or commands in the terminal input buffer (also called the console stack) for processing after execution of the program-in-progress has been completed.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
4
IPL CMS
5
CMS MODE
Enter CMS Commands
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand (Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
CMS Subset Commands
FILE or QUIT Subcommands
EDIT MACROS
Issue the INPUT Subcommand
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

5B RETURNING TO CP (FROM CMS)

| If you want to return to the CP environment, type CP or #CP or signal attention the appropriate number of times. On a 3270 terminal, press the PA1 key. (To return to the CMS environment from the CP environment,
| enter the CP command BEGIN.) For further information on attention
| signalling, see the TERMINAL MODE command description in the *VM/370:*
| *Terminal User's Guide*.

5C ENTERING EDIT MODE

The EDIT command invokes the CMS Editor. In EDIT mode, EDIT subcommands and macros may be issued.

5D ENTERING EDIT MODE VIA THE RETURN COMMAND

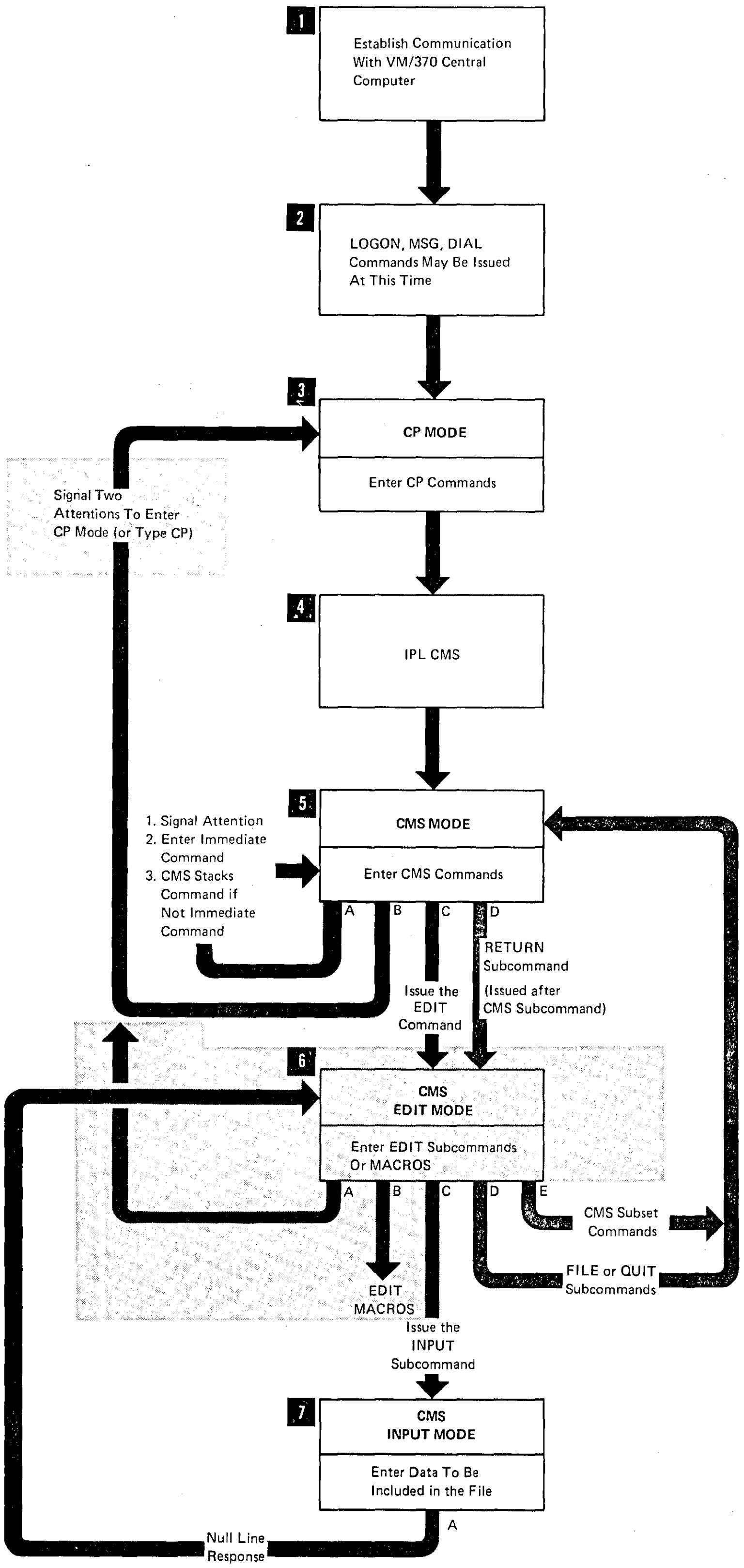
If you got into CMS mode from EDIT mode by issuing the EDIT subcommand CMS, you can return to EDIT mode by issuing the CMS subset command RETURN. Paragraph 6E lists the other CMS subset commands that can be issued.

6 *CMS EDIT MODE*

Editing is performed upon a copy of the CMS file that is completely resident in the user's virtual storage. This approach allows rapid searching both forward and backward through the file. It does, however, limit the size of files to be edited to those that can be whclly contained within the user's available virtual storage. It is possible to perform editing operations upon larger files by splitting the file into smaller files that can be handled by the Editor in the available storage area, or by temporarily increasing the size of the virtual machine by use of the CP command DEFINE. Using DEFINE, you can increase your storage up to the maximum permitted for your virtual machine (that is, the MSTOR value in the VM/370 directory entry) for the duration of the terminal session.

EDIT mode is indicated by either of two messages:

Message	*Meaning*
EDIT:	Indicates that the file specified by the CMS EDIT command has been loaded into the virtual machine; on a typewriter terminal, the message is followed by a carriage return and the unlocking of the keyboard. On a display terminal, the message is displayed in the message line (Line 1). Since the 3270 keyboard is always unlocked, you can start entering data immediately. You can then make changes to a file by issuing EDIT subcommands and macros. All changes to the file become effective immediately in main storage, but not on disk. The changed file must be written to your disk storage with the FILE, SAVE, or AUTOSAVE subcommands if you want to keep a permanent copy of the changed file. If you issue a QUIT subcommand, any changes to the file in storage are ignored, and the original file is kept on disk as it existed before any changes were made. If the file is newly created, it is not written onto disk.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
4
IPL CMS
5
CMS MODE
Enter CMS Commands
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand
(Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
EDIT MACROS
CMS Subset Commands
FILE or QUIT Subcommands
Issue the INPUT Subcommand
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

Message	Meaning
NEW FILE: EDIT:	Indicates that the file specified by the CMS EDIT command does not presently exist. To enter INPUT mode, enter the EDIT subcommand INPUT.

The INPUT mode is indicated by the message

```
INPUT:
```

a carriage return, and the unlocking of the keyboard. You can then type successive lines of input to the file. To insert a blank line in a file, type at least one space and then press the RETURN key (or equivalent end-of-line key). A null line (that is, a carriage return with no prior blanks or characters) entered when in INPUT mode does not add a blank line to the file; instead it is a signal to the Editor that you have finished keying in input data, and the Editor returns to EDIT mode.

If a null line is entered while in EDIT mode, the following message is displayed:

```
EDIT:
```

Note: A null line is a terminal input line consisting of a carriage return or line-end character as the only information in the last logical line. The line-end character brings you back to EDIT mode; thus it has the same effect as a null line. Verification, invoked by the VERIFY ON subcommand, is the normal response mode of the Editor. In this mode, the Editor displays, at the terminal, each line changed or found. If you turn verification off, only normal Editor messages are displayed. To display the line at the terminal, you must issue the TYPE subcommand.

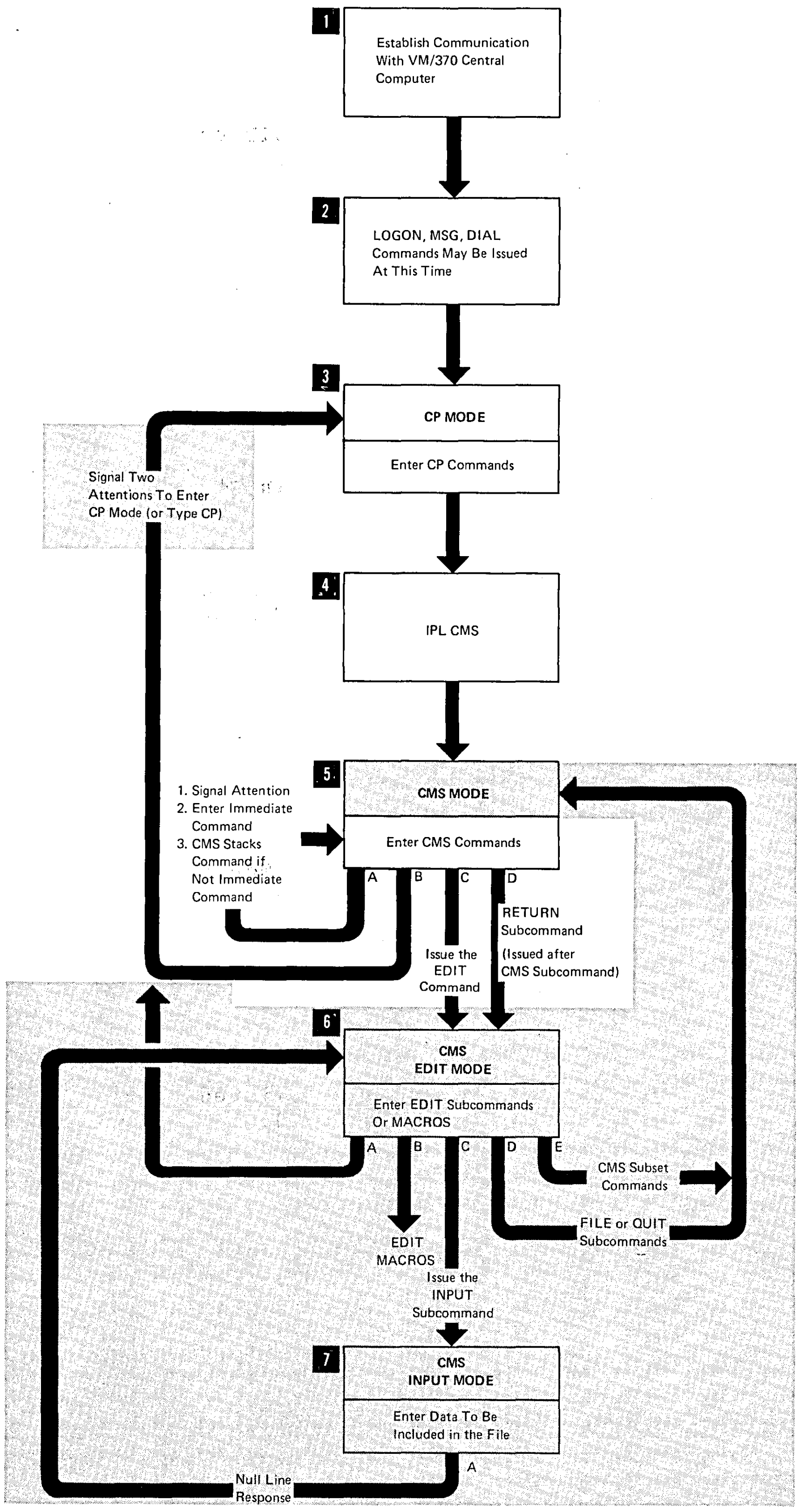
6A RETURNING TO CP (FROM EDIT)

| If you want to return to CP, key in #CP or signal attention the appropriate number of times. If you are using a 3270 display terminal, you may press the PA1 key instead. (To return to EDIT mode, issue the CP command BEGIN.)

You can also enter CP by issuing a QUIT or FILE subcommand, which | returns control to CMS, then issuing the CP or #CP command (without any operands).

6B ENTERING EDIT MACROS

You can issue EDIT macros when the Editor is in EDIT mode. These are special procedures that allow you to manipulate and edit files. If issued in INPUT mode, they are considered to be normal data input to the file. See "Section 4: EDIT Macros" for additional information.


1
Establish Communication With VM/370 Central Computer
2
LOGON, MSG, DIAL Commands May Be Issued At This Time
3
CP MODE
Enter CP Commands
Signal Two Attentions To Enter CP Mode (or Type CP)
4
IPL CMS
5
CMS MODE
Enter CMS Commands
1. Signal Attention
2. Enter Immediate Command
3. CMS Stacks Command if Not Immediate Command
A
B
C
D
Issue the EDIT Command
RETURN Subcommand
(Issued after CMS Subcommand)
6
CMS EDIT MODE
Enter EDIT Subcommands Or MACROS
A
B
C
D
E
CMS Subset Commands
FILE or QUIT Subcommands
EDIT MACROS
Issue the INPUT Subcommand
7
CMS INPUT MODE
Enter Data To Be Included in the File
A
Null Line Response

|6C| ENTERING INPUT MODE

Whether or not the file specified in the EDIT command exists, EDIT mode is entered. To make changes to a file, type:

```
     input
```

and the Editor responds with:

```
     INPUT:
```

New data can then be entered.

|6D| ENTERING CMS VIA FILE OR QUIT SUBCOMMANDS

Issuing the subcommands FILE or QUIT causes a return to the environment from which the Editor was initiated. FILE stores the current file on disk, whereas QUIT does not.

|6E| CMS SUBSET COMMANDS

While in EDIT mode, a subset of CMS commands may be issued after specifying the EDIT subcommand CMS. They are:

```
     ACCESS   LISTFILE   RENAME
     CP       PRINT      RETURN
     DISK     PUNCH      SET
     ERASE    QUERY      STATE
     EXEC     READCARD   TYPE
```

Note: If CMS commands other than those listed are issued, an error message is typed and the command is rejected.

|7 | _INPUT MODE_

| Except for logical editing characters, all data entered in INPUT mode is considered to be part of the file.

|7A| RETURNING TO EDIT MODE FROM INPUT MODE

To change from INPUT to EDIT mode, enter a null line (RETURN key or equivalent).

Section 3: Edit Subcommands

EDIT SUBCOMMAND SUMMARY BY FUNCTION

- Moving the Pointer

```
FIND     TOP      UP    NEXT    SCROLL    BACKWARD
LOCATE   BOTTOM   DOWN  nnnnn   FORWARD
```

- Changing EDIT Modes

```
INPUT
(NULL LINE)
```

- Specifying Record Attributes

```
IMAGE     RECFM    CASE
FNAME     FMODE    LRECL (from EDIT command)
```

- Modifying Data

```
| CHANGE    REPLACE   INPUT    TABSET   DSTRING
  ALTER     DELETE    REPEAT   TRUNC
  OVERLAY   GETFILE   RENUM    ZONE
```

- Displaying Editor Output

```
TYPE      LONG
VERIFY    SHORT
```

- Saving Intermediate Results

```
SAVE   AUTOSAVE
```

- Miscellaneous

```
CMS       PRESERVE    SERIAL
RESTORE   ?           STACK
REUSE     X or Y
```

- Ending the Session

```
QUIT
FILE
```

- Editing Line Numbers

```
LINEMODE   RENUM
nnnnn      PROMPT
```

| • Changing 3270 Terminal Modes

```
| FORMAT
```

ALPHABETIC LISTING OF EDIT SUBCOMMANDS

The following pages describe the EDIT subcommands, which are listed in alphabetic order for easy reference.

Some EDIT subcommands operate differently on 3270 terminals. These subcommands are:

BACKWARD	DSTRING	OVERLAY
BOTTOM	FORWARD	TOP
CHANGE	INPUT	TYPE
DELETE		

The SCROLL subcommand, and the CHANGE subcommand issued without operands, operate only on 3270 terminals in DISPLAY mode. The FORMAT subcommand operates only on 3270 terminals, and only if the EDIT command was issued without the NODISP option.

ALTER SUBCOMMAND

The ALTER subcommand searches, starting at the current line, all or any part of a file for a character or a byte of data and alters that character or byte, if found, to another character or byte of data. A space is a delimiter.

Note: Do not use this subcommand to change a character string more than one character long.

ALter	{parm1} {parm2} [1|n|* [G|*]]

parm1
Specifies either the character or two contiguous hexadecimal digits (0-9, A-F) for which the Editor is to search.

parm2
Specifies either a character or two contiguous hexadecimal digits (0-9, A-F) with which the Editor is to replace parm1, if found.

n or *
Indicates the number of lines to be searched, starting at the current line. If * is entered, the search is performed until the end of the file is reached. If this option is omitted, then only one line is searched.

G or *
Requests the Editor to alter every occurrence of parm1 in the lines specified. If G or * is not specified, only the first occurrence of parm1 in each line specified is altered.

When verification is on, each line that is altered is displayed at the terminal.

Example:

ALTER may be used to create an SLC card in the following manner:

In INPUT mode type:

```
XSLC  00100
```

then, in EDIT mode type:

```
alter X 02
```

The sequence in this example results in the proper hexadecimal character being placed in position 1 of the record.

Note: You can issue the ZONE subcommand to indicate those columns that are to be operated on by ALTER.

AUTOSAVE SUBCOMMAND

This subcommand allows you to save automatically the current copy of your file during an EDIT session. The frequency of the automatic save execution is specified by you. The subcommand format is:

AUTOsave	[n|OFF]

n
   Specifies the number of updates to be made between automatic saves. n is a decimal number between 1 and 32767. An addition, deletion, or change of a line is usually treated as one update. Each line affected by the $MOVE macro is treated as one update. However, all changes caused by one CHANGE, DELETE, DSTRING, GETFILE, or OVERLAY subcommand are treated as a single update, no matter how many lines are affected.

OFF
   Disables the automatic save feature.

no operands
   Displays the current AUTOSAVE setting at the terminal.

The AUTOSAVE subcommand causes execution of the EDIT SAVE function when the specified number of updates is made to the file while in EDIT or INPUT mode. Since the original file is replaced by updated versions of the file, it is up to you to rename or otherwise protect your original file.

The message "_SAVED" is displayed at the terminal when the SAVE operation occurs. The AUTOSAVE subcommand issued with no operands results in the current AUTOSAVE status being displayed at the terminal. The default mode for all filetypes is OFF. Therefore, if you desire automatic save mode, you must issue the AUTOSAVE subcommand with a numeric operand. You can issue the AUTOSAVE subcommand at any time during an EDIT session.

BACKWARD SUBCOMMAND

BAckward	[1|n]

n

Is the number of records backward you wish to move in the file being edited. On a typewriter terminal, the record that appears n lines before the current line is typed. On a 3270, records move down and off the screen n lines. (This is equivalent to UP.) If n exceeds the number of records in the file before the current line, "TOF:" is displayed on line 9.

BOTTOM SUBCOMMAND

The BOTTOM subcommand makes the last line of the file the current line without causing an end-of-file condition. When verification is on, the bottom line is displayed at the terminal. The BOTTOM subcommand has no operands. If the BOTTOM subcommand is issued at a 3270 display terminal in DISPLAY mode, "EOF:" is displayed on line 10, lines 2 through 9 contain the last eight records of the file, and lines 11 through 23 are blank.

Bottom	

CASE SUBCOMMAND

The CASE subcommand indicates how the Editor is to process (or inquires how the Editor is processing) uppercase and lowercase letters.

CASE	[M|U]

M

Indicates that the Editor is to accept any mixture of uppercase and lowercase letters for the file as they are entered at the terminal. If you have a 3270 that does not have the lowercase feature (RPQ), you can key in lowercase characters, but they appear on the screen as uppercase characters.

U

Indicates that the Editor is to translate all lowercase letters to uppercase letters before the letters are entered into the file.

The Editor assumes a default of U for all filetypes except MEMO and SCRIPT. If no operand is specified for CASE, the present setting is displayed at the terminal.

CHANGE SUBCOMMAND

The CHANGE subcommand causes the Editor to search all or any part of a file for a specified group of characters, and, if found, change that group of characters to another group of characters of the same or a different length.

If you issue the CHANGE subcommand without operands at a 3270 display terminal in DISPLAY mode, the following occurs:

1. The record pointed to by the current line pointer appears in the user input area of the display. (Nonprintable characters are stripped from the record.)

2. You can then alter the record in the user input area by retyping part or all of the line, or by using the INSERT or DELETE keys to insert or delete characters.

3. When the change is completed, you press the ENTER key, which causes the record in the user input area to replace the old record at the current line in the output display area.

The INSERT MODE key on the 3270 allows you to insert one or more characters into the display line. All existing data to the right of the insertion are progressively shifted one position to the right until position 80 is encountered. Any additional insertions move spillover characters to the beginning of the next line. If you use LINEMODE RIGHT (sequence numbers on the right side of each record), the use of the INSERT and DELETE keys causes the sequence number to shift; therefore, the sequence number may become invalid.

The CHANGE subcommand is treated as an invalid subcommand if it is issued without operands at a typewriter terminal or at a 3270 display terminal that is not in DISPLAY mode.

The CHANGE subcommand should not be issued without operands when you are editing a file that contains binary information (such as an assembler TEXT file) at a display terminal. This causes nonprintable characters (such as backspaces, tabs, and carriage returns) to be stripped from the line when the line is moved to the user input area. You should issue the CHANGE subcommand with operands to change a line that contains binary information, so that you overlay only those nonprintable characters you wish to change.

If you bring a line down to the user input area and decide not to change it, press the ERASE INPUT key and the ENTER key, and the line will be replaced in the file intact.

If the character string inserted causes the data line to extend beyond the truncation column, the excess characters are truncated. (See the description of the TRUNC subcommand for additional information on truncation.)

```
┌──────────────────────────────────────────────────────────────────────┐
│              │                                                       │
│ Change       │ [string1/string2[/[1|n|* [G|*]]]]                     │
│              │                                                       │
└──────────────────────────────────────────────────────────────────────┘
```

/ (diagonal)
   Signifies any unique delimiting character that does not appear in the character strings involved in the change.

string1
Specifies the group of characters to be changed (old data). String1 may be a null string.

string2
: Specifies the group of characters to be inserted (new data). If no additional information is to be entered on the command line, the closing delimiter may be omitted. String2 may be a null string.

n or *
: Indicates the number of lines to be searched, starting at the current line. If * is entered, the search is performed until the end of the file is reached. If this option is omitted, then only one line is searched.

G or *
: Requests the Editor to change every occurrence of string1 in the lines specified. If G or * is not specified, only the first occurrence of string1 in each line specified is changed. If string1 is null, G or * may not be specified.

When verification is on, each line that is changed is displayed at the terminal. Global changes made on a 3270 cause each line that is changed to be temporarily moved to line 9. All other lines on the screen are moved accordingly to maintain the proper record sequence. When the change operations are complete, the word "EOF" (indicating the end of the file) is displayed on line 9, preceded by the last seven records of the file. Lines 10 through 23 are blank.

Example:

Changing First Occurrence Only

```
BEFORE: QLPHQ=QLPHQ-BETQ
CODE:   c /q/a/
AFTER:  ALPHQ=QLPHQ-BETQ
```

Changing All Occurrences in the Line

```
BEFORE: QLPHQ=QLPHQ-BETQ
CODE:   c ZqZaZ1 *
AFTER:  ALPHA=ALPHA-BETA
```

Note: Z is used as a delimiter in this example to show that the delimiter need not be a diagonal (/).

The CHANGE subcommand can be used to display without change all lines that contain the information specified in string1. This may be accomplished by entering:

```
Change /string1/string1/ * *
```

When verification is on, every occurrence in every line is displayed.

Note: The ZONE subcommand can be issued to indicate those columns that are to be operated on by CHANGE.

Inadvertent Changes

Occasionally, if the data contained within string1 is not sufficiently definitive, global changes can cause serious errors to be made throughout the file.

For example, assume the following two lines exist:

```
This is a
wise choice.
```

If you change the "is" to "may be" but do not precede and follow the "is" with blanks, the command:

```
change /is/may be/ * *
```

results in:

```
Thmay be may be a
wmay bee choice.
```

A better way is to key in:

```
change /ƀisƀ/ƀmay beƀ/ * *
```

Note: The ƀ is for illustration only, and signifies one blank per ƀ.

Result:

```
This may be a
wise choice.
```

Thus, you can see that you should use caution when making global changes.

CMS SUBCOMMAND

The CMS subcommand causes the Editor to enter the CMS Subset mode, which allows you to execute those CMS commands that do not need to use the main storage being used by the Editor. The CMS subcommand has no parameters.

CMS	

The CMS commands that you can execute are: ACCESS, CP, DISK, ERASE, EXEC, LISTFILE, PRINT, PUNCH, QUERY, READCARD, RENAME, RETURN, SET, STATE, and TYPE. Any attempt to execute an invalid CMS command, or one that requires main storage, causes the response:

```
UNKNOWN CP/CMS COMMAND
```

Since use of system loading facilities could potentially overlay either the Editor or the file on which it is operating, the commands LOAD, INCLUDE (RESET), START, and RUN should not be executed.

To resume editing, enter the CMS subset command RETURN.

DELETE SUBCOMMAND

The DELETE subcommand deletes all or any part of a file. The new current line is the one following the deleted lines. A DELETE subcommand or an INPUT subcommand that includes the data line to be inserted causes a verification output display if the terminal is a display terminal and VERIFY is ON.

If you delete a record when using a display terminal in DISPLAY mode, the Editor rewrites the output display area with the top seven records unchanged. The bottom 12 records move up by one, and a new record (if one exists) moves into the bottom of the output display area (line 23).

DELete	[1|n|*]

n or *
: Indicates the number of lines to be deleted, starting at the current line. If * is entered, the remainder of the file is deleted. If this option is omitted, only one line is deleted.

DOWN SUBCOMMAND

The DOWN subcommand advances the line pointer forward in the file. The line pointed to becomes the new current line. DOWN operates in the same way as the NEXT subcommand. (Also see the "UP Subcommand" discussion.)

DOwn	[1|n]

n
: Indicates the number of lines to advance the pointer, starting at the current line. If this option is omitted, the pointer is advanced only one line.

When verification is on, the line pointed to is displayed at your terminal; if the DOWN subcommand encounters the end of the file, "EOF:" is displayed.

DSTRING SUBCOMMAND

The DSTRING subcommand deletes the lines from the current line down to, but not including, the first line containing the specified character string. The current line is not checked for the character string.

DString	/string[/]

/ (diagonal)
signifies any unique delimiting character that does not appear in the string.

string
specifies the group of characters to be checked for.

The zone set by the ZONE subcommand or the default zone setting is checked for the presence of the character string. A character string with a length greater than the current zone setting causes the error message "ZONE ERROR."

A null character string deletes the current line.

If the character string is not found by the end of the file, no deletions occur, the current line pointer is unchanged, and the message "STRING NOT FOUND, NO DELETIONS MADE" is displayed.

When the DSTRING subcommand is issued at a display terminal in DISPLAY mode, the screen is changed to reflect the changes to the file, if verification is on.

FILE SUBCOMMAND

The FILE subcommand writes the edited file on disk and, optionally, overrides the file identification originally supplied in the EDIT command.

FILE	[filename [filetype [filemode]]]

filename
Indicates the filename for the file. A new filename can also be specified by the FNAME subcommand. If filename is omitted, filetype and filemode cannot be specified, and the existing filename, filetype, and filemode are used.

filetype
Indicates the filetype for the file.

filemode
Indicates the filemode for the file. A new filemode can also be specified by the FMODE subcommand.

Any existing file that has an identical file identification is replaced, and the Editor operation is completed. If errors are encountered in performing the operation, appropriate error messages are displayed, and control returns to EDIT mode to allow recovery attempts. (See "Section 6: Error Conditions and Recovery Procedures" for further information.)

FIND SUBCOMMAND

The FIND subcommand searches the beginning of each line of data for a match on some specified information. Only one blank delimiter may be used after the subcommand. Additional blanks are considered to be a part of the specified character string.

Find	[line]

line
   Indicates any valid input line. It may contain blanks and tab characters, as well as any combination of alphanumeric or special characters.

The file is searched, beginning with the next line, examining only those characters in the beginning of each line which correspond in position to the nonblank characters in the specified line. The Editor replaces the TAB character or characters with the appropriate number of blanks before searching for a match. The first line in which a match occurs becomes the new current line. If none is found before the end of the file is reached, the message NOT FOUND is displayed. If the specified line is null or blank, the search is successful on the first line examined. If the line pointer is at the end of the file when the FIND subcommand is issued, the search starts from the top of the file.

Searches may be made for character strings that begin in columns other than column one by including sufficient logical or physical tabs before starting the text information. The column in each line in which searching begins is determined by the position of logical tab settings and the number of times you pressed the tab key.

When verification is on, the line is displayed at your terminal when it is found.

FMODE SUBCOMMAND

The FMODE subcommand resets the filemode for subsequent FILE, SAVE, or AUTOSAVE subcommands, or displays the current filemode setting.

FMode	[filemode]

filemode
   Indicates the filemode that is to replace the current filemode setting. If only a mode letter is given, the existing mode number is retained. If the filemode is not specified, then the current filemode is displayed.

If a subsequent FILE, SAVE, or AUTOSAVE subcommand is issued and the Editor determines that the filemode prevents writing, a message is displayed to tell you to change the filemode.

FNAME SUBCOMMAND

The FNAME subcommand resets the filename for subsequent unqualified FILE and SAVE subcommands, or displays the current filename.

FName	[filename]

filename
   Indicates the filename that is to replace the current filename. If a new filename is not specified, then the current filename is displayed.

FORMAT SUBCOMMAND

The FORMAT subcommand changes the mode of a local or remote 3270 terminal from DISPLAY to LINE (or LINE to DISPLAY). You can issue this subcommand at any time during the editing session, if you invoked the EDIT command without the NODISP option.

FORMat	{DISPLAY|LINE}

DISPLAY
   Specifies that a full screen display of data is to occur. Subcommands do not appear as part of the data displayed.

   If you are using a remote 3270 in DISPLAY mode, the terminal is forced into LINE mode whenever you enter INPUT mode to add to your file or create a new file. The terminal returns to DISPLAY mode when you leave INPUT mode.

LINE
   Specifies that the display station is to operate as a typewriter terminal. Every line you enter is displayed on the screen; the screen looks like a typewriter terminal's console sheet.

The FORMAT subcommand is treated as an invalid subcommand if it is issued:

1. Without operands, or
2. At a typewriter terminal, or
3. At a display terminal if the EDIT command was invoked with the NODISP option.

The terminal mode that you set by issuing the FORMAT subcommand remains in effect until you:

1. Issue a QUIT or FILE subcommand to end the EDIT session, or
2. Change it by issuing the FORMAT subcommand with the other mode.

FORWARD SUBCOMMAND

The FORWARD subcommand moves the current line pointer forward in the file you are editing.

FOrward	[1|n]

n
: Is the number of records forward you wish to move in the file being edited. On a typewriter terminal, the record that appears n lines after the current record is typed. On a 3270, records move up and off the screen n lines. (This is equivalent to DOWN or NEXT.) If n exceeds the number of records remaining in the file, EOF is displayed on line 9, the current line.

GETFILE SUBCOMMAND

The GETFILE subcommand inserts all or part of a specific CMS file into a file that you are processing.

Getfile	filename [filetype|* [filemode|* [1|m [n|*]]]]

filename
: Indicates the filename of the file that contains the data to be inserted into the file you are processing.

| filetype or *
: Indicates the filetype of the file that contains the data to be inserted. The filetype may be specified as an asterisk (*), which indicates that all filetypes with the given filename are to be inserted. If a filetype is not specified, the filetype of the file you are processing is assumed.

filemode or *
: Indicates the filemode of the file that contains the data to be inserted. If a filemode is not specified, then * is assumed and all of your disks are searched for the file.

m
: Indicates the first line of the file that is to be inserted into the file being edited. If m is omitted, then the first line of the file is assumed.

n or *
: Indicates the number of lines to be inserted, starting with the line specified by m. If a number is not specified, or * is specified, then that part of the file between m and the end of the file is inserted.

The GETFILE operand list is positional; if you omit one operand, you cannot specify any operands that follow. Thus, if you want to specify _m_ and _n_, you must specify the filetype and filemode for the file.

When verification is on, if the end of the file containing the data is encountered, the message EOF REACHED is displayed. The last line inserted becomaes the new current line. If the record length of the records in the file containing the data to be inserted exceeds that of the file being edited, an error message is displayed, and the GETFILE is not executed; if shorter, the records are padded to the record length of the file being edited and inserted in the file.

If you use the GETFILE subcommand to insert lines into a VSBASIC file, you must also use the RENUM subcommand to resequence the file.

Example:

Suppose you are writing a program in which you want to process certain types of data in the same way as you do in other programs, and the source statements of the other programs are available on a disk that can be processed by the Editor. You can use the GETFILE subcommand to copy needed parts of the other programs and insert them into the program you are writing:

```
.
.
.
LABLEX   B  XYZ                 Statement in your program
g listx assemble a1 452 120     Translate routine from LISTX
          BR   R14
g pay assemble a1 457 93        Convert routines from PAY
          BR   R14
g print assemble a1             Get all of PRINT program
EOF REACHED
.
.
.
```

You must establish the proper linkage to and from those programs that you are using so that your program will execute properly.

IMAGE SUBCOMMAND

The IMAGE subcommand controls how the Editor looks at, manipulates, and expands input data, or displays the current IMAGE setting.

IMAGE	[ON|OFF|CANON]

ON

If ON is specified, text entered while in INPUT mode or as lines of data in the operands of the FIND, INPUT, OVERLAY, and REPLACE subcommands, is expanded into a "line image" where backspaces are removed and tabs are replaced by the appropriate number of blanks. The process that builds the line image simulates a typewriter having 133 columns per line for printing, with appropriate tab stops. If the input line contains backspaces, the column pointer can move forward and backward. The rules are as follows:

- Backspace characters immediately following a command name are interpreted as separator characters and do not delete any part of the command name.

- Backspace characters act in a similar manner to the logical character-delete symbol, in deleting the previous characters if a sufficient number of other characters or blanks follow the backspace characters.

- If a backspace character is the last character in the input line, it is ignored.

- If an attempt is made to put a nonblank character beyond column 133, or if a nonblank character remains in the line beyond the normal or redefined truncation column for that filetype, then truncation occurs.

Text entered in the form of delimited strings, as in CHANGE, LOCATE, and ALTER, is not expanded. Thus, tabs and backspaces are treated in the same way as other characters.

IMAGE ON is the default for all filetypes except SCRIPT.

OFF

If OFF is specified, tabs and backspaces are treated as data characters in the same way as other characters. Thus, they enter the file without being deleted, translated, expanded, or reordered.

CANON

When CANON is specified, backspaces can produce compound characters such as underlined words, headings, or phrases. Before they are inserted in the file, compound characters are put into an arrangement that is independent of the order in which the characters were keyed in. (Backspaces are arranged singly between the characters that overlay each other, and the overlaying characters are arranged according to their EBCDIC values.) Tab characters do not receive special treatment, and enter the file in the same way as ordinary printing characters. CANON is the assumed default for SCRIPT files. For an example of the use of CANON, see "Data Files" in Section 1.

When a line of text follows a subcommand in the same line of input, the text begins with the character following the delimiter after the subcommand name.

Note: Some subcommands, such as UP, DOWN, and NEXT, which can take only a single numeric operand, do not require a delimiter separating the subcommand name from the operand.

INPUT SUBCOMMAND

The INPUT subcommand creates new lines and places them in a file, or, if no data line is specified, leaves EDIT mode and enters INPUT mode.

Input	[line]

line
Specifies the exact input line to be entered into the file. It can contain blanks and tabs.

If a data line is specified, it is put into the file following the current line and subsequently becomes the current line. The Editor remains in EDIT mode. This data line option is invalid if the LINEMODE option is set LEFT or RIGHT.

If no data is entered, the Editor enters INPUT mode, receives lines from the terminal, and puts them into the file following the current line. As each line is entered, it becomes the new current line. EDIT mode is restored by entering a null line. Typing at least one blank inserts a blank line.

Display Terminal Considerations

If you insert a record when using a local display terminal in DISPLAY mode, the Editor rewrites the entire output display area. The newly inserted record becomes the current line on line 9. The old current line and all records above it move up one line, except for the topmost record formerly on line 2, which is deleted from the screen.

If you issue an INPUT subcommand with no text when using a local display terminal in DISPLAY mode, the Editor causes a change to INPUT mode. INPUT appears in the EDIT status field and you can enter input text in the display user input area. After you enter each input line, the output area reflects the current status of the file; that is, the line last entered appears on line 9 of the display and all previous lines are moved upward and finally dropped from the screen. As before, a null entry in INPUT mode causes a return to EDIT mode.

If you are using a local or remote display terminal in LINE mode, the terminal remains in LINE mode when you issue the INPUT subcommand. That is, it continues to operate as if it were a typewriter terminal, displaying lines on the screen as you enter them. A terminal can be in LINE mode and in INPUT mode at the same time.

If you are using a remote display terminal in DISPLAY mode and you issue the INPUT subcommand with no text, the terminal is forced into LINE mode. The display of the file on the screen disappears and the word INPUT: appears. As you enter input lines, they appear in the output display area. When you leave INPUT mode by entering a null line, the remote terminal returns to DISPLAY mode. The display of the file reappears on the screen, with the lines you have just entered in their proper place in the file.

When you are entering data in INPUT mode at a display terminal that is in LINE mode, a tab character generated by a program function (PF) key only generates one character, and appears as one character on the screen. That is, the line does not appear spaced according to the tab settings.

LINEMODE SUBCOMMAND

The LINEMODE subcommand can be invoked for COBOL, BASIC, VSBASIC, FREEFORT, or similar files having fixed 80-character records. It permits you to choose whether or not the Editor is to create a file in which every line is prompted for and numbered in ascending order, and in which the line number is placed either at the beginning or end of each record. If no operand is specified, the current LINEMODE setting is displayed.

When in INPUT mode, the Editor displays at the beginning of each line the line number associated with the next record to be keyed in (except when LINEMODE OFF is specified). This prompting appears at the beginning of each line, regardless of whether LINEMODE LEFT or RIGHT was specified. The 3270 display terminals display the line number in the message area at the top of the screen. You can use the nnnnn subcommand described later in this section to insert records into a numbered file while in EDIT mode.

Note: LINEMODE LEFT or RIGHT should not be specified for files containing variable-length records, because these positions in the records may already contain valid data.

LINEmode	[Left|Right|OFF]

LEFT
LEFT is the default for BASIC, VSBASIC, and FREEFORT files. If LEFT is in effect for BASIC or VSBASIC files, columns 1-5 of the records are set aside for line numbers. If the file already exists, the Editor assumes that all records are numbered in ascending order in columns 1-5 and that all numbers are right justified with leading blanks. The near zone and the first column in which text is entered (first tab setting) are set to 7 so the ALTER, CHANGE, FIND, LOCATE, and OVERLAY subcommands do not process the line numbers. If tabs are set anywhere in positions 1 through 5, the data in columns 1 through 5 is lost for each line entered after that point.

If LEFT is in effect for FREEFORT files, columns 1-8 are set aside for line numbers, and the near zone is set to 9. All the other above considerations still apply.

LINEMODE LEFT should not be specified for ASSEMBLE files, since the label of assembler statements appears on the left.

RIGHT
If RIGHT is specified, column 76-80 of the record are set aside for line numbers. If the file already exists, the Editor assumes that all records are numbered in ascending order in columns 76-80, and that all numbers are right justified with leading zeros. RIGHT is only valid for files with fixed length 80-character records.

Specify LINEMODE RIGHT for ASSEMBLE files only if you want line number prompting. If not, do not specify LINEMODE at all since serialization is already in effect.

Lines entered at a typewriter terminal are reformatted so that the line number, followed by a blank, appears in the file at the beginning of the line being typed. On entering LINEMODE RIGHT, the verification column is reset to 72 to prevent the line number in columns 76-80 from printing a second time. In INPUT mode, you are prompted with a 5-digit number followed by a blank (as when lines are typed). This number is placed into the file in columns 76-80. Its appearance on the left is to minimize typing time and does not indicate its true position. A display terminal always reflects the true position of the line number.

If LINEMODE is LEFT or RIGHT, the data line operand of the INPUT subcommand is disabled, and treated as an invalid request. However, if while in EDIT mode you simply invoke the INPUT subcommand with no operands, the Editor goes into INPUT mode and prompts you with line numbers. The line numbers usually are successive multiples of the prompting increment. However, if the number so generated for prompting is larger than the next line number in the file, the Editor selects a number between the current line number and the next line according to a pre-established algorithm. If a prompting number cannot be generated, because the current line and the next line differ by only one, a message is displayed telling you to renumber the lines, and EDIT mode is entered. For example, if LINEMODE RIGHT is in effect for an ASSEMBLE file, respond as follows:

```
RENUMBER LINES
EDIT:
line off
serial all 10
save
EDIT:
line right
```

If the file is a BASIC, VSBASIC, or FREEFORT file with LINEMODE LEFT in effect, do the following:

```
RENUMBER LINES
EDIT:
renum
```

If a SAVE, AUTOSAVE, or FILE subcommand is issued while LINEMODE is set to RIGHT, reserialization in columns 76-80 is suppressed to allow the line numbers that are saved on disk to have the same values they had during the editing session. If LINEMODE is set to OFF before issuing the subcommand, reserialization takes place according to the SERIAL options currently in effect. This allows you to renumber all your lines during an editing session.

OFF
If OFF is specified, line numbers are omitted unless serialization is in effect. There is no prompting with numbers in INPUT mode. The nnnnn subcommand is disabled. The start column (first logical tab stop) is reset to '1' and the ZONE setting is unchanged.

The default settings are:

Filetype	LINEMODE Setting
BASIC	LEFT
FREEFORT	LEFT
VSBASIC	LEFT
All Others	OFF

LOCATE SUBCOMMAND

The LOCATE subcommand scans the file, beginning with the next line, for the first occurrence of a specified string of characters.

Locate	/string[/]

/ (diagonal)
Signifies any unique delimiting character that does not appear in the string. The delimiter may be any nonblank character. The closing delimiter is optional.

string
Specifies any group of characters to be searched for in the file.

If the string is found before the end of the file is reached, the line containing it becomes the new current line; otherwise, the message NOT FOUND is displayed.

If the beginning delimiter is /, the subcommand name LOCATE or an abbreviation for it need not be entered. If the string is null or blank, the search is successful on the first line encountered. If the line pointer is at the end of the file when LOCATE is issued, scanning starts from the top of the file. When verification is on, the located line is displayed at your terminal.

Example:

```
l /format/
FORMAT('DAILY AUDIT')
```

Note: The ZONE subcommand can be issued to indicate those columns that are to be operated on by LOCATE.

LONG SUBCOMMAND

The LONG subcommand instructs the Editor to respond ?EDIT: when an invalid EDIT subcommand or macro is entered. This is the default mode for the Editor. The LONG subcommand has no operands. For the short form of Editor responses, see the description of the SHORT subcommand.

LONG	

NEXT SUBCOMMAND

The NEXT subcommand advances the line pointer forward to the next line or a specified number of lines. The line pointed to becomes the new current line.

Next	[1|n]

n
   Indicates the number of lines to advance the line pointer. If n is omitted, then the pointer is moved down only one line.

NEXT operates in the same way as the DOWN subcommand. When verification is on, the line pointed to is displayed at your terminal. Also, if NEXT causes the end of the file to be reached, the message EOF: is displayed.

Example:

```
next 2                          results in:
This is the line pointed to.
n 1000                          results in (if the file is less
                                than 1000 lines long):
EOF:
```

OVERLAY SUBCOMMAND

The OVERLAY subcommand selectively replaces one or more character strings in the current line with the corresponding nonblank characters in the line being keyed in. Blank characters in the input line indicate that the corresponding characters in the current line are not to be altered.

This subcommand may be entered at a typewriter terminal by typing the letter "o", followed by a backspace, followed by the overlaying characters. This sets up the correct alignment on the terminal (see example).

Because the backspace function on the 3270 display device causes the next character entered to replace the previous one, column alignment for the OVERLAY subcommand cannot be achieved on the 3270 in the same way as on a typewriter terminal. As an alternative, however, you can issue the OVERLAY subcommand with no operands and then put the overlay argument in the user input area of the display and press ENTER.

Overlay	line

line
   Specifies an input line that replaces corresponding character positions in the current line.

An underscore in the overlaying line must be used to place a blank into the corresponding position of the current line. Thus, an underscore cannot be placed (or replaced) in a line by using an OVERLAY subcommand.

Example:

```
EDIT:
top
TOF:
n
programmer
o       ing
programming
```

OVERLAY should be used with care on lines containing underscored words or other compound characters.

When verification is on, the line is displayed at the terminal after it has been overlaid.

PRESERVE SUBCOMMAND

The PRESERVE subcommand retains the current settings of the CASE, FMODE, FNAME, IMAGE, LINEMODE, LONG, RECFORM, SERIAL, SHORT, TABSET, TRUNC, VERIFY, and ZONE subcommands. Any or all of these subcommands may be then set to new values. All of the saved settings are restored when you issue a RESTORE subcommand. The PRESERVE subcommand has no operands.

PREserve	

PROMPT SUBCOMMAND

The PROMPT subcommand changes the prompting increment by which line numbers are incremented during line number prompting. This is only applicable to INPUT mode when LINEMODE RIGHT or LINEMODE LEFT is in effect.

PROMPT	[n]

n
: Specifies the prompting increment. The setting specified should not exceed 32,767. 10 is the initial increment setting. If n is omitted, the current setting is displayed.

QUIT SUBCOMMAND

The QUIT subcommand terminates the current editing session and leaves the previous copy of the file, if any, intact on the disk. You would normally use the QUIT subcommand if you discover that serious errors have been made during the editing session and you do not wish to preserve the contents of the file currently being edited. If a SAVE or AUTOSAVE subcommand was issued before the QUIT subcommand and the file identification was not changed, the file on disk contains the changes that occurred during the session up to the time the file was saved. The QUIT subcommand has no operands.

QUIT	

RECFM SUBCOMMAND

The RECFM subcommand indicates to the Editor whether the record format for the file is fixed or variable, or displays the current RECFM setting.

RECfm	[F|V]

F
   Indicates fixed-length records. F is assumed by default for all new files except LISTING, SCRIPT, FREEFORT, VSDATA, and BASDATA.

V
   Indicates variable-length records. V is assumed by default for all new LISTING, FREEFORT, VSBDATA, BASDATA, and SCRIPT files. Usually, a variable format file occupies a smaller amount of disk space because trailing blanks are deleted from each line before it is written onto disk.

If neither V nor F is entered, the current RECFM setting is displayed.

RENUM SUBCOMMAND

The RENUM subcommand recomputes the line numbers for VSBASIC and FREEFORT source files. Also, for all VSBASIC statements that have statement numbers for operands, those operands are recomputed so that they correspond to the new line numbers. The VSBASIC statements with line number operands are:

DELETE	IF	READFILE
EXIT	INPUT	REREADFILE
GET	PRINT USING	REWRITEFILE
GOSUB	PUT	WRITEFILE
GOTO		

The Editor generates new line numbers when you enter the RENUM subcommand and specify initial and increment line number values.

RENum	[strtno|10 [incrno|strtno]]

strtno
   Indicates the number from which you wish to start renumbering your file. Because RENUM renumbers the whole file from beginning to end, the number you specify as strtno becomes the statement number of the first statement in the newly renumbered file. This number may not exceed 99999 for VSBASIC files or 99999999 for FREEFORT files. The default start number value is 10 and the specified start number must not be zero.

incrno
   Indicates the increment number value by which you wish to renumber your file. This value may not exceed 99999 for VSBASIC files or 99999999 for FREEFORT files. The default for incrno is strtno, the first sequence number in the renumbered file, and the specified incrno must not be zero.

If you do not specify strtno and incrno, the default value for both is 10. If you specify only strtno, incrno defaults to the same value as strtno.

The current line pointer remains as it was before you specified RENUM regardless of whether or not RENUM completes successfully. If you are editing a VSBASIC file, the file to be renumbered must either originate from a read/write disk or you must issue an FMODE subcommand to change the file destination to a read/write disk.

If any error occurs during the RENUM operation, the Editor terminates the RENUM operation and the file being edited remains unchanged.

REPEAT SUBCOMMAND

The REPEAT subcommand executes the immediately following OVERLAY subcommand (or an X or Y subcommand assigned to invoke OVERLAY) for the specified number of lines, or to the end of the file.

If the subsequently entered subcommand is not one of the above, REPEAT has no effect on it.

```
 REPEAT      | [1|n|*]
```

n or *
: Indicates the number of times to repeat the specified OVERLAY subcommand or its equivalent X or Y subcommand that immediately follows, starting at the current line. The asterisk (*) indicates that the request is to be repeated until the end of the file is reached. If neither n nor * is specified, then only one line is handled. The last line processed becomes the new current line.

REPLACE SUBCOMMAND

The REPLACE subcommand replaces the current line with a specified line. If no line is specified, the Editor deletes the current line and enters INPUT mode.

```
 Replace     | [line]
```

line
: Specifies an input line that is to replace the current line. If a line is specified, then the Editor puts it into the file in place of the current line. If no line is specified, the Editor deletes the current line, enters INPUT mode, and receives lines from the terminal. As each line is entered, it becomes the new current line. EDIT mode is restored by entering a null line. If the LINEMODE option is set LEFT or RIGHT, then REPLACE with a line operand is not valid. If REPLACE is used without any operands when LINEMODE is set LEFT or RIGHT, the current line is replaced using the next available line number.

RESTORE SUBCOMMAND

The RESTORE subcommand restores the settings of CASE, FMODE, FNAME, IMAGE, LINEMODE, LONG, RECFM, SERIAL, SHORT, TABSET, TRUNC, VERIFY, and ZONE to the values they held before the last issued PRESERVE subcommand, or to their initial values if a PRESERVE subcommand has not been issued. The RESTORE subcommand has no operands.

REStore	

RETURN SUBCOMMAND

RETURN is not an EDIT subcommand, but a CMS subset command used to return to EDIT mode from the CMS subset environment. It is listed here as a companion to the CMS subcommand. RETURN has no operands.

RETURN	

REUSE SUBCOMMAND

The REUSE subcommand (which can also be specified as =) allows you to stack last in, first out (LIFO) the last EDIT subcommand, except for REUSE or a question mark, and then execute the stacked subcommands.

{REUSE|=}	[subcommand]

subcommand
   Specifies any EDIT subcommand.

If an invalid EDIT subcommand is specified (and is therefore not executed), the stacked subcommand is deleted. Thus, the invalid subcommand has no effect except to display an error message.

REUSE can also be used to repeat a valid EDIT subcommand or, in some cases, to correct an invalid one. For instance, consider a user, with verification set off, who mistakenly thinks he is in INPUT mode and enters:

```
     The self-abnegation must be complete.  From
```

Because he is not in INPUT mode, the message

```
     ?EDIT: The self-abnegation must be complete.  From
```

is displayed. He then enters

```
     = input
```

| to insert the INPUT subcommand in front of the previously entered line
| of data, and continues entering lines of text.

```
     the point of view of Gautama, the dread of
     death, that greed for an endless continuation
     ...
```

If, instead of entering the "= input," he had entered "input," it would have been necessary to re-enter the first line of text.

The file now appears as:

```
     The self-abnegation must be complete.  From
     the point of view of Gautama, the dread of
     death, that greed  for an endless continuation
     ...
```

SAVE SUBCOMMAND

The SAVE subcommand saves the file that is being edited on disk.

SAVE	[filename [filetype [filemode]]]

filename
   Indicates the filename of the file to be saved.

filetype
   Indicates the filetype of the file to be saved.

filemode
   Indicates the filemode of the file to be saved.

The SAVE subcommand uses the specified filename, filetype, and filemode, or the current value in effect as a default for any parameter not specified. Any file on disk that has the identical filename, filetype, and filemode identification is replaced. You remain in the EDIT mode after SAVE is issued; after FILE is issued you are in CMS mode.

If you want to save data automatically, see the description of the AUTOSAVE subcommand.

SCROLL SUBCOMMAND

This subcommand provides you with a more efficient method of scanning an EDIT file than the TYPE subcommand. It displays a file in increments of 20 lines if the VERIFY setting is 80 characters or less, or increments of 10 lines if the VERIFY setting is greater than 80. This subcommand is valid only with a display terminal in DISPLAY mode, and is the only subcommand that causes the screen to go into MORE... status.

S[croll[Up]	[*|n|1]

SCROLL
   Causes the Editor to scan forward through the file. The minimum abbreviation for SCROLL is S.

SCROLLUP
   Causes the Editor to scan backward through the file. SCROLLUP can be specified by any combination of the abbreviations of SCROLL and UP; the minimum abbreviation is SU.

*
   Displays in increments of 20 (or 10) lines to the end of the file (or to the beginning of the file, if SCROLLUP is specified).

n
   Is a number from 1 to 255 that specifies the number of screens of data to be displayed.

The current line pointer advances by 20 (or 10) output lines and the Editor rewrites the output display using the new CLP. This occurs the number of times that you indicate in the subcommand (with a one-minute pause between screen loads).

If you start scrolling forward from the beginning of the file, the first 7 lines (lines 2 through 8) are blank, line 9 (the CLP line) contains "TOF;", indicating the null line that precedes the first real record in the file, and lines 10 through 21 contain the first 12 records of the file. After the one-minute interval has elapsed (or after you press the CANCEL key), the next 20 records are displayed. Records 13 through 20 appear at the top part of the screen, and record 21 is the line currently pointed to.

At end-of-file, if the CLP was advanced by less than 20 lines, some lines are repeated in the next-to-last and last displays. This repetition is for consistency in displaying the CLP always at line 9. Lines 2 through 8 contain the last seven records of the file. Line 9 contains "EOF;", and the rest of the lines are blanks.

To stop scrolling, enter the HT command and press the CANCEL key twice.

SERIAL SUBCOMMAND

The SERIAL subcommand controls the serialization of records in columns 73-80. To change the record numbers in VSBASIC and FREEFORT files, use the RENUM subcommand.

SERial	{OFF|ON|ALL|seq}[incr|<u>10</u>]

OFF
Indicates that neither serialization number nor identifier is to be placed in columns 73-80.

ON
Indicates that the first three characters of the filename are to be used in columns 73-75 as an identifier.

ALL
Indicates that columns 73-80 are to be used for serialization numbers.

seq
Specifies a three-character identification to be used in columns 73-75.

incr
Specifies the increment for the line number in columns 76-80 (or 73-80). This number also becomes the first line number. If the number is not specified, then 10 is assumed.

Initially, serialization is determined by the filetype specified in the EDIT command. The SERIAL subcommand is valid only if the file uses fixed-length records with a record length of 80. It takes effect after LINEMODE has been turned off and a FILE, SAVE, or AUTOSAVE subcommand is issued. The SERIAL subcommand cannot be used to delete serialization inserted in a file by a previous SAVE, AUTOSAVE, or FILE subcommand that was issued with the serialization option active.

If LINEMODE has been set RIGHT, the SERIAL subcommand is ignored when the next FILE, AUTOSAVE, or SAVE is issued, and the following message is displayed:

```
RESERIALIZATION SUPPRESSED
```

Except for COBOL records, which have serialization in columns 1 through 7 of each record, normal serialization consists of a three-character identifier in columns 73 through 75, followed by a five-digit number in columns 76 through 80. This is obtained by issuing SERIAL with a specific three-character identifier or by using the ON option (which means that the sequence name is to be taken from the first three characters of the filename of the file being edited).

An alternate form of serialization uses an eight-digit number in columns 73-80. This is obtained by issuing SERIAL ALL. Serialization can be turned off by issuing SERIAL OFF.

When serialization is indicated for files with LINEMODE RIGHT active, the Editor sets the truncation value to the minimum of 72; the existing setting is changed, and the message TRUNC SET TO 72 is displayed. The end zone is also set to 72; if the end zone is changed, the message END ZONE SET TO 72 is displayed. The setting for the VERIFY subcommand is not changed.

SHORT SUBCOMMAND

The SHORT subcommand sets the message response mode. Invalid EDIT subcommands invoke the response ¬ instead of "?EDIT:". Invalid EDIT macros invoke the response ¬$. The SHORT subcommand has no operands.

SHORT	

STACK SUBCOMMAND

The STACK subcommand stacks lines or EDIT subcommands in the terminal input buffer for subsequent processing.

STACK	[_1_|n|subcommand]

n
: Indicates the number of lines to be stacked. If a number or a subcommand is not specified, then one line is assumed by default. The subcommand stacks _n_ lines on a first-in, first-out (FIFO) basis, starting with the current line, in the terminal input buffer. The line pointer position after execution of the STACK subcommand depends upon the lines (subcommands) that were stacked and executed. The length of the lines is taken from the column set by the TRUNC subcommand. If STACK is issued with an argument of 0 (zero), a null line is stacked. A maximum of 25 lines can be stacked.

subcommand
: Specifies that an EDIT subcommand is to be stacked. If a subcommand is specified, it is stacked FIFO. STACK enables subcommands to be issued from the file, and also makes it possible to move or copy lines.

TABSET SUBCOMMAND

The TABSET subcommand sets the logical tabs for the file. A space is used as a delimiter. TABSET may not be issued without at least one tab setting.

TABSet	{n_1 n_2 ... n_x}

n
: Indicates column positions for logical tab settings. Up to 25 are allowed. The first tab entry indicates the column that the column pointer identifies before the TAB key is pressed. The default settings are:

Filetypes	Default Tab Settings
ASM3705, ASSEMBLE, MACRO, UPDATE, UPDTxxxx	1, 10, 16, 31, 36, 41, 46, 69, 72, 80
FORTRAN	1, 7, 10, 15, 20, 25, 30, 80
FREEFORT	9, 15, 18, 23, 28, 33, 38, 81
BASIC, VSBASIC	7, 10, 15, 20, 25, 30, 80
PLIOPT, PLI	2, 4, 7, 10, 13, 16, 19, 22, 25, 31, 37, 43, 49, 55, 79, 80
COBOL	1, 8, 12, 20, 28, 36, 44, 68, 72, 80
Others	1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 61, 71, 81, 91, 101, 111, 121, 131

Tab settings have no effect if a SCRIPT filetype is used (canonical ordering in effect) or if the IMAGE subcommand is set off. A tab entered into a file under these conditions appears as X'05'.

TOP SUBCOMMAND

The TOP subcommand moves the line pointer to the top of the file. The null top line becomes the current line. The TOP subcommand has no operands. When you are using a display terminal, if you specify TOP and VERIFY is on, line 9 contains the characters TOP (indicating the top of the file), lines 2 through 8 are blank, and lines 10 through 21 contain the first 12 lines of the file.

TOP	

TRUNC SUBCOMMAND

The TRUNC subcommand changes the truncation column of records or displays the current truncation column setting.

TRUNC	[n|*]

n
Indicates the column at which truncation is to occur.

*
Indicates that truncation is to be set to the record length for the filetype.

The TRUNC subcommand sets the column of truncation to _n_, or to the record length. If no operand is specified, the current truncation column is displayed. The truncation value is used by the INPUT, REPLACE, CHANGE (for display terminals only), STACK, and OVERLAY subcommands. The defaults are:

Filetypes	Truncation Column
ASSEMBLE, UPDATE, MACRO, UPDTxxxx	71
FORTRAN, COBOL, PLI, PLIOPT	72
FREEFORT	81
Others: Fixed Length	Record Length
Others: Variable Length	132

Example:

```
        trunc 6
        top
        TOF:
        input ||||||||||||||||||||
        TRUNCATED
        ||||||
        replace ****************
        TRUNCATED
        ******
        overlay ++++++++++++++
        TRUNCATED
        ++++++
```

TYPE SUBCOMMAND

The TYPE subcommand displays all or any part of a file at the terminal. Entering the TYPE subcommand on the display console erases the present output display.

If you enter the TYPE subcommand with no operands, the record pointed to by the CLP appears on the line on which the 3270 "SYSTEM AVAILABLE" hardware indicator appears (line 9). Enough records precede and follow line 9 to complete the 20-line output display. If the records are 80 characters or less, each record occupies one display line. If a record exceeds 80 characters, the excess beyond 80 characters appears on the following line.

The TYPE subcommand displays _m_ lines, or to the end of the file, starting with the current line. The last line displayed becomes the new current line. The maximum length of the displayed lines is determined by the column set by the VERIFY subcommand. If LINEMODE RIGHT has been specified, the line numbers stored in columns 76 through 80 are displayed on the left. TYPE m issued with a display terminal performs the same function as the subcommand NEXT m-1.

Type	[1|m|* [n|*]]

m or *
Indicates the number of lines to be displayed. The asterisk (*) indicates all lines between the current line and the end of the file. If m or * is omitted, only one line is displayed. If the number of lines specified exceeds the number remaining in the file, displaying stops at the end of the file.

n
Indicates the column at which displaying is to stop. If you are using a display terminal, the second operand is ignored and no truncation of the record occurs.

*
Indicates that displaying is to take place for the full record length.

UP SUBCOMMAND

The UP subcommand repositions the line pointer to a line with a lower line number than the one that you are processing.

Up	[1|n]

n
Indicates the number of lines the pointer is to be moved back (up) in the file. If a number is not specified, then the pointer is moved up only one line. The line pointed to becomes the new current line.

When verification is on, the line pointed to is typed at your terminal.

VERIFY SUBCOMMAND

The VERIFY subcommand causes the Editor to display all or any part of a line at the terminal after it is processed, or to display the current verification setting.

Verify	[ON|OFF][[startcol|1] endcol|*]

ON
Allows editing verification. Lines that are located, altered, or changed are displayed, and changes between EDIT and INPUT mode are indicated.

OFF
Prevents editing verification. Lines that are located, altered, or changed are not displayed, and changes between EDIT and INPUT mode are not indicated.

startcol
is a number that indicates the column in which verification is to begin. The default is column 1. startcol must not be greater than the record length or greater than endcol.

endcol
is a number that indicates the last column to be verified. endcol must not be greater than the record length.

*
Indicates that the last column to be verified is the last column of the record.

Unless you specify VERIFY OFF, verification is on for all terminal types and for all filetypes.

If you issue the VERIFY subcommand with only one operand, that operand is assumed to be the endcol operand. For example, if you issue V 10, verification occurs in columns 1 through 10.

If you issue VERIFY with no operands, the current startcol and endcol settings are displayed, regardless of whether verification is on or off. Unless you have changed these settings, the start column is 1 and the end column depends on the filetype, as follows:

<u>Filetypes</u>	<u>Verification End Column</u>
ASSEMBLE, UPDATE, UPDTxxxx, FORTRAN, COBOL, PLI, PLIOPT, MACRO	Column 72
Others (Including FREEFORT)	Record Length

X OR Y SUBCOMMAND

The X or Y subcommands cause the Editor to assign to X or Y a given EDIT subcommand, or to execute the previously assigned subcommand a specified number of times.

```
{X|Y}        [1|n|subcommand line]
```

n
Indicates the number of times the previously assigned subcommand is to be executed. If X or Y is entered with no operands, 1 is assumed.

subcommand line
Indicates any EDIT subcommand followed by its usual operands. The Editor assumes that you have specified a valid EDIT subcommand, and no error checking is done.

If n is specified and is greater than 1, the current line is not necessarily advanced between executions. Advancement depends upon the EDIT subcommand that has been assigned to X or Y. Execution stops at the end of the file. If a number or a subcommands is not specified, the previously assigned subcommand is executed once.

X and Y are initially set to null strings.

Example:

If you wanted to create a number of entries similar to the boxes around the EDIT subcommands, it would be repetitious to enter successive lines of dashes and headings for each box. By issuing the {X|Y} subcommand in EDIT mode as follows:

```
edit test script
NEW FILE:
EDIT:
x i --------------------
y o |        |          |
```

The repetitious entries would be entered once initially. Then you can create the input file:

```
input
INPUT:
   SUB      SYN
   T1       T2
   T3       T4
   T5       T6
```

In EDIT mode, the X and Y subcommands are added:

```
EDIT:
top
TOF:
next
   SUB      SYN
repeat 4
y
|  SUB   |  SYN    |
|  T1    |  T2     |
|  T2    |  T3     |
|  T3    |  T4     |
|  T5    |  T6     |
top
TOF:
x#next
|  SUB   |  SYN    |
x#n
|  T1    |  T2     |
x#n
|  T3    |  T4     |
```

A display of the file would appear as follows:

```
top
TOF:
type *
--------------------
|  SUB   |  SYN    |
--------------------
|  T1    |  T2     |
--------------------
|  T3    |  T4     |
|  T5    |  T6     |
EOF:
```

ZONE SUBCOMMAND

The ZONE subcommand tells the Editor the portion of each record (starting position and ending position) to be scanned when scanning occurs, or to display the current ZONE settings.

Zone	[1|m|* [n|*]]

m

Indicates the first column of the zone of each record to be scanned. If m is specified as *, column 1 is assumed by default.

n

Indicates the last column of the zone of each record to be scanned. If n is specified as *, the length of the record is assumed by default.

The ZONE settings are used by the ALTER, CHANGE, and LOCATE subcommands to define the columns that will be scanned. If no operand is entered, the current settings are displayed. The defaults for the starting and ending zones are:

Filetypes	Start Zone	End Zone
ASSEMBLE, UPDATE, MACRO, UPDTxxxx	Column 1	Column 71
FORTRAN, COBOL, PLI, PLIOPT	Column 1	Column 72
BASIC, VSBASIC	Column 7	Record Length
FREEFORT	Column 9	Column 81
Others	Column 1	Record Length

Example:

```
edit newfile memo
NEW FILE:
EDIT:
zone
   1  80
zone 10 20
input the zone is now set for columns 10-20

EDIT:
top
TOF:
locate o
the zone is now set for columns 10-20
change /o/*/
the zone is n*w set for columns 10-20
```

Note that the LOCATE and CHANGE subcommands operated on the word now, not the word zone, because scanning started in position 10, not in position one.

? SUBCOMMAND

The ? subcommand displays the last EDIT subcommand executed (or issued in the user input area of a display terminal), except for REUSE or a question mark subcommand. On a display terminal, pressing the ENTER key causes this subcommand to be executed again.

?	

After an X or Y subcommand, the last EDIT subcommand is the subcommand that was executed as a result of issuing the X or Y subcommand.

nnnnn SUBCOMMAND

The nnnnn subcommand enters lines by number, while in EDIT mode, into the file, or searches the file for that line number. This subcommand can be used only if LINEMODE LEFT or RIGHT is in effect for the file.
| If no text is given, the record number nnnnn is made the current line,
| and is displayed if verification is on. INPUT mode is not entered.

nnnnn	[text]

nnnnn
   Indicates a line number between 0 and 99999 if the filetype is BASIC or VSBASIC, or 0 and 99999999 if the filetype is FREEFORT.

text
   Specifies a line of text to be put into the file.

A line number not followed by any text is interpreted as a subcommand to locate the line with that number. If such a line is found, it is displayed (unless verification is off). If the line is not found, the line pointer is reset to the previous line and LINE NOT FOUND is displayed. If LINEMODE is LEFT and the filetype is BASIC or VSBASIC, the line number is assumed to be in columns 1 through 5, right justified with leading blanks. If LINEMODE is RIGHT or OFF, the line number is assumed to be in columns 76 through 80, right justified with leading zeros.

A line number followed by text inserts the text into the file if the line does not already exist, or replaces the existing line in the file.

With LINEMODE LEFT, the line number is padded with blanks on the left (if necessary) and inserted in columns 1 through 5. The text is placed on the same line at the first tab position (column 7). The entire reformatted line is placed in the file in the location appropriate to its line number. If a line with that number already exists, it is replaced. Line numbers with leading zeros are not accepted.

With LINEMODE RIGHT, the line number is padded with zeros on the left (if necessary) and inserted in columns 76 through 80. The text is placed on the same line at the first tab position (normally column 1). The entire reformatted line is placed in the file at the location appropriate to its line number. If a line with that number already exists, it is replaced. Line numbers with leading zeros are accepted.

For FREEFORT files, the same procedures are followed, but the line number occupies columns 1 through 8 or 73 through 80.

With LINEMODE OFF, the nnnnn subcommand is invalid.

The nnnnn subcommand does not operate with variable-length files.

Section 4: Edit Macros

An EDIT macro defines a sequence of EDIT subcommands that can be executed by issuing the macro name while in EDIT mode.

EDIT macros are CMS EXEC files that allow you to create complex sequences of EDIT subcommands. See the _VM/370: EXEC User's Guide_ for additional information on creating and invoking EXEC files.

The following EDIT macros are supplied with VM/370 for your convenience. See "Appendix B: User-Written EDIT Macros" for samples of user-supplied macros.

```
| $DUP   | [1|n]                                                |
```

Duplicates the current line _n_ times. The last copy of the line becomes the new current line. If _n_ is omitted, the line is duplicated once.

```
| $MOVE  |  n {Up m|Down m|To label}                            |
```

Moves _n_ lines (starting with the current line) up or down _m_ lines; or moves _n_ lines (starting with the current line) and inserts them after the specified label, or at the end of the file if the label is not found. The last line moved becomes the new current line. The label to be found should start in column 1, and should not contain any lower-case letters. Truncations may be used for UP, DOWN, and TO.

Up to 25 lines can be moved or duplicated with one $MOVE or $DUP macro.

Note: The $DUP and $MOVE subcommands delete any existing stacked lines when invoked.

To display the $DUP or $MOVE macros, type:

```
type $dup exec *

  -- or --

type $move exec *
```

WRITING EDIT MACROS

If you have a good knowledge of the CMS EXEC facilities, you can write your own EDIT macros. You must ensure that any EDIT macro you write checks the validity of its operands and displays an error message if necessary.

The conventions followed when creating EDIT macros are:

1. EXEC files that are EDIT macros have a filename that starts with a dollar sign ($) and a filetype of EXEC. These files are referred to as EDIT macro files.

2. An EDIT macro subcommand consists of the name of an EDIT macro file (including the initial $), possibly followed by operands.

3. An EDIT macro file contains only EDIT subcommands and EXEC control statements. EDIT macros can execute only in EDIT mode.

Operands of an EDIT macro must be separated from the macro name, and from each other, by at least one blank. Percent signs (%) cannot be entered as operands, since they have a special meaning to the EXEC interpreter. Operands passed to an EDIT macro are subject to the same rules as any other EXEC file (that is, the length of an operand must not exceed eight characters).

When you create the macro, IMAGE mode must be off if you include tab characters (X'05').

All EDIT subcommands in EDIT macros must be stacked (that is, you must specify &STACK or &BEGSTACK before the EDIT subcommands). If your EDIT macro uses variables, you should use &STACK rather than &BEGSTACK, since &BEGSTACK inhibits substitution of variables.

If an EDIT macro is issued, and the EDIT macro file does not exist, the Editor issues the message ?EDIT:. If an EDIT macro is used incorrectly, the Editor displays a message, and the macro is ignored. If an EDIT macro is assigned to X or Y, it is an error to issue that X or Y subcommand with a numeric operand other than 0 or 1.

Some EDIT macros use the CMS function DESBUF during their execution (for example, $DUP and $MOVE). If stacked lines exist when one of these macros is invoked, the macro deletes the stacked lines and issues the message STACKED LINES CLEARED BY (macro name). This also occurs in user-written macros if the CMS line end character has been used to stack additional subcommands after the macro is issued.

A user-written EDIT macro that uses first-in, first-out (FIFO) stacking should ensure that the stack is initially clear. You do this by including in your EDIT macro the line

```
   &IF &READFLAG EQ STACK DESBUF
```

before you stack anything. The DESBUF function clears the console stack. Alternatively, your EDIT macro can use last-in, first-out (LIFO) stacking to avoid having to initially clear the console stack.

If the operation of an EDIT macro is completed without an error, the Editor clears any stacked lines and issues the message STACKED LINES CLEARED. Thus, the macro has no effect on the Editor or its contents.

To avoid having the Editor type during execution of your EDIT macros, you can specify that your EDIT macros operate with verification off. You can accomplish this without losing your setting by stacking PRESERVE and VERIFY OFF for execution first, and RESTORE for execution last.

Do not interrupt the execution of an EDIT macro by pressing the ATTN (attention) key or its equivalent.

Section 5: Operational Characteristics

NUMBER OF RECORDS HANDLED BY THE EDITOR

| Figure 5 shows the approximate number of records, rounded to the nearest | hundred, that the Editor can handle in different amounts of virtual storage.

| These numbers are for a CMS system with only one disk accessed.

Record Length	Virtual Machine Storage Size			
	320K	512K	768K	1024K
80 Char	1700	3800	6800	9800
120 Char	1100	2600	4700	6800
132 Char	1100	2400	4300	6200
160 Char	900	2000	3600	5100

Figure 5. Number of Records Handled by the Editor

TOP-OF-FILE AND END-OF-FILE CONDITIONS

At the top of every file is a null line. It is not a part of the file on disk, but is present only during editing, and cannot be modified. If displayed, it appears as a null line. There is also an end-of-file (EOF) condition, which arises when you attempt to advance the line pointer beyond the last line of the file. "EOF:" is displayed.

When issued at the top of the file or at the end of the file, the EDIT subcommands behave as follows:

Subcommand	Effect at the Top of the File
ALTER, CHANGE	If the subcommand applies to the top line only, the message NOT FOUND is issued. If it applies to more than one line, the number is decremented by one before proceeding to the next line.
DELETE	Processed as though it is successful, but the top line is not deleted.
| DSTRING	If the string is in the first data line of the file, that line becomes the current line. If the string is in any other line, all lines before that line are deleted.
GETFILE	The new lines obtained are inserted at the top of the file, and the last line that is inserted becomes the current line.
DOWN, NEXT, TYPE, STACK	The top line is processed as a blank line.

OVERLAY	If OVERLAY applies to the top line only, it has no effect. If it applies to more than one line, the number is decremented by one before proceeding to the next line.
REPLACE	The new line is inserted and becomes the current line.
TOP, UP	Have no effect.

<u>Subcommand</u>	<u>Effect at the End of the File</u>
ALTER, CHANGE	When verification is off, the response is NOT FOUND. When verification is on, the response is NOT FOUND, followed by EOF:. The EOF condition is not cleared.
FIND, LOCATE	Processed as though they are issued at the top of the file.
BOTTOM	The EOF condition is cleared, and the last line of the file becomes the current line.
DELETE, DOWN, NEXT, OVERLAY STACK	If verification is off, the subcommand is ignored; if on, "EOF:" is displayed.
DSTRING	The response is STRING NOT FOUND, NO DELETIONS MADE. The current line remains at EOF:.
GETFILE	The EOF condition is cleared, the new lines obtained are inserted after the last line of the file, and the last line obtained becomes the current line.
TYPE	"EOF:" is displayed.
INPUT, REPLACE	The EOF condition is cleared, and the new lines are inserted after the last line.
UP	Processed as though the current line is a null line following the last line of the file.

<u>PLACING A CONTINUATION CHARACTER IN COLUMN 72</u>

Several methods of inserting a continuation character in column 72 of Assembler Language source records are available. They are:

1. Specify the LINEMODE RIGHT subcommand, which causes the zone setting to default to columns 2 and 72. This allows a continuation character to be placed in column 72, but does not permit column 1 to be used for ALTER, CHANGE, or LOCATE subcommands, unless the zone is reset to column 1 by specifying ZONE 1 *.

2. Use the default zone settings for ASSEMBLE files (columns 1 and 71), and invoke an EDIT macro similar to the user-supplied macro $MARK (see "Appendix B: User-Written EDIT Macros").

3. Redefine the default zone setting from columns 1 and 71 to columns 1 and 72. The default zone settings have been established for columns 1 and 71 to minimize inadvertent data entries in column 72.

Section 6: Error Conditions and Recovery Procedures

EDITOR ERROR PROCEDURES

All EDIT subcommands are fully checked for validity. When an invalid subcommand or macro is not processed, the count for the REPEAT subcommand is reset to 1, and the message

```
¬ or ?EDIT:
```

is issued. If a valid subcommand or macro fails during execution, a more detailed message is displayed.

During an editing session, the Editor displays status information about current subcommands. These messages do not have an associated error code and do not indicate a condition serious enough to require termination of Editor processing. Some messages are to inform you of actions taken or to confirm normal Editor operation.

For your convenience, a complete list of responses and informative messages is provided under "Editor Messages" in this section. For a more complete description of the messages, see the VM/370: System Messages.

EXCEEDING VIRTUAL STORAGE CAPACITY

If the available storage of your virtual machine becomes full, the Editor issues the message:

```
AVAILABLE STORAGE IS NOW FULL
```

It is then possible to add new lines only if existing ones are deleted first. If you attempt to insert a line (or a file) when there is no room, the Editor responds:

```
NO ROOM
```

ensures that EDIT mode is set, deletes any stacked lines, and if necessary issues the message:

```
STACKED LINES CLEARED
```

to avoid multiple error messages if any lines are cleared (see the description of the STACK subcommand).

You can use the CP command DEFINE STORAGE to temporarily increase the size of your virtual storage.

EXCEEDING DISK STORAGE CAPACITY

In executing a FILE, SAVE, or AUTOSAVE subcommand, the Editor writes a temporary work file, called EDIT CMSUT1. During this process, if the output disk becomes full, CMS displays the message:

```
DMSBWR170S DISK 'mode (cuu)' IS FULL
```

The Editor immediately erases the work file (which is incomplete), and issues the message:

```
SET NEW FILEMODE, OR ENTER CMS SUBSET AND CLEAR SOME SPACE
```

The original file remains intact with no updates. Thus, the FILE, SAVE, or AUTOSAVE subcommands have no effect except to produce the messages. A similar response follows an attempt to create more than the maximum number of CMS files.

When enough space is available on the disk to hold the EDIT CMSUT1 work file, the old copy of the file (if one exists) is erased, and the EDIT CMSUT1 file is then renamed to the required filename and filetype. The updated file thus replaces any old copy on the disk. This is the reason that a file with mode=2 can, in effect, be edited. It is not modified, but completely replaced by the utility file.

UNPLANNED TERMINAL SESSION ENDING

If there are data transmission errors on the communication line connecting the terminal to the central computer, or if the central computer is shut down, the terminal is automatically placed in disconnected status for 15 minutes. If you log back on within that 15-minute interval, you can resume where you left off. Otherwise, you are automatically logged off. (In the case of a central computer shutdown, active files are saved up to the point of the last issued FILE, SAVE, or AUTOSAVE subcommand.)

If the terminal session was ended due to a central computer shutdown, communication cannot be reestablished until the central computer is operational. To resume the terminal session, issue the LOGON command.

EDITOR MESSAGES

_SAVED

Explanation: An automatic save (AUTOSAVE) was just performed on the file currently being edited.

AVAILABLE STORAGE IS NOW FULL

Explanation: The size of the file cannot be increased. Any attempt to add lines produces the message NO ROOM. Other commands are unaffected. To continue editing, you may temporarily increase the size of your virtual machine by issuing the CP command DEFINE, or split the file into two smaller ones.

EDIT:

Explanation: Indicates entry to EDIT mode. During initialization, if the file identification specified in the EDIT command is found on disk, this is the first response; otherwise, the file is new and the message NEW FILE precedes EDIT:. Also, during the editing session, this message indicates a return from INPUT mode, or indicates a null line while in EDIT mode, subject to the setting of the VERIFY subcommand.

END ZONE SET TO 72

Explanation: The SERIAL subcommand was issued and the second zone specification was set within the serialization field. The end zone is reset to column 72.

EOF:

Explanation: The line pointer is positioned after the bottom line of the file or, if the file is empty, after the null line at the top of the file (subject to the setting of the VERIFY subcommand).

EOF REACHED

Explanation: The number of lines beyond the starting line specified in a GETFILE subcommand exceeded the end of the indicated file. The lines from the starting line to the end of the file were inserted in the file. When verification is on, the last line inserted is displayed at the terminal.

FILE IS EMPTY

Explanation: An attempt to FILE, SAVE, or AUTOSAVE a null file was detected. If the subcommand was FILE, the Editor exits; if it was SAVE or AUTOSAVE, control returns to EDIT mode. In either case, the file is not stored on your disk.

FILE NOT FOUND

Explanation: The file identification specified in a GETFILE subcommand was not found on an auxiliary storage device.

GETFILE IS INCOMPLETE

Explanation: The available storage was exceeded while attempting to execute a GETFILE subcommand. The last line inserted into the file is displayed at the terminal.

GIVEN STARTING LINE IS BEYOND EOF

Explanation: The starting line specified in a GETFILE subcommand points beyond the last line of the indicated file.

NON-NUMERIC CHARACTER IN LINE NUMBER COLUMNS

Explanation: A nonnumeric character was found in the columns reserved for line numbers. The line pointer identifies the line in error. You should correct or delete the line in error.

INPUT:

Explanation: Indicates entry to INPUT mode; lines entered at the terminal become part of the file.

INVALID LINE NUMBER REFERENCE IN STMNT nnnnn

Explanation: This message occurs for VSBASIC files only. The line number referenced in statement nnnnn is invalid (not numeric). The old line number is nnnnn. The Editor terminates the RENUM subcommand without renumbering the file. To continue, correct statement nnnnn and reissue the subcommand.

INVALID SYNTAX IN STMNT nnnnn

Explanation: This message occurs with VSBASIC files only. RENUM cannot convert the line number operand in statement nnnnn due to incorrect language usage. The old line number is nnnnn. The Editor terminates the RENUM subcommand. To continue, correct the statement in line nnnnn and reissue the command.

INVALID $name PARAMETER LIST

Explanation: The indicated EDIT macro was invoked with one or more errors in the subcommand line.

LINE xxxxx REFERENCED IN STMNT nnnnn, NOT FOUND

Explanation This message occurs for VSBASIC files only. The line number specified as an operand in statement nnnnn, was not found. The old line number is nnnnn. The Editor terminates the RENUM subcommand. To continue, correct the line number operand xxxxx in statement nnnnn and reissue the command.

MAXIMUM LINE NUMBER EXCEEDED

Explanation: The RENUM subcommand specified values for strtno and incrno which would result in a line number that exceeds 99999 for VSBASIC files or 99999999 for FREEFORT files. The Editor terminates the RENUM subcommand. To continue, reissue RENUM with proper strtno and incrno values.

This message is also issued for other serialized files if the line number exceeds 99999. The file must be reserialized.

NEW FILE:

Explanation: The message is issued during Editor initialization if the file identified in the EDIT command is not found on any disk to which the user has read or write access.

NO LINES MOVED

Explanation: The EDIT macro $MOVE was invoked with number of lines to be moved equal to 0.

NO ROOM

Explanation: An attempt to enter additional lines to a file has been detected after the "full storage" message was typed. Any stacked lines are cleared to avoid multiple error messages or improper subcommand execution sequences. At this point, you must either split the file into two smaller files or temporarily increase the storage size of your virtual machine via the CP DEFINE STORAGE command. The maximum virtual storage permitted is determined by the MSTOR value in your directory entry.

NOT FOUND

Explanation: The search operand specified in the ALTER, CHANGE, FIND, or LOCATE subcommand was not encountered in the delimited range (current ZONE setting), or before the end of the file was reached.

OVERFLOW AT STATEMENT nnnnn

Explanation: This message occurs with VSBASIC files only. The conversion of the line number operand in statement nnnnn would produce a record exceeding the logical record length. The old line number is nnnnn. The Editor terminates the RENUM subcommand; to continue, correct the statement at old line number nnnnn and reissue the subcommand.

READ ERROR - GETFILE IS INCOMPLETE

Explanation: An unrecoverable error was encountered during the execution of a GETFILE subcommand. The last line inserted into the file is displayed at the terminal.

RECORD LENGTH OF FILE TOO LARGE

Explanation: The file identification of a GETFILE subcommand indicates a file with a record length greater than the file being edited. The GETFILE subcommand is not executed.

RENUM MODULE NOT FOUND

Explanation: The RENUM subcommand requires that there be a RENUM module on the system disk. The Editor terminates the RENUM subcommand. Your installation system programmer must place the RENUM module on the system disk.

RENUMBER LINES

Explanation:

1. The line number prompter cannot proceed because there are no more numbers between the current line number and the line number of the next line already in the file (that is, they differ by one). In LINEMODE RIGHT, the user can turn LINEMODE OFF, issue a SERIAL subcommand, SAVE the file on disk (reserializing it), and finally turn LINEMODE RIGHT on and continue with the editing session.
2. The next line number, 100000000 or 100000, is too large.
3. If you are editing a VSBASIC or FREEFORT file, you can use the RENUM subcommand to renumber your file.

RESERIALIZATION SUPPRESSED

Explanation: Reserialization on a SAVE, AUTOSAVE, or FILE subcommand is suppressed when LINEMODE RIGHT is set so that the numbers used during the editing session are retained. To reserialize, repeat the SAVE, AUTOSAVE, or FILE with LINEMODE OFF set.

SAVED (See _SAVED.)

SERIALIZATION IS INCOMPLETE

Explanation: During the execution of a SAVE, AUTOSAVE, or FILE subcommand that is serializing a file, the disk becomes full before the last line is written. the partial file is erased and the user is notified of the condition.

SET NEW FILEMODE, OR ENTER CMS SUBSET AND CLEAR SOME SPACE

Explanation: During the execution of a SAVE, RENUM, AUTOSAVE, or FILE subcommand, the disk becomes full before writing the last line of the file. The Editor erases the partial file. To continue, either (1) alter the Edit file with the FMODE subcommand, or (2) enter CMS SUBSET and erase unneeded files to make more room available.

SET NEW FILEMODE AND RETRY

Explanation: An attempt was made to SAVE, AUTOSAVE, or FILE a file on a disk that is read-only or not accessed. The user may reissue the subcommand using a read/write disk as the filemode specification in the subcommand line; or, if he does not have a read/write disk active, he may enter the CMS SUBSET environment by issuing the subcommand CMS, then issue the ACCESS command to gain access to a disk in read/write status, then return to the EDIT environment by issuing the RETURN command.

If you are using a VSBASIC file and issued a RENUM subcommand, you must access the disk you specified in read/write status for the subcommand to operate. The Editor terminates the RENUM subcommand without renumbering the file. To continue, use the FMODE subcommand to direct the file to a read/write disk and reissue the RENUM subcommand.

SET NEW FILENAME AND RETRY

Explanation: During the execution of a SAVE, AUTOSAVE, or FILE subcommand, an error occurred while altering the name of the CMS work file. You can now institute recovery procedures, since the Editor returns to EDIT mode. The work file remains. It should be erased, and a different file identification for a subsequent SAVE, AUTOSAVE, or FILE subcommand should be specified.

STACKED LINES CLEARED

Explanation: Multiple subcommands were detected after a failure to increase the file size when the Editor had indicated NO ROOM. This message is also displayed when an abnormal exit from the EDIT mode occurs (to preserve the CMS command environment from stacked EDIT subcommands), or when an error is encountered in executing an EDIT macro.

STACKED LINES CLEARED BY $name

Explanation: When the named EDIT macro (such as $MOVE) is invoked, any stacked lines are cleared by the macro before its execution. This message also occurs at the top of the file or the end of the file. At any other point in the file, the message does not occur unless lines are stacked in the input buffer.

| STRING NOT FOUND, NO DELETIONS MADE

| Explanation: The specified character string has not been found by
| the end of the file. No deletions have been made, and the current
| line pointer remains unchanged.

TOF:

Explanation: The current line pointer is positioned at the null line at the top of the file. This message appears either after the TOP subcommand has been issued or after any other EDIT subcommand has positioned the line pointer at the null line at the beginning of the file.

TOO MANY LINES TO MOVE

Explanation: The $MOVE EDIT macro was invoked with the number of lines to be moved greater than 25.

TOO MANY LINES TO STACK

Explanation: During initialization, the parameter of the STACK subcommand implies a storage requirement in excess of that reserved for the execution of the subcommand. The limit is 25 lines.

TRUNC SET TO 72

Explanation: The SERIAL subcommand was issued and the truncation column was set within the serialization field. The truncation column is reset to column 72.

TRUNCATED

Explanation: The current line has exceeded the truncation column. If verification is on, the truncated line is displayed, followed by the message INPUT: (if in INPUT mode).

WRONG FILE FORMAT FOR LINEMODE RIGHT

Explanation: The LINEMODE RIGHT option is not compatible with variable-length files or files that have a fixed record length other than 80.

WRONG FILE FORMAT FOR RENUM

Explanation: The filetype of the file you are editing is not VSBASIC or FREEFORT, or the Editor detected an invalid line number. For VSBASIC files, the line number must be the first five characters of the record. For FREEFORT files, the line number must be the first eight characters of the record. The Editor terminates the RENUM subcommand without renumbering the file. To continue, correct the line number or filetype and reissue the RENUM subcommand.

WRONG FILE FORMAT FOR SERIALIZATION

Explanation: The SERIAL subcommand was issued for a variable-length file or for a file that does not have a fixed record length of 80.

ZONE ERROR

Explanation: The string specified in a CHANGE subcommand is too long for the current zone specification. The file is not changed.

¬

Explanation: Same as ?EDIT:, but the input line is not displayed because the SHORT subcommand is in effect.

¬$

Explanation: Same as ?EDIT:, but is displayed when an invalid EDIT macro is issued and the SHORT subcommand is in effect.

?EDIT:

Explanation: An unrecognizable EDIT subcommand or invalid subcommand operand has been encountered. The input line is displayed for inspection. This form is used if the LONG subcommand is in effect.

DMSBWR170S DISK 'mode (ccu)' IS FULL

Explanation: CMS issues this message if the output disk being filled by a FILE, SAVE, RENUM, or AUTOSAVE subcommand becomes full. The Editor terminates the subcommand, erases the work file (which is incomplete), and requests the user to specify a new filemode or make more room on the disk.

Appendix A: Summary of Edit Subcommands

Subcommand	Operands
ALter	{parm1}{parm2} [1|n|* [G|*]]
AUTOsave	[n|OFF]
BAckward	[1|n]
Bottom	
CASE	[M|U]
Change	/string1/string2[/ [1|n|* [G|*]]]
CMS	
DELete	[1|n|*]
DOwn	[1|n]
Dstring	/string[/]
FILE	[filename [filetype [filemode]]]
Find	[line]
FMode	[filemode]
FName	[filename]
FORMat	{DISPLAY|LINE}
FOrward	[1|n]
Getfile	filename [filetype|* [filemode|* [1|m [n |*]]]]
IMAGE	[ON|OFF|CANON]
Input	[line]
LINEmode	[Left|Right|OFF]
Locate	/string[/]
LONG	
Next	[1|n]
Overlay	line
PREserve	
PROMPT	[n]
QUIT	

Subcommand	Operands
RECfm	[F|V]
RENum	[strtno|10 [incrno|strtno]]
REPEAT	[1|n|*]
Replace	[line]
REStore	
RETURN	
{REUSE|=}	[subcommand]
SAVE	[filename [filetype [filemode]]]
S[croll[Up]	[*|n|1]
SERial	{OFF|ON|ALL|seq}[incr|10]
SHORT	
STACK	[1|n|subcommand]
TABSET	{n_1 n_2 ... n_x }
TOP	
TRUNC	[n|*]
Type	[1|m|* [n|*]]
Up	[1|n]
Verify	[ON|OFF][[startcol|1]endcol|*]
{X|Y}	[1|n|subcommand line]
Zone	[1|m|* [n|*]]
?	
nnnnn	[text]

EDIT MACROS

Macro	Operands
$DUP	[1|n]
$MOVE	n {Up m|Down m|To label}

Appendix B: User-Written Edit Macros

The user-supplied EDIT macros in this appendix have not been formally tested by IBM; they are presented for your convenience only. See the VM/370: EXEC User's Guide for a detailed description of the EXEC facilities illustrated in this appendix. The $MOVE and $DUP EDIT macros are not illustrated here since they are distributed with the CMS system, and can be displayed at your terminal by keying in:

```
   type $move exec *
```

and

```
   type $dup exec *
```

or simply:

```
|  list $* EXEC * (EXEC#CMS TYPE
```

which results in all EXEC files with a filename that starts with $ being displayed.

$MACROS

The $MACROS EDIT macro enables you to key in "$MACROS filename" to verify the existence of any files that have a filename beginning with $ on your system. If the specified filename is present, full identifications of all files with that filename are displayed. If no operands are specified, all existing files whose filenames begin with $ are displayed. If $MACROS ? is entered, the macro usage is explained.

$MACROS	[filename1 [filename2 [filenamen]]]

filename1, 2, or n
   The filename or filenames of EDIT macros.

To create $MACROS, enter:

```
   edit $macros exec
```

and in INPUT mode, enter the following:

```
   &CONTROL OFF
   &IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
   &IF &INDEX GT 0 &GOTO -PARTIC
   *
   &BEGTYPE ALL
   EXEC FILES STARTING WITH A DOLLAR-SIGN ARE AS FOLLOWS.
   FOR INFORMATION ON ONE OR MORE OF THEM, TYPE:
   $MACROS FILENAME1 <FILENAME2>
   &END
   LISTF $* EXEC * (NOHEADER FNAME)
   &EXIT
```

```
*
-PARTIC &TRIP = 0
&INDEX1 = 0
*
&LOOP -ENDLOOP &INDEX
&INDEX1 = &INDEX1 + 1
&SUB = &SUBSTR &&INDEX1 1 1
&IF &SUB EQ $ &GOTO -STATIT
&TYPE &&INDEX1 IS INVALID
&TRIP = 1
&GOTO -ENDLOOP
-STATIT STATE &&INDEX1 EXEC *
&IF &RETCODE EQ 0 &GOTO -CALLIT
&TYPE &&INDEX1 NOT FOUND
&TRIP = 1
&GOTO -ENDLOOP
-CALLIT EXEC &&INDEX1 ?
-ENDLOOP
*
&EXIT &TRIP
*
-TELL &BEGTYPE
'$MACROS' HANDLES THE '$MACROS' REQUEST.
TYPE '$MACROS' ALONE FOR MORE INFORMATION.
&END
&EXIT
```

<u>$MARK</u>

The $MARK EDIT macro allows you to insert from one to six characters, starting with the current line and in the column specified, for <u>n</u> records. If $MARK ? is entered, the macro usage is explained.

$MARK	[<u>1</u>|n [<u>72</u>|col [<u>*</u>|char]]]

n
: Indicates the number of consecutive lines, starting with the record currently being pointed to, that will have a character or characters inserted. If <u>n</u> is not specified, 1 is assumed for <u>n</u>, and the other default values are also assumed.

col
: Indicates the starting column in each record where the character string is to be inserted. The default is column 72.

char
: Indicates from 1 to 6 characters to be inserted in each record. The default is an asterisk (*).

To create $MARK, enter:

```
edit $mark exec
```

and in INPUT mode, enter the following:

```
&CONTROL OFF
&IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
&IF &INDEX GT 3 &GOTO -BADPARM
*
&INDEX1 = 1
&IF &INDEX GT 0 &INDEX1 = &1
&IF &INDEX1 LT 0 &GOTO -BADPARM
&INDEX2 = 72
&IF &INDEX GT 1 &INDEX2 = &2
&IF &INDEX2 LT 0 &GOTO -BADPARM
&IF &INDEX2 GT 133 &GOTO -BADPARM
&CHAR = *
&IF &INDEX EQ 3 &CHAR = &3
&LEN3 = &LENGTH &CHAR
&IF &LEN3 GT 6 &GOTO -BADPARM
*
&STACK LIFO RESTORE
&STACK LIFO OVERLAY (TAB)[1] &CHAR
&STACK LIFO REPEAT &INDEX1
&STACK LIFO TABS &INDEX2
&BEGSTACK LIFO
IMAGE ON
TRUNC *
VERIFY OFF
LONG
PRESERVE
&END
&EXIT
*
-BADPARM &BEGTYPE
INVALID $MARK OPERANDS
&END
&EXIT 1
*
-TELL &BEGTYPE
CORRECT FORM IS:  $MARK  <N  <COL  <CHAR>>>
PUTS A 1-6 CHARACTER  STRING IN COLUMN 'COL' OF 'N' LINES, STARTING
WITH THE CURRENT LINE.  THE NEW CURRENT LINE IS THE LAST LINE MARKED.
DEFAULTS ARE:  N=1;  COL=72;  CHAR=*.
&END
&EXIT
```

----

[1]The word (TAB) represents pressing the tab key (or equivalent logical tab) and should not be included in the data line. Instead, enter the appropriate tab character.

$POINT

The $POINT EDIT macro allows you to scan only the columns 73-80 of each record in the file, starting with the first record, for a match on a one- to eight-character key. The current ZONE and TRUNC subcommand settings are ignored. If $POINT ? is entered, the macro usage is explained.

$POINT	key

key
   Contains a one- to eight-character field. If the specified key is less than eight characters long, it is padded with leading zeros.

To create $POINT, enter:

```
edit $point exec
```

and in INPUT mode, enter the following:

```
&CONTROL OFF
&IF &INDEX EQ 0 &GOTO -TELL
&IF &INDEX EQ 1 &IF &1 EQ ? &GOTO -TELL
&IF &INDEX GT 1 &GOTO -BADPARM
*
&KEYL = &LENGTH &1
&INDEX1 = 8 - &KEYL
&Z = &SUBSTR 00000000 1 &INDEX1
&1 = &CONCAT &Z &1
*
&STACK LIFO RESTORE
&STACK LIFO FIND (TAB)[1]   &1
&BEGSTACK LIFO
TOP
TABS 1 72
IMAGE ON
LONG
PRESERVE
&END
&EXIT
*
-BADPARM &BEGTYPE ALL
INVALID $POINT OPERANDS
&END
&EXIT 1
*
-TELL &BEGTYPE ALL
CORRECT FORM IS:  $POINT  KEY
IF 'KEY' CONTAINS LESS THAN 8 CHARACTERS, IT IS PADDED WITH LEADING
ZEROS. THE FILE IS THEN SEARCHED FROM THE TOP FOR 'KEY' IN COLUMNS
73-80.
&END
&EXIT
```

----

[1]The word (TAB) represents pressing the tab key (or equivalent logical tab) and should not be included in the data line. Instead, enter the appropriate tab character.

$COL

The $COL EDIT macro allows you to insert, after the current record pointed to, a line containing column numbers (that is, 1, 6, 11, ..., 76). If any operands are entered, the macro usage is explained.

```
| $COL |
```

No operands are used with $COL.

To create $COL, enter:

```
edit $col exec
```

and in INPUT mode, enter the following:

```
&CONTROL OFF
&IF &INDEX NE 0 &GOTO -TELL
&STACK LIFO RESTORE
&STACK LIFO
&BEGSTACK LIFO ALL
1    6    11   16   21   26   31   36   41   46   51   56   61   66   71   76
&END
&STACK LIFO INPUT
&BEGSTACK LIFO
TRUNC *
VERIFY OFF
LONG
PRESERVE
&END
&EXIT
*
-TELL &BEGTYPE
CORRECT FORM IS:  $COL
INSERTS A LINE INTO THE FILE SHOWING COLUMN NUMBERS.
&END
&EXIT
```

Index

Trim Along This Line

READER'S COMMENTS

Title: IBM Virtual Machine Facility/370: EDIT Guide

Order No. GC20-1805-3

Please check or fill in the items; adding explanations/comments in the space provided.

Which of the following terms best describes your job?

☐ Programmer	☐ Systems Analyst	☐ Customer Engineer
☐ Manager	☐ Engineer	☐ Systems Engineer
☐ Operator	☐ Mathematician	☐ Sales Representative
☐ Instructor	☐ Student/Trainee	☐ Other (explain below)

Does your installation subscribe to the SL/SS? ☐ Yes ☐ No

How did you use this publication?

☐ As an introduction ☐ As a text (student)
☐ As a reference manual ☐ As a text (instructor)
☐ For another purpose (explain) ____________

Did you find the material easy to read and understand? ☐ Yes ☐ No (explain below)

Did you find the material organized for convenient use? ☐ Yes ☐ No (explain below)

Specific criticisms (explain below)
Clarifications on pages ____________
Additions on pages ____________
Deletions on pages ____________
Errors on pages ____________

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GC20-1805-3

Trim Along This Line

YOUR COMMENTS PLEASE . . .

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance and/or additional publications or to suggest programming changes will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.* Your comments will be carefully reviewed by the person or persons responsible for writing and publishing this material. All comments or suggestions become the property of IBM.

FOLD FOLD

FIRST CLASS
PERMIT NO. 172
BURLINGTON, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
VM/370 PUBLICATIONS
24 NEW ENGLAND EXECUTIVE PARK
BURLINGTON, MASS. 01803

FOLD FOLD

IBM VM/370: EDIT Guide Printed in U.S.A. GC20-1805-3

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM Technical Newsletter

This Newsletter No.	GN20-2660
Date	March 31, 1975
Base Publication No.	GC20-1805-3
File No.	S370-36
Previous Newsletters	None

IBM Virtual Machine Facility/370:
EDIT Guide

This Technical Newsletter, a part of Release 2 PLC 13 of IBM Virtual Machine Facility/370, provides replacement pages for your publication. These replacement pages remain in effect for subsequent VM/370 releases unless specifically altered. Pages to be removed and/or inserted are listed below.

Title Page, 2	47-50.2
Summary of Amendments	53-54.1
Contents (2 pages)	63,64
5-10.4	67-70.1
11,12	71,72
17-18.1	75-78
19,20	83-84.1
31,32	85,86
39-44.1	

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

SUMMARY OF AMENDMENTS

This Technical Newsletter incorporates changes reflecting support for remote 3270 Display Stations and enhancements to the CMS Editor.

Note: Please file this cover letter at the back of your publication to provide a record of changes.

IBM Corporation, VM/370 Publications, 24 New England Executive Park, Burlington, Massachusetts 01803

GC20-1805-3

IBM VM/370: EDIT Guide Printed in U.S.A. GC20-1805-3

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)