CP-EDIT
User's Guide and Reference
Version 5.23

Joseph A. Thompson

THOMPSNJ at WMAVM7 (IBM Internal email)
thompsnj@wmavm7.vnet.ibm.com (Internet)
Loral Federal Systems -- Gaithersburg
700 N. Frederick Ave.
Gaithersburg, MD 20879

Contents

Document Notes 1

Introduction 3
Machine Requirements 3
Software Requirements 4
Getting Started 4
Quick Start for Using CP-Edit 5

Invocation of CP-Edit 7

Editing files using CP-Edit 9
Editing Using Full-Screen Features of CP-Edit 9
Editing Using Prefix Commands 15
Editing Using CP-Edit Commands 31
CP-Edit Targets 33
Entering Text 35
Using Function Keys 39
CP-Edit Macros 40
  REXX Macros 40
  CP-Edit Native Macros 42
  The Profile Macro 42
Putting it All Together 43

CP-Edit Commands 45
Syntax 45
File Names 45
ADD COMMAND 47
AUTOIN COMMAND 48
BACKWARD COMMAND 49
BOTTOM COMMAND 50
CANCEL MACRO 51
| CAPPEND MACRO 52
| CDELETE COMMAND 53
| CFIRST COMMAND 54
CHANGE COMMAND 55
| CINSERT COMMAND 57
| CLAST COMMAND 58
| CLOCATE COMMAND 59
CMSG COMMAND 61
COMMAND 62
| COVERLAY COMMAND 63
| CREPLACE COMMAND 64
CURSOR COMMAND 65
| DEFINE COMMAND 66
| Case 66
| Keynames on Left Side of Define Statement 66
| Key Name Aliases 69
| Key Name Usage Notes 70
| Keynames on Right Side of Define Statement 71
| Summary of Keys and Characters 72

DELETE COMMAND 77
DOS COMMAND 78
DOWN COMMAND 81
EDIT COMMAND 82
EMSG COMMAND 83
END COMMAND 84
EXTRACT COMMAND 85
FILE COMMAND 92
FIND COMMAND 93
FORWARD COMMAND 94
GET COMMAND 95
HELP COMMAND 96
HEXTYPE COMMAND 97
INPUT COMMAND 98
JOIN COMMAND 99
LEFT COMMAND 100
LOCATE COMMAND 101
LOWERCAS MACRO 103
MACRO COMMAND 104
MERGE MACRO 105
MSG COMMAND 106
NEXT COMMAND 107
OS COMMAND 108
OVERLAY COMMAND 109
POWERINP COMMAND 110
PUT COMMAND 111
PUTD COMMAND 112
QUERY CASE COMMAND 113
QUERY FREE COMMAND 114
QUERY HEX COMMAND 115
QUERY IMAGE COMMAND 116
QUERY LINEND COMMAND 117
QUERY NBFILE COMMAND 118
QUERY PF COMMAND 119
QUERY PGUP/PGDN COMMANDS 120
QUERY RING COMMAND 121
| QUERY STREAM COMMAND 122
QUERY TABS COMMAND 123
QUERY TRUNC COMMAND 124
QUERY VERIFY COMMAND 125
QUERY VERSHIFT COMMAND 126
QUERY VERSION COMMAND 127
QUERY WRAP COMMAND 128
QUERY ZONE COMMAND 129
QQUIT COMMAND 130
QUIT COMMAND 131
RECOVER COMMAND 132
REPEAT COMMAND 133
RESET COMMAND 134
RIGHT COMMAND 135
SAVE COMMAND 136
SET ALT COMMAND 137
SET AUTOSAVE COMMAND 138
SET CASE COMMAND 139

SET CMDLINE COMMAND 141
SET COLOR COMMAND 142
SET CURLINE COMMAND 144
SET DSPF COMMAND 145
SET ENTER COMMAND 146
SET FNAME COMMAND 147
SET FONT COMMAND 148
SET HEX COMMAND 149
SET IMAGE COMMAND 150
SET KEYBOARD COMMAND 151
SET LINEND COMMAND 152
SET MSGMODE COMMAND 153
SET NUMBER COMMAND 154
| SET PALETTE COMMAND 155
SET PAUSE COMMAND 156
SET PF COMMAND 157
SET PGUP/PGDN COMMANDS 158
SET POINT COMMAND 159
SET PREFIX COMMAND 160
SET READONLY COMMAND 161
SET SCALE COMMAND 162
SET SCREEN COMMAND 164
SET SOUND COMMAND 167
| SET STREAM COMMAND 168
| SET SWAPEXEC COMMAND 169
SET SYNONYM COMMAND 170
SET TABLINE COMMAND 171
SET TABS COMMAND 173
SET TRUNC COMMAND 174
SET VERIFY COMMAND 175
SET WRAP COMMAND 176
SET ZONE COMMAND 177
SI COMMAND 178
SORT COMMAND 179
SOS COMMAND 181
SPLIT COMMAND 182
SPLTJOIN COMMAND 183
STATUS MACRO 184
TOP COMMAND 185
UP COMMAND 186
UPPERCAS MACRO 187
= COMMAND 188
& (Ampersand) 189
* (Asterisk) 190
? COMMAND 191

CP-Edit Prefix Commands 193
A PREFIX COMMAND 194
I PREFIX COMMAND 195
/ PREFIX COMMAND 196
" PREFIX COMMAND 197
M PREFIX COMMAND 198
C PREFIX COMMAND 199
MX PREFIX COMMAND 200

CX PREFIX COMMAND 201
G PREFIX COMMAND 202
SI PREFIX COMMAND 203
< PREFIX COMMAND 204
> PREFIX COMMAND 205
D PREFIX COMMAND 206
P PREFIX COMMAND 207
F PREFIX COMMAND 208
SCALE PREFIX COMMAND 209
TABL PREFIX COMMAND 210
CE PREFIX COMMAND 211
TF PREFIX COMMAND 212
TJ PREFIX COMMAND 213
PREFIX .xxxx (SET SYMBOLIC NAME) 214

Appendix A. Differences Between CP-Edit and XEDIT 215
XEDIT Compatibility 215
Target Limitations 215
Screen and Keyboard Differences 215
Prefix Differences. 216
Profile Differences 217
Miscellaneous Differences and Limitations 217

Appendix B. Default Settings 219
Default Function key settings 219
Default Synonyms 219
Default SET values 219

Appendix C. CP-Edit Messages 223
Informational Messages: 224
Warning and Error Messages: 225

Figures

1. CP-Edit Screen Format 10
2. Screen with Scale and Tab Lines 11
3. KEYDEF Key Names 14
4. KEYDEF XEDIT Key Names 14
5. Block Delete Commands (Before) 16
6. Block Delete Commands (After) 17
7. Multiple Prefix Commands (Before) 18
8. Multiple Prefix Commands (After) 19
9. First Block Delete Command Entered 20
10. Second Block Delete Command Entered 21
11. Block Delete Commands Executed 22
12. Move and Copy Prefix Commands (Before) 23
13. Move and Copy Prefix Commands (After) 24
14. Duplicate Prefix Command (Before) 25
15. Duplicate Prefix Command (After) 26
16. Shift Prefix Commands (Before) 27
17. Shift Prefix Commands (After) 28
18. Insert Prefix Command (Before) 29
19. Insert Prefix Command (After) 30
20. Switching to Input Mode 36
21. Entering Text in Input Mode 37
22. Entering Text in Power Typing Mode 38
23. Text on Exit from Power Typing 38
24. Key Name Aliases (Sorted By Key Name) 69
25. Key Name Aliases (Sorted By Alias) 70
26. ASCII Character Aliases 71
27. Editing Functions 72
28. ASCII Collating Sequence (Left Side of Define Statement - Part I) 73
29. ASCII Collating Sequence (Left Side of Define Statement - Part II) 74
30. ASCII Collating Sequence (Right Side of Statement) - Part I 75
31. ASCII Collating Sequence (Right Side of Statement) - Part II 76
32. CP-Edit Screen with Scale On 163
33. CP-Edit Screen with Tab Line On 172

Document Notes

Since the "not" character used by CP-Edit (the ASCII tilde) is not present in some EBCDIC printer fonts, the character is represented in this document by the following character: ˜. Where you see the character ˜, you should read it as the ASCII tilde, which is the CP-Edit "not" character.

Introduction

CP-Edit is a full-screen text editor which is patterned after several IBM 360/370 family large-computer editors, most notably XEDIT (but including its ancestors, CMS EDIT, NEDIT, CEDIT, EDGAR, SEDIT, the ISPF editor, and others). With its default settings, it most resembles XEDIT running with a PROFILE which makes it look a bit like the ISPF editor. In most cases, where defaults differ from those of XEDIT, the settings can be changed to the XEDIT defaults.

Currently, CP-Edit is available for DOS and OS/2 for PCs and the IBM PS/2, and for AIX for the IBM RISC System/6000.

CP-Edit can be used for creating and revising correspondence and other text, program source text, batch files, and any other type of text files on the IBM PC.

If you are familiar with XEDIT running under IBM's VM/CMS operating system, you will be able to use CP-Edit very easily. And if you learn to use CP-Edit and later want to use XEDIT on a large host computer, you will find the transition very easy. For the convenience of anyone using both CP-Edit and XEDIT, Appendix A, “Differences Between CP-Edit and XEDIT” on page 215 documents the differences between them.

As its name implies, this document contains sections which tell you how to use CP-Edit, and sections which serve as a reference after you become an experienced user of CP-Edit.

If you have questions about CP-Edit that aren't answered by this document, or if you have comments, suggested changes, or a bug report, please direct them to CPEDIT FORUM on the IBMPC conferencing disk.

Some useful macros in addition to those supplied with the package are available in the file CPEDIT MACROS on the IBMPC conferencing disk.

Machine Requirements

The DOS version of CP-Edit requires an IBM PC, PC-XT, PC-AT, Portable PC, 3270-PC, PC jr, PC Convertible, or PS/2 with 512K bytes or more of memory (to edit larger files, more memory is required). (CP-Edit has been tested on several PS/2 models and should run OK on all of them.) One or more diskette drives may be used if installed. One or more fixed disks may also be used with CP-Edit, but are not required.

The DOS version of CP-Edit will operate with either a monochrome or color display in 80 column mode or 40 column mode. CP-Edit operates with the IBM monochrome adapter, the Enhanced Graphics Adapter (EGA), and the VGA[1]. On the EGA or VGA, CP-Edit will use the full length of the screen, any number of lines from 25 to 75.

The OS/2 version of CP-Edit will run on any machine that will run OS/2 1.3 or later.

CP-Edit does not manipulate or switch fonts on the EGA. Some other program, such as EGAPAL, EGAXTRA, EGA Maestro, HOWARD, etc. is required for that.

---

[1] CP-Edit Version 5 and later will operate with the original IBM color display adapter, but the results may be unsatisfactory due to "snow" or "fletching" on the screen during screen updates.

Software Requirements

The DOS version of CP-Edit requires PC-DOS 3.0 or higher. It will not operate correctly with DOS 2.0, 2.1, 1.0 or 1.1.

The DOS version of CP-Edit works correctly in the OS/2 DOS box, for OS/2 1.2, 1.3, and OS/2 2.0.

Getting Started

Current versions of CP-Edit are available from PCTOOLS as CPEDIT5 PACKAGE for DOS and from OS2TOOLS as CPEDOS2 PACKAGE for OS/2. See the PACKAGE files for current contents.

You should add the CPEDIT5.EXE or CPEDVIO.EXE file to a directory in your DOS or OS/2 PATH, or create a directory to hold CP-Edit and add it to your PATH statement (in your AUTOEXEC.BAT for DOS or in your CONFIG.SYS for OS/2). Note that the .EXE file and macros must be either in the current directory or in a directory that is in your PATH somewhere in order for them to be found.

Two sample profile macros are provided in the CPEDMAC.EXE file: SAMPROF.MAC and SAMPROFN.MAC. The SAMPROF.MAC file is a REXX sample profile, and the SAMPROFN.MAC file is a non-REXX sample profile. To use one of them, just rename it to PROFILE.MAC:

```
  RENAME SAMPROF.MAC PROFILE.MAC
or
  RENAME SAMPROFN.MAC PROFILE.MAC
```

You will probably want to edit the profile with CP-Edit and customize it to your liking.

The help files should be installed in a subdirectory of their own, and the name of the directory put in an environment variable so the files can be found by CPEDIT. Make a directory for the help files, for example

```
  MKDIR C:\CPEDHLP
```

Then unpack the help files into that directory:

1. COPY CPEDHLP.EXE C:\CPEDHLP
2. C:
3. CD \CPEDHLP
4. CPEDHLP
5. ERASE CPEDHLP.EXE

Then add the following statement to the end of your C:\AUTOEXEC.BAT file for DOS or C:\CONFIG.SYS file for OS/2 using CPEDIT:

```
  SET CPEDHLP=C:\CPEDHLP
```

If you wish, you may also execute the above SET command from the DOS command line so that the help files will be available immediately. Otherwise, they will be available after the next reboot of your PC.

For compatibility with older versions of CP-Edit, you may still put help files in a directory in your path (for example, the same file that contains the CPEDIT.EXE file and macros), but this is not recommended, since it adds a lot of files to your path and lengthens the time required to search for programs to execute.

The KEYDEF program allows you to customize (to some extent) the way CPEDIT uses the keyboard keys. Currently, KEYDEF is available from PCTOOLS and must run under DOS or in the DOS box of OS/2.

If you want the CPEDIT.EXE file to have a different name, just rename it using the DOS RENAME command. For example:

```
RENAME C:\CPEDIT\CPEDIT5.EXE E.EXE
```

Quick Start for Using CP-Edit

In order to get the full benefit from CP-Edit, you should take the time to read the sections of this document on the use of CP-Edit, and scan the later reference sections, so you will know where to find detailed information about CP-Edit facilities.

But, if you are like most of us, you want to get started right away. For your convenience, here is a summary of how to get into CP-Edit, how to get out, how to save text you have edited, and how to get help.

To start CP-Edit from DOS or a full-screen OS/2 session, you just type in "cpedit" followed by the name of the file you want to edit:

C:> cpedit sample.txt

After the program loads the file and displays the first few lines of it on the screen, you may use the full-screen editing facilities of CP-Edit to change the file, or use the prefix commands to manipulate lines of the file, or use the powerful CP-Edit commands to manipulate the file.

If you need help in using CP-Edit commands and other features, press the F1 key to enter a general help file. This file lists all the CP-Edit commands with their syntax, and lists some of the other features of CP-Edit as well. You can get help on a specific CP-Edit command by typing in "HELP" and the name of the command (or its abbreviation) on the CP-Edit command line:

```
===> HELP LOCATE
```

To exit from any help file, just press the Esc key.

When you have made all your changes to the file, you may save the results and exit CP-Edit by entering the FILE command:

```
===> FILE
```

Or, if you have made no changes and just want to exit back to DOS, you may press the Esc key or use the QUIT command. If you want to exit and discard any changes, you may use the QQUIT command. It is a good idea to get in the habit of quitting using the QUIT command or Esc key, since CP-Edit will not allow you to accidentally QUIT if you made changes to the file.

Invocation of CP-Edit

The CP-Edit program resides on disk as one file, named CPEDIT5.EXE . This file is the only one absolutely required to use CP-Edit. However, a set of Help files (files whose names end with .HLP) is supplied with CP-Edit, and you may wish to have them available in a help directory in case you need to use the HELP command to get help in using CP-Edit. See “Introduction” on page 3 for details on installation of help files. (If the .HLP files are not available, the HELP command will not work.)

If you are using the REXX version, you will probably also want to have the HELP macro (HELP.MAC) available also.

Also, you may wish to use a Profile file (whose default name is PROFILE.MAC) to personalize your editing session. The Profile is described below.

CP-Edit is invoked by typing "CPEDIT". This may be followed by the name of the (first) file to be edited, and, optionally, by several other parameters. The filename is specified in the normal DOS or OS/2 manner.

Others parameters are:

The profile parameter consists of -p and the name(s) of the profile file (optionally preceded by drive and path). Multiple profile names may be specified, separated by semicolons. If multiple profile names are specified, a search is made for each one until one is found. The first one found is the one which is executed.

The User parameter consists of -u followed by arbitrary text. If this text contains blanks or commas, it must be enclosed in double quotes. This text is not interpreted by CP-Edit, but is just passed to the profile macro as an argument when the profile is executed. Only REXX profile macros can recognize arguments.

The User parameter may be useful for passing information to the profile to modify its settings, or to control its processing when the profile actually does some significant "batch-mode" editing with no human intervention.

The -nologo parameter turns off the initial program logo. This saves a few milliseconds at startup, but is mainly useful for getting rid of some annoying screen flashing when running multiple executions of CP-Edit in batch mode.

The font parameter consists of -f followed by the name of the font to be used. The -f parameter allows specifying the size text to be used in the editing session. Currently, only CO40 (40-column mode) and CO80 (80-column mode) are allowed with the DOS character-mode version of CP-Edit.

The complete syntax for invocation of CP-Edit is:

```
cpedit [ [d:][path]filename[.ext]
         [ -p [d:][path]profile[.ext]]
         [-f [ CO80 | CO40 ] ]
         [-nologo ]
         [-u "arbitrary text" ]
```

Examples:

```
CPEDIT CPEDIT.TXT

CPEDIT a:autoexec.bat

CPEDIT C:\CPEDIT3\XTSCREEN.PAS  -p C:\PASDIR\PROFILE.PAS

cpedit x.y -f CO40 -nologo

CPEDIT testfile -p batprof -u "My arbitrary processing parameters"
```

If you don't put the filename of the file you wish to edit on the command line, CP-Edit will ask you to enter the name. If you want to create a file, simply edit it using CP-Edit as if it existed. CP-Edit will inform you that you are editing a new file. When you issue the SAVE or FILE command for the file, the file will be created on disk for you.

During CP-Edit initialization, the program attempts to read the editing profile. This file may be a simple list of commands to be automatically executed before you start editing, in the case of a non-REXX profile; this allows simple tailoring of the editing session. Or it may contain some rather complex logic, if it is written in REXX, including setting different editing conditions depending on the type of file being edited, or it may actually do "batch-mode" automatic editing of the file, including a FILE command to save the file and terminate CP-Edit.

The profile is also executed each time a new file is edited from the CP-Edit environment using the Edit command.

The default name of the profile is

```
PROFILE.MAC
```

If you do not specify the drive and/or path for the profile, CP-Edit looks for the profile file on the default disk drive in the current directory. If it does not find it there, it uses the PATH to search for the profile file. (See the DOS or OS/2 command reference manual for information on using the PATH command to set the PATH.)

The profile may be in REXX or may be a CP-Edit "native" macro (see "Macros" below).

Two sample profiles are provided as part of the PACKAGE: SAMPROF.MAC is a sample REXX profile, and SAMPROFN.MAC is a sample "native" macro profile. You may rename either of these to PROFILE.MAC and use it as is, or edit it to customize your editing session.

The profile is optional. If none is found, CP-Edit warns you, but continues normally.

Editing files using CP-Edit

CP-Edit is a full-screen editor. This means that you can create and change text files by simply displaying the section of the file on the screen, using the cursor movement keys (the keys with arrows on them on the PC numeric keypad) to move the cursor to the point where you want to change something, and overtyping the text. Or you may use the Insert key to put the program into Insert mode to insert text into a line. Or you may use the Delete key to delete characters from a line.

Other keys, such as Home and End help you move the cursor around the screen, and you may use the PgUp and PgDn keys to page through the text in a file.

You can display multiple files on the screen with CP-Edit, or you can display multiple views of the same file. This makes it easy to compare files or move text back and forth among files or from widely separated places in the same file.

In addition to screen editing, CP-Edit lets you manipulate your file in two other ways as well. There are prefix commands, which make it easy to insert, delete, move, and copy lines of the file; and there are CP-Edit primary commands, which perform many powerful editing functions, such as locating a particular piece of text in your file, doing global changes, saving your file, editing multiple files, and entering text into the file.

CP-Edit includes an interface to REXX88. Using macros written in REXX, you can build powerful customized editing functions. See Section “CP-Edit Macros” on page 40.

CP-Edit also lets you assign commands to the function keys on the PC keyboard. Any primary command may be assigned to a function key. This allows you to enter a frequently used command with a single keystroke. A few function keys are pre-defined for you. See Appendix B, “Default Settings” on page 219 for the default settings.

Editing Using Full-Screen Features of CP-Edit

You will probably find the full-screen editing features of CP-Edit easiest to use for simple editing tasks. Your file is displayed on the monitor, and you use the editing keys on the PC keyboard to manipulate the text on the screen.

First a word or two about the cursor: The cursor is a marker on the screen to show you where characters will be changed, entered, or deleted. CP-Edit uses a different cursor from that used by DOS or OS/2. CP-Edit's cursor is a character-sized box of inverse color.

Next let's look at the layout of the screen used by CP-Edit. Figure 1 on page 10 shows a picture of the screen format. The top line of the screen contains the filename of the file currently being edited, some information on the size of the file and the line number of the current line. This line of the screen is called the ID line.

The next line on the screen contains an arrow (===>) and is called the Command line. This is where you enter CP-Edit primary commands (more on CP-Edit commands below).

The next 21 or 22 lines are called the File area of the screen. (If you are using an EGA or VGA display with a small font, you may have as many as 47 or 48 lines in the File area.) This is where the text of your file is displayed. This area really acts as a window into your file. You may move this window up and down using the PgUp and PgDn keys, and you may move the window left and right using the Left and Right commands (normally assigned to F9 and F10 for convenience).

The first 5 columns of the file area are occupied by the Prefix area. The Prefix is the 5-digit line number of the line in the file. The first line of the file is numbered 00001, the second one 00002, etc. The prefix area, in addition to displaying the line number, is used to enter Prefix commands. More on those below.

There is a pseudo-line numbered 00000 above the first real line of the file, which when displayed says

```
"* * * Top of File * * *"
```

There is a similar pseudo-line at the bottom of the file. These lines have their uses, as we will see later.

```
  SAMPLE.PAS              TRUNC=255 SIZE=11 LINE=0
===>









00000 * * * TOP OF FILE * * *
00001 PROCEDURE SAMPLE (INPUT,OUTPUT);
00002   (* This is a sample Pascal program *)
00003
00004 VAR
00005   greeting : LSTRING(255);
00006 BEGIN
00007   greeting := 'Hello there!';
00008   Writeln (greeting);
00009
00010 END (* of the SAMPLE program *) .
00011 * * * END OF FILE * * *

F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 1. CP-Edit Screen Format

The current line appears in the File area of the screen. By default, the current line appears one line above the middle of the screen (about line 12 of the screen for a 25-line display), but you can change this to suit yourself (using the SET CURLINE command).

This line is high-lighted on a monochrome monitor and is in bright white on a color monitor. (The color of this and other areas of the screen can be changed using the SET COLOR command). The current line is not really used in full-screen editing, but it becomes very important in using CP-Edit primary commands. More on these below.

There are two other special displays you can put on the screen: the Scale line and the Tab line. The Scale line shows you the column numbers for text in the File area and is similar to the scale on some typewriters. The Tab line shows you where your tab stops are set. Figure 2 on page 11 shows a sample screen with the Scale and Tab lines turned on. No two of the Current line, the Scale line, and the Tab line may appear on the same line of the screen.

The bottom two lines of the screen normally display the Function key settings for 10 function keys (the default is the first 10). This display may be turned off using the SET DSPF command.

Example:

```
SET DSPF OFF
```

When the Function key display is turned on, 21 lines of text are displayed on the screen.

When the Function key display is turned off, 22 lines of text are displayed, and the bottom line of the screen becomes a status display line. It shows the CP-Edit version number (in normal full-screen mode) or the word "Input" (in Input mode) or "Auto-indent" (in Auto-indent mode) and the number of files currently being edited (in storage).

From time to time, error or information messages may appear on the screen. They overlay the bottom-most lines of the screen. Usually, there will only be one message, so only the Function key display or Status area of the screen will be overlayed. Occasionally, there will be more than one message, and lines of your file will be overlayed on the screen. Rest assured, your text has not been damaged, and you can remove the messages from the screen after reading them by pressing Return or any function key.

Some Query commands, such as the Query PF command, clear the screen, display the output, then display the message "Press Space bar to continue". The display stays on the screen until you press the Space bar.

Error messages are high-lighted on a monochrome display and appear in red on a color display. They are accompanied by a "beep" (bell) to alert you to the error condition.

```
  SAMPLE.PAS              TRUNC=255 SIZE=11 LINE=0
===>




      T--------T---------T---------T---------T---------T---------T-
      <...+....1....+....2....+....3....+....4....+....5....+....6.
00000 * * * TOP OF FILE * * *
00001 PROCEDURE SAMPLE (INPUT,OUTPUT);
00002   (* This is a sample Pascal program *)
00003
00004 VAR
00005   greeting : LSTRING(255);
00006 BEGIN
00007   greeting := 'Hello there!';
00008   Writeln (greeting);
00009
00010 END (* of the SAMPLE program *) .
00011 * * * END OF FILE * * *

F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 2. Screen with Scale and Tab Lines

Now let's look at the keys and their functions in CP-Edit.

The default settings of the keys are given here. Most of these functions can be assigned to other keys on the keyboard using the KEYDEF program. See the KEYDEF.DOC file for details on the use of KEYDEF. Figure 3 on page 14 contains the KEYDEF key names for the various editing and cursor movement functions.

The **Return** key is used to signal to CP-Edit that you have entered a command and want to have it processed. It also results in the screen being updated and re-displayed.

The **Home** key always returns the cursor to the first column in the command area of the command line.

The **Ctrl-e** key combination moves the cursor to the last line on the file area of the screen. It does not change the current column.

Pressing the **Ctrl-Home** key combination moves the cursor to the beginning of the command line in the current logical screen.

On the Enhanced (101-key) Keyboard, the **keypad-5** key moves the cursor to the current line on the screen. In INPUT mode, the keypad-5 moves the cursor to the top line of the input area of the screen (the line following the current line).

The **PgUp** and **PgDn** keys are used to page up and down through your file. You can set them to page up and down by a full page or half page using the **SET PGUP** and **SET PGDN** commands.

The arrow keys (**Up, Down, Left,** and **Right** arrows) at the right side of the keyboard move the cursor as you would expect.

The **Insert** key puts CP-Edit in Insert Mode. The word INSERT appears on the ID line of the screen and characters typed into a line in the file area of the screen are inserted in the line preceding the cursor position. Pressing Return or any function key (including PgUp and PgDn) turns off Insert Mode. Also pressing the Insert key while in Insert Mode turns off Insert mode. (That is, the Insert key acts as a toggle for Insert Mode.) Insert Mode only works in the parts of the screen where your text is displayed and in the command line. If you type into the Prefix area while INSERT mode, it is as if INSERT mode were not on. If you attempt to insert characters elsewhere (such as in the ID line, the Scale, etc.), CP-Edit will emit a beep and won't allow the insertion.

The **Delete** key deletes a single character under the cursor. Like Insert mode, the Delete key only works in the command line and in your text. If you try to delete characters elsewhere on the screen, CP-Edit will beep at you.

The **backward arrow** key has the same function in CP-Edit as the Left arrow on the numeric keypad (a non-destructive backspace -- this is different from its use in DOS).

The **Forward Tab** key (unshifted Tab key) behaves much like the tab key on a typewriter. It moves the cursor to the right to the next tab stop you have set. If the last tab stop on the screen has been passed, it moves to the next line of the screen. If the cursor is in the prefix area, pressing the tab key moves the cursor to the first column of the text on the line.

The **Back Tab** (shifted Tab key) moves the cursor to the next tab stop to the left.

The **Ctrl-Right-Arrow** and **Ctrl-Left-Arrow** key combinations perform a word-tab and backward-word-tab function respectively. Words are defined to be delimited by blanks on the screen. Pressing the word-tab key results in moving the cursor to the begining of the next word on the screen. Pressing the backward-word-tab moves the cursor to the beginning of the previous word on the screen.

The **Plus(+)** bar and **Minus(-)** key to the right of the numeric keypad on the PC keyboard have special uses for full-screen editing with CP-Edit. The Plus bar is used as a Newline key. Pressing this key causes the cursor to move

to the beginning of the next line on the screen. In addition, the **Ctrl-Return key** combination performs a Newline function as well.

The **Minus** key on the numeric keypad is used as a Clear-To-End-Of-Line key. Pressing this key results in the rest of the line from the cursor position to the right edge of the screen being cleared to blanks.

The **Alt-L** key combination moves the cursor to the leftmost column on the screen.

The **End** key moves the cursor to the first blank after the rightmost non-blank character on the line.

There are several other keys that are most useful when the screen is split into multiple logical screens (see the SET SCREEN command):

The **Ctrl-Home** key combination moves the cursor to the beginning of the command line of the logical screen which currently contains the cursor.

The **Ctrl-PgDn** key combination moves the cursor to the beginning of the command line of the next logical screen on the physical screen ("next" is the next screen to the right or down, normally).

The **Ctrl-PgUp** key combination moves the cursor to the beginning of the command line of the previous logical screen on the physical screen ("previous" is the next screen to the left or up, normally).

The **Function** keys (or Program Function -- PF -- keys) may be set to frequently used CP-Edit commands. More on them below.

The **Esc** (Escape) key is used for exiting from HELP files and from other files (if they have not been changed). Actually, it behaves like a Function key to which the QUIT command has been permanently assigned. Pressing the Esc key is equivalent to typing in a QUIT command.

Name	Aliases	Function performed
f1 - f40	(pf1 - pf40)	Program Function keys.
esc	(escape)	QUIT command
ins	(insert)	Toggle Insert mode
del	(delete)	Delete a character
home		Cursor to 1st screen command line
end	(rightmost)	Cursor to end of line
pgup	(pageup)	Page up
pgdn	(pagedown)	Page down
up	(uparrow, cursorup)	cursor up
down	(downarrow, cursordown)	cursor down
left	(leftarrow, cursorleft)	cursor left
right	(rightarrow, cursorright)	cursor right
backspace		same as cursor left
newline		Cursor to beginning of next line.
enter	(Return)	Execute command
eraseeof	(eraseeol)	Erase from cursor to end of line
center	(pad5, curline)	Cursor to current line
bottom		Cursor to bottom of screen
leftmost		Cursor to left edge of screen
wordtab	(wordtabf, ctrlright)	forward word tab
wordtabb	(ctrlleft)	backward word tab
xtabcmd		tab to current screen command line
xtabcmdf		tab to next screen command line
xtabcmdb		tab to previous screen cmd line

Figure 3. KEYDEF Key Names

In addition to all these editing functions and cursor movement functions, CP-Edit also contains several other editing functions designed to mimic XEDIT operating on a 3270 display. These functions are not assigned to any key by default, but may be assigned to keys using KEYDEF. See Figure 4 for KEYDEF names for these keys and their function. Also see the KEYDEF.DOC file for details on the use of KEYDEF.

Name	Aliases	Function performed
desbksp		Destructive Backspace
xtab		XEDIT-style tab
xbacktab		XEDIT-style backtab
clear		XEDIT Clear key
dead	(nothing)	(Ignored)

Figure 4. KEYDEF XEDIT Key Names

A few keys are not controlled by CP-Edit, but you can take advantage of them while using CP-Edit:

The **Caps Lock** key has its normal PC function. That is, it toggles between Caps-lock mode and normal mode. This provides an easy way to enter text which is all in upper case. The word "CAPS" is displayed on the top line of the screen in Caps-lock mode.

The **Shift-PrtScreen** key performs its normal PC function. You can use it to capture a CP-Edit screen on the printer while using CP-Edit. It is an easy way to get a printout of a small part of a file, if you aren't too fussy about the format of the printout.

The **Num Lock** key has its normal PC function. Normally you will want to leave Num Lock off, since you cannot use the keys on the keypad for cursor movement or paging if Num Lock is on. If you ever get numbers on the screen when you were trying to move the cursor, you may have accidentally pressed the Num Lock key. If so, just press it again to turn Num Lock off.

Some keys are disabled by CP-Edit. In particular, the **Ctrl-Break** key combination is disabled so you cannot accidentally terminate CP-Edit.

Keys which normally autorepeat retain this function in CP-Edit. That is, if you hold them down for a while, you will get multiple characters entered. This can be useful for tabbing quickly to the right on the screen, or deleting a lot of text from the middle of the line, for example.

Editing Using Prefix Commands

The CP-Edit Prefix commands give you an easy way to manipulate whole lines of text. They are easy to remember (they have mnemonic names) and require only a few keystrokes to perform powerful editing functions. The Prefix commands give you the ability to move **(M, MM)** or copy **(C, CC)** single lines or blocks of lines, insert **(I)** lines, delete **(D, DD)** single lines or blocks of lines, duplicate **(", "")** single lines or blocks of lines, shift lines left **(<, <<)** or right **(>, >>)** , and make a line on the screen the current line **(/)**, or set the Scale **(SCALE)** or Tab line **(TABL)** on the screen. A powerful Text Flow **(TF)** prefix command formats text into even lines for you.

Several of the Prefix commands accept a number along with the command. This number may be entered in the prefix area either before or after the prefix command. The number has a different meaning for different commands. For the block Shift commands **(<<, >>)** and the block Duplicate (Repeat) commands **("", RR)**, the number must accompany the first command (the one nearest the beginning of the file).

You may enter a prefix command anywhere in the prefix area for a line, overtyping the line number. Only one prefix command may be entered per line. You may enter spaces freely in the prefix area. They are ignored by CP-Edit. Some of the examples shown in this section show spaces around prefix commands for clarity.

Several of the Prefix commands can operate on either single lines or blocks of lines. The single line prefix command consists of a single character, and the block version of the same command is two of the same character. For example, the single line delete prefix command is "**D**", and the block delete prefix command is "**DD**". To mark a block for manipulation, the two-character block prefix command is entered in the prefix of the first and last line of the block. See Figure 5 on page 16 and Figure 6 on page 17 for an example of use of the block delete commands.

```
  SAMPLE2.PAS             TRUNC=255 SIZE=91 LINE=66
===>
00056 BEGIN  (* BODY OF SORTLIST *)
00057
00058
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
dd 63     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00064
dd 65     (* DISTRIBUTE RECORDS TO POCKETS *)
00066
00067     WHILE
00068       P <> NIL
00069     DO BEGIN
00070       PNEXT := P@.NEXT;
00071       ADDPOCKET (P,ORD(P@.KEY1(. J .)),POCKET,ENDPOCKET);
00072       P := PNEXT
00073     END;  (* OF DISTRIBUTE LOOP *)
00074
00075     COLLECT;   (* COLLECT POCKETS BACK ON CHAIN *)
00076
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 5. Block Delete Commands (Before)

```
  SAMPLE2.PAS             TRUNC=255 SIZE=89 LINE=66
===>
00056 BEGIN  (* BODY OF SORTLIST *)
00057
00058
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
00063
00064     WHILE
00065       P <> NIL
00066     DO BEGIN
00067       PNEXT := P@.NEXT;
00068       ADDPOCKET (P,ORD(P@.KEY1(. J .)),POCKET,ENDPOCKET);
00069       P := PNEXT
00070     END;  (* OF DISTRIBUTE LOOP *)
00071
00072     COLLECT;   (* COLLECT POCKETS BACK ON CHAIN *)
00073
00074   END;   (* FOR LOOP THRU CHARS OF KEY1 *)
00075
00076
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 6. Block Delete Commands (After)

(Note that although the example prefix commands in this section may be shown in upper case for clarity, alphabetic prefix commands may be entered in either upper or lower case, or any combination of upper and lower. For example, a block delete command might be entered as "dd", "DD", "Dd", or "dD". All are equivalent.)

Multiple prefix commands may be entered on the screen at the same time. They are not actually executed by CP-Edit until you press Return or a Function key. See Figure 7 on page 18 and Figure 8 on page 19 for an example of multiple prefix commands.

```
  SAMPLE2.PAS             TRUNC=255 SIZE=91 LINE=68
===>
00058
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
00063     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00064
00065     (* DISTRIBUTE RECORDS TO POCKETS *)
00066
dd 67     WHILE
00068       P <> NIL
dd 69     DO BEGIN
<<2 0       PNEXT := P@.NEXT;
00071       ADDPOCKET (P,ORD(P@.KEY1(. J .)),POCKET,ENDPOCKET);
<< 72       P := PNEXT
d 2       END;  (* OF DISTRIBUTE LOOP *)
00074
00075     COLLECT;   (* COLLECT POCKETS BACK ON CHAIN *)
00076
00077   END;   (* FOR LOOP THRU CHARS OF KEY1 *)
00078
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 7. Multiple Prefix Commands (Before)

```
  SAMPLE2.PAS             TRUNC=255 SIZE=86 LINE=68
===>
00058
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
00063     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00064
00065     (* DISTRIBUTE RECORDS TO POCKETS *)
00066
00067     PNEXT := P@.NEXT;
00068     ADDPOCKET (P,ORD(P@.KEY1(. J .)),POCKET,ENDPOCKET);
00069     P := PNEXT
00070     COLLECT;   (* COLLECT POCKETS BACK ON CHAIN *)
00071
00072   END;   (* FOR LOOP THRU CHARS OF KEY1 *)
00073
00074
00075   FOR
00076     J := LKEY2 DOWNTO 1
00077   DO BEGIN
00078     P := ROOT;
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 8. Multiple Prefix Commands (After)

Also, CP-Edit can remember one partially-entered set of block prefix commands across screens. For example, you might enter a DD prefix command on line 58 of the file, page the display forward to line 77 of the file (or move forward some other way, such as the Locate or Find commands), and enter another DD command in the prefix of line 77. If you then pressed Return, lines 58 through 77 of the file would be deleted. See Figure 9 on page 20, Figure 10 on page 21, and Figure 11 on page 22 for an example of this.

```
  SAMPLE2.PAS             TRUNC=255 SIZE=91 LINE=60
===>
00050       END  (* IF *)
00051     END  (* COLLECT LOOP *)
00052
00053   END;  (* OF COLLECT *)
00054
00055
00056 BEGIN  (* BODY OF SORTLIST *)
00057
dd 58
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
00063     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00064
00065     (* DISTRIBUTE RECORDS TO POCKETS *)
00066
00067     WHILE
00068       P <> NIL
00069     DO BEGIN
00070       PNEXT := P@.NEXT;
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 9. First Block Delete Command Entered

```
  SAMPLE2.PAS             TRUNC=255 SIZE=91 LINE=68
===>
DD 58
00059   FOR
00060     J := LKEY1 DOWNTO 1
00061   DO BEGIN
00062     P := ROOT;
00063     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00064
00065     (* DISTRIBUTE RECORDS TO POCKETS *)
00066
00067     WHILE
00068       P <> NIL
00069     DO BEGIN
00070       PNEXT := P@.NEXT;
00071       ADDPOCKET (P,ORD(P@.KEY1(. J .)),POCKET,ENDPOCKET);
00072       P := PNEXT
00073     END;  (* OF DISTRIBUTE LOOP *)
00074
00075     COLLECT;   (* COLLECT POCKETS BACK ON CHAIN *)
00076
 dd 7   END;   (* FOR LOOP THRU CHARS OF KEY1 *)
00078
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 10. Second Block Delete Command Entered. (First Block Delete Command is Pending)

```
  SAMPLE2.PAS             TRUNC=255 SIZE=71 LINE=57
===>
00047         POCKET(. I .) <> NIL
00048       THEN BEGIN
00049         ADDCHAIN (ROOT,LEND,POCKET (. I .),ENDPOCKET (. I .))
00050       END  (* IF *)
00051     END  (* COLLECT LOOP *)
00052
00053   END;  (* OF COLLECT *)
00054
00055
00056 BEGIN  (* BODY OF SORTLIST *)
00057
00058
00059
00060   FOR
00061     J := LKEY2 DOWNTO 1
00062   DO BEGIN
00063     P := ROOT;
00064     INITPOCKETS;  (* CLEAR OUT POCKETS *)
00065
00066     (* DISTRIBUTE RECORDS TO POCKETS *)
00067
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 11. Block Delete Commands Executed

Prefix commands which have been partially entered are said to be "pending". As mentioned above, CP-Edit can remember one set of pending prefix commands. If any commands of a set of pending prefix commands happens to appear on the screen, they are high-lighted. You may overtype a pending prefix command on the screen to change it. You may overtype it with spaces to delete it. Entering the RESet command clears any and all pending prefix commands.

The **M** (Move) and **C** (Copy) prefix commands require the use of an additional prefix command to mark where the lines are to be moved or copied. If you enter a move or copy command and mark a line with **P** for Preceding or **B** for Before, the line(s) will be moved or copied before the line marked with **P** or **B**. (A block move or copy may also consist of a single M or C and a number of lines to move or copy.) If you mark the position for the move or copy with **F** for Following or **A** for After, the line(s) will be moved or copied following the line marked with **F** or **A**.

Note that for a block move or copy a total of three prefix commands constitute a complete set: a **CC** or **MM** to mark the beginning of the block, a matching **CC** or **MM** to mark the end of the block, and a **P (B)** or **F (A)** to mark the place to which the block will be moved or copied. The set of commands will not be executed until all three commands of the set are entered, and you have pressed Return or a Function key. See Figure 12 on page 23 and Figure 13 on page 24 for examples of move and copy prefix commands.

```
  SAMPLE2.TXT             TRUNC=255 SIZE=10 LINE=4
===>




00000 * * * TOP OF FILE * * *
mm 01 This is line 1 of the sample text.
mm 02   This is line 2.
f 003     This is line 3.
00004       this is line 4
00005         This is line 5 of the sample text file
00006         THIS IS LINE 6.
c2 07       THIS IS LINE 7.
00008     This is line 8.
00009   This is line 9.
p 010 This is line 10 of the sample text.
00011 * * * END OF FILE * * *



F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 12. Move and Copy Prefix Commands (Before)

```
  SAMPLE2.TXT             TRUNC=255 SIZE=12 LINE=4
===>




00000 * * * TOP OF FILE * * *
00001     This is line 3.
00002 This is line 1 of the sample text.
00003   This is line 2.
00004       this is line 4
00005         This is line 5 of the sample text file
00006         THIS IS LINE 6.
00007       THIS IS LINE 7.
00008     This is line 8.
00009   This is line 9.
00010       THIS IS LINE 7.
00011     This is line 8.
00012 This is line 10 of the sample text.
00013 * * * END OF FILE * * *

F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 13. Move and Copy Prefix Commands (After)

The " (Duplicate) command can make one or more copies of a line or group of lines. See Figure 14 on page 25 and Figure 15 on page 26 for an example of the Duplicate command.

```
  SAMPLE.TXT              TRUNC=255 SIZE=4 LINE=5
===>









00000 * * * TOP OF FILE * * *
"" 01 Some sample text.
"" 02
00003 Some more sample text.
 "7 4 ********* HOORAY ************
00005 * * * END OF FILE * * *









F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 14. Duplicate Prefix Command (Before)

```
  SAMPLE.TXT              TRUNC=255 SIZE=13 LINE=2
===>




00000 * * * TOP OF FILE * * *
00001 Some sample text.
00002
00003 Some sample text.
00004
00005 Some more sample text.
00006 ********* HOORAY ************
00007 ********* HOORAY ************
00008 ********* HOORAY ************
00009 ********* HOORAY ************
00010 ********* HOORAY ************
00011 ********* HOORAY ************
00012 ********* HOORAY ************
00013 ********* HOORAY ************
00014 * * * END OF FILE * * *
F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 15. Duplicate Prefix Command (After)

The Shift **(<, <<, >, >>)** prefix commands are useful for changing indentation for structured programming or simple text processing. See Figure 16 on page 27 and Figure 17 on page 28 for an example of use of the shift commands.

```
  SAMPLE2.TXT             TRUNC=255 SIZE=10 LINE=4
===>




00000 * * * TOP OF FILE * * *
00001 This is line 1 of the sample text.
2> 02   This is line 2.
00003     This is line 3.
2< 04       this is line 4
<<4 5         This is line 5 of the sample text file
<< 06         THIS IS LINE 6.
00007       THIS IS LINE 7.
00008     This is line 8.
00009   This is line 9.
00010 This is line 10 of the sample text.
00011 * * * END OF FILE * * *



F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 16. Shift Prefix Commands (Before)

```
  SAMPLE2.TXT             TRUNC=255 SIZE=10 LINE=4
===>




00000 * * * TOP OF FILE * * *
00001 This is line 1 of the sample text.
00002     This is line 2.
00003     This is line 3.
00004     this is line 4
00005     This is line 5 of the sample text file
00006     THIS IS LINE 6.
00007       THIS IS LINE 7.
00008     This is line 8.
00009   This is line 9.
00010 This is line 10 of the sample text.
00011 * * * END OF FILE * * *



F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 17. Shift Prefix Commands (After)

The **I** (Insert) prefix command inserts one or more lines in the file. If only the **I** is entered in the prefix area of a line, a single blank line is added following that line. If **I** and a number is entered, that number of blank lines is added to the file. Figure 18 on page 29 shows the screen after entering **I** prefix commands and before pressing Return; Figure 19 on page 30 shows the screen after pressing Return.

```
  SAMPLE2.TXT             TRUNC=255 SIZE=10 LINE=4
===>




00000 * * * TOP OF FILE * * *
00001 This is line 1 of the sample text.
I 002   This is line 2.
00003     This is line 3.
00004       this is line 4
00005         This is line 5 of the sample text file
 2i 6         THIS IS LINE 6.
00007       THIS IS LINE 7.
00008     This is line 8.
00009   This is line 9.
00010 This is line 10 of the sample text.
00011 * * * END OF FILE * * *



F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 18. Insert Prefix Command (Before)

```
  SAMPLE2.TXT             TRUNC=255 SIZE=13 LINE=5
===>




00000 * * * TOP OF FILE * * *
00001 This is line 1 of the sample text.
00002   This is line 2.
00003
00004     This is line 3.
00005       this is line 4
00006         This is line 5 of the sample text file
00007         THIS IS LINE 6.
00008
00009
00010       THIS IS LINE 7.
00011     This is line 8.
00012   This is line 9.
00013 This is line 10 of the sample text.
00014 * * * END OF FILE * * *

F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 19. Insert Prefix Command (After)

Placing a slash **(/)** in the prefix area of a line and pressing Return makes it the current line. This is a useful alternative to the Locate command and other commands which change the current line of the file.

The Scale and Tabl commands are prefix alternatives to the **SET SCALE ON** and **SET TABL ON** commands.

In CP-Edit, block prefix commands may not overlap. You may enter only one prefix command on any line. And single line prefix commands may not be placed inside a pair of block prefix commands.

The pseudo-lines at the top and bottom of the file may be used to insert, move, or copy data before the first line of the file or after the last line of the file. If you put a delete or shift prefix command on the top or bottom pseudo-line, it is ignored. Similarly, trying to move or copy the top or bottom pseudo-line is ignored.

The section “CP-Edit Prefix Commands” on page 193 of this document gives you complete details on the use of each prefix command.

The prefix itself may be turned on or off or set on the right or left side of the screen using the **SET PREFIX** command. The prefix contain line numbers or just '=====' for each line depending on the setting of the **SET NUMBER** command.

Editing Using CP-Edit Commands

CP-Edit's English-like commands provide you with powerful editing capabilities. They give you the means to locate fields in your file, move rapidly through your file, make global changes, edit multiple files, move text between files, set Function keys, tab stops, and various other parameters, and enter text efficiently into your file.

CP-Edit commands are entered on the command line. The command line is the second line of the screen. Commands are entered following the arrow (===>). You can get the cursor to the command line easily by pressing the Home key. After typing in the command you must press Return or a Function key in order for CP-Edit to realize that you have entered a command. Before you press Return, you can use the screen editing keys (such as the left and right arrows, Insert and Delete) to correct the command you have typed

Case is not significant when entering CP-Edit commands except when using strings which must match text in your file. In other words, you may enter the names of the commands in upper or lower case or any combination of upper and lower. Examples:

```
Edit
edit
EDIT
edIT
```

All the above are equivalent.

Many CP-Edit commands may be abbreviated. In this document, we show the minimum abbreviation in capital letters (upper case) and the rest of the command in small letters (lower case). (As mentioned above, the commands may be entered in either upper or lower or any combination.) In order for CP-Edit to recognize a command, you must enter at least the minimum abbreviation. You may enter as many more characters of the command as you wish up to the full name of the command.

When CP-Edit successfully executes a command, it clears the command line. If there is an error in the command, CP-Edit gives you an error message and leaves the command on the command line. You may then edit the command to correct it, or overtype the command with another one, or just erase the command from the screen (with the Clear-to-end-of-line key, for example). You may retrieve any of the last eight commands entered using the "?" command. The retrieved command may then just be re-executed or edited and then re-executed.

Let's look at some groups of commands.

First, the **Help** command allows you to get help in using CP-Edit commands.

Next there are the commands which get you into and out of files and save them for you. The **FILE** command saves the current file on disk and deletes the file from storage. If it is the only file being edited, CP-Edit exits to DOS (terminates). The **SAVE** command just saves the current file on disk. The Edit (or TEDit) command brings a new file into storage and makes it the current file, or switches between files already in storage. The **QUIT** command just deletes the file from storage without saving it to disk. Pressing the **Esc** key is the same as entering a QUIT command.

If you have made changes to the file, CP-Edit will not allow you to QUIT, but a "quit-anyway" command named **QQuit** is provided if you wish to quit without saving any changes made. The **CANCEL** command does a QUIT from all files and exits to DOS (if CP-Edit cannot QUIT from any file, it will not terminate).

You may edit as many files as will fit in storage. You can find out how many files are currently in storage using the Query **NBFile** command. You can also use the **Query RING** command to display the number and names of all

the files currently in storage. If the PF key display is not on the bottom of the screen (you have issued the **SET DSPF OFF** command), the number of files in storage is displayed at the bottom of the screen.

The **GET, PUT** and **PUTD** commands are used to move lines between files in storage and on disk. The **GET** command gets lines from a file on disk and inserts them in the current file in storage. The **PUT** command write lines from the current file in storage out to disk. The **PUTD** command is similar to the PUT command except it also deletes the lines from the current file.

The Locate command is used to position the current line to an arbitrary line of the file or to a line containing an arbitrary string of characters (or NOT containing the string, forward or backward in the file -- see the discussion of Targets in the section “CP-Edit Targets” on page 33 for details). The Find command does a column-dependent search for an arbitrary string of characters.

The Next command moves forward one or more lines in the file (positions the current line). The Up command moves backward one or more lines. The **TOP** command postions the current line to the top line of the file (the pseudo-line before the first line of your text). The **BOttom** command positions the current line to the bottom line of the file (the pseudo-line after the last line of your text). These commands result in moving the display window up or down in the file. In fact, the **PgUp** key is set to an Up 10 command and the **PgDn** key is set to Next 10 command.

The **Right** and **Left** commands are used to move the display window right or left respectively over your text. Function key 9 is set to Left 10, and Function key 10 is set to Right 10 initially.

The Change command allows you to make global changes in your file. You may change any arbitrary string to any other arbitrary string. Either the old string or the new string may be null. The range of lines over which the changes are made starts with the current line; the end of the range may be specified in any of several ways. The Change command is one of the most powerful commands in CP-Edit.

The **SPLIT CURSOR** command can be used to split a line of text into two lines. Usually this is easiest if the SPLIT CURSOR command is assigned to a Program Function key. You position the cursor to the place where the line is to be split, then press the assigned Program Function key, and CP-Edit splits the line into two lines. See the SPLIT CURSOR command in the section “CP-Edit Commands” on page 45.

Similarly, the **JOIN** and **JOIN CURSOR** commands may be used to splice together lines of text. See the JOIN command in the section “CP-Edit Commands” on page 45.

The **Input** command is used to insert a single line of text into your file, or to enter Input Mode for entering multiple lines of text.

The **POW**erinp command puts you into Power Typing mode. It gives you an easy full screen way of entering lots of text without worrying about line boundaries. We will talk about Power Typing mode more later in the section “Entering Text” on page 35.

The **AUTOI**n (AI) command is similar to the Input command except that it automatically positions the cursor so that text is consistently indented. Auto-indent mode is useful for entering indented text or for structured programming, where indentation is used to show program structure.

There are several **SET** commands which set various editing parameters.

The **SET CASE** command allows you enter text in all caps or to disregard case in target string searches.

The **SET COLOR** allows you to change the colors used for the various areas of the screen if you are using a color display, or change the highlighting of the various areas of the screen if you are using a monochrome adapter and display.

The **SET CURL**ine command sets the position of the Current Line on the screen.

The **SET FN**ame command allows you to change the name of the current file.

The **SET LINEND** command allows you to change or turn off the Linend character used in Power Typing.

The **SET PF**nn command allows you to set Function keys (Program Function keys).

The **SET DSPF** command allows you to turn on and off the Function key display at the cottom of the screen and to set the range of Function keys displayed.

The **SET SCAL**e command allows you to turn on or off and set the position of the Scale line on the screen. The **SET TABL**ine command allows you to turn on or off and set the position of the Tab line display on the screen.

The **SET TABS** command allows you to set the tab stops.

The **SET ZONE** and **SET TRUNC** commands allow you to set limits on the range of columns you are using at any particular time. The **SET TRUNC** command can keep text to the right of the truncation column from being affected by Change commands or screen editing. The **SET ZONE** command allows you to determine the range of columns in your file that are affected by the Change and Locate commands.

The **SET V**erify command sets the starting and ending columns of the file which are displayed on the screen.

The **SET SCR**een command allows you to split the physical screen into multiple logical screens and to display multiple files or multiple views of the same file on the screen at the same time.

There are Query commands to query some of the above parameters.

The **Query PF** command queries the setting of one or all of the Program Function keys. The **Query TABS** command queries the setting of the tab stops.

The **RESET** command removes any pending Prefix commands.

The **=** command just re-executes the last command entered. It is very useful for repetitively executing the same command over and over.

The **?** command retrieves any of the last eight commands entered and places it on the command line for editing or re-execution.

Finally, a CP-Edit comment is any command line which begins with an *****. This is mostly useful to document the commands in CP-Edit "native" macros.

CP-Edit Targets

Targets are used on several CP-Edit commands. A target specifies the lines over which the command operates. (The range of lines always begins with the current line.) Targets in CP-Edit have one of the following forms:

1. an absolute line number, specified as a decimal number preceded by a colon:

   Example:

   ```
   :27
   ```

If an absolute line number is used as the target, the command acts on lines beginning with the current line up to, but not including, the absolute line number in the target.

2. A relative displacement from the current line which may be preceded by a plus sign or minus sign:

   Examples:

   ```
   +18

   -10

   14
   ```

   Most CP-Edit commands move forward only and will not accept negative relative line numbers. Most will accept positive relative line numbers. If a positive relative line number is specified for the target, the command acts from the current line for the number of lines specified.

3. A simple string expression consisting of a string within delimiters optionally preceded by a plus ("+") or minus ("-") sign and/or a not-sign (the ASCII tilde, represented here by "~"):

   ```
   /string1/

   +/string1/

   -/string1/

   ~/string1/

   +~/string1/

   -~/string1/
   ```

   The plus sign is always optional. It specifies movement forward (toward the end) of the file. The minus sign specifies backward movement (toward the beginning) of the file.

   The "~" is used to mean "not". A string expression containing "~" means "up to, but not including, the first line NOT containing string1". If both a plus or minus and a not-sign are used, the plus or minus must precede the not-sign.

   Examples:

   ```
   /BEGIN/

   -/PROCEDURE/

   -~/(*/
   ```

4. A pointer target, which is a symbolic name for a line set using the SET POINT command or the .xxxx prefix command.

   ```
   .start
   .abc
   .endline
   ```

   In CP-Edit, case is significant in pointer symbolic names. Thus, the following are different labels:

   ```
   .start
   .START
   .Start
   ```

Searches for string targets are affected by the **SET CASE R**espect or Ignore setting. If Respect is in effect, string targets must match file text exactly. If Ignore is in effect, alphabetic characters will match corresponding letters whether they are upper or lower case. See the **SET CASE** command for further details.

See also the examples of targets in the descriptions of the Locate and Change commands.

Entering Text

There are several ways to enter new lines of text into a file with CP-Edit: You may insert a single line using the Input command. If you only want to insert a small number of lines of text, you may use the I prefix command, or the Add command. If you want to enter more than a few lines of text, you probably will want to use Input Mode. Or if you want to use auto-indenting, you can use Auto-indent mode (the AUTOIN or AI command), or the SI prefix command.

The I prefix command is entered anywhere in the prefix of the line after which you wish to insert line(s). If no number is typed in in the prefix along with the I, one blank line is inserted in the file. If you type in a number preceding or following the I, that number of lines is inserted following the line whose prefix you over-typed. The cursor is positioned at the beginning of the first blank line, and you may then enter text on the line.

The Input command may be used in two ways. If you include text on the command line following the Input command, that text is inserted into the file immediately following the current line. Obviously, you cannot insert a blank line this way!

If you do not include text on the Input command, CP-Edit switches to Input Mode. You will see the appearance of the screen change in Input Mode. (See Figure 20 on page 36 below for an example.) When you first enter Input Mode, "Input:" will be displayed at the bottom of the screen. The file area of the screen below the current line will clear and the cursor will be placed in the first data column of this clear area (the Input Area). You may then type in text anywhere in the input area. If you wish to include blank lines in the file, you must enter at least one space on the line in the input area (lines on which nothing at all has been entered are not inserted into the file). Note that pressing the tab key alone does not enter any text on a line.

When you press Return once in Input Mode, the lines you have entered are inserted into the file. The Input Area is cleared for you and the cursor is repositioned to the first data column of the first line of the Input Area. You may then continue entering text. (The INPUT: message will be cleared from the screen and the Status Area of the screen will indicate Input and the number of files currently being edited. See Figure 21 on page 37 below for an example.)

If you press Return twice in a row without entering any text in between, CP-Edit will exit from Input Mode. You will see the screen return to normal editing mode.

Auto-indent mode is very similar to Input mode, except that the cursor is automatically positioned to the column which contains the first non-blank character in the current line. You enter Auto-indent mode by using the AUTOIn command. It has no operands. The cursor may be moved to the left or right of its original position to enter text if the indentation is to change. In order for a change in the indentation column to take place, the Return key or a Function key must be pressed. Exiting from Auto-indent mode is the same as from Input mode.

You may use all the normal editing keys (such as Insert, Delete, Cleareol, etc.) to change text in the input area or in the area on the screen above the input area. While CP-Edit is in Input Mode or Auto-indent mode, you may use the PF keys, enter commands on the command line, and use prefix commands on the lines of the screen above the Input Area. Some combinations of these may cause CP-Edit to exit from Input Mode, however.

The AUTOIn or AI command was put into CP-Edit before the SI command (and SI prefix command) existed in XEDIT, but CP-Edit also has the SI command and prefix command. They are similar to the AI command, except that they operate via the prefix, and do not present more than one line of input area. They are designed to be very similar to the ISPF editor I prefix command.

The Add command just adds one or more blank lines to the file following the current line. The current line does not move.

Note that operation of input of all kinds is affected by the Case setting. If you have entered a SET CASE UPPER command, all the text entered will be converted to upper case.

Also, entering text is affected by the setting of the truncation column. If, for example, you have entered the command

SET TRUNC 40

text entered will be truncated after column 40.

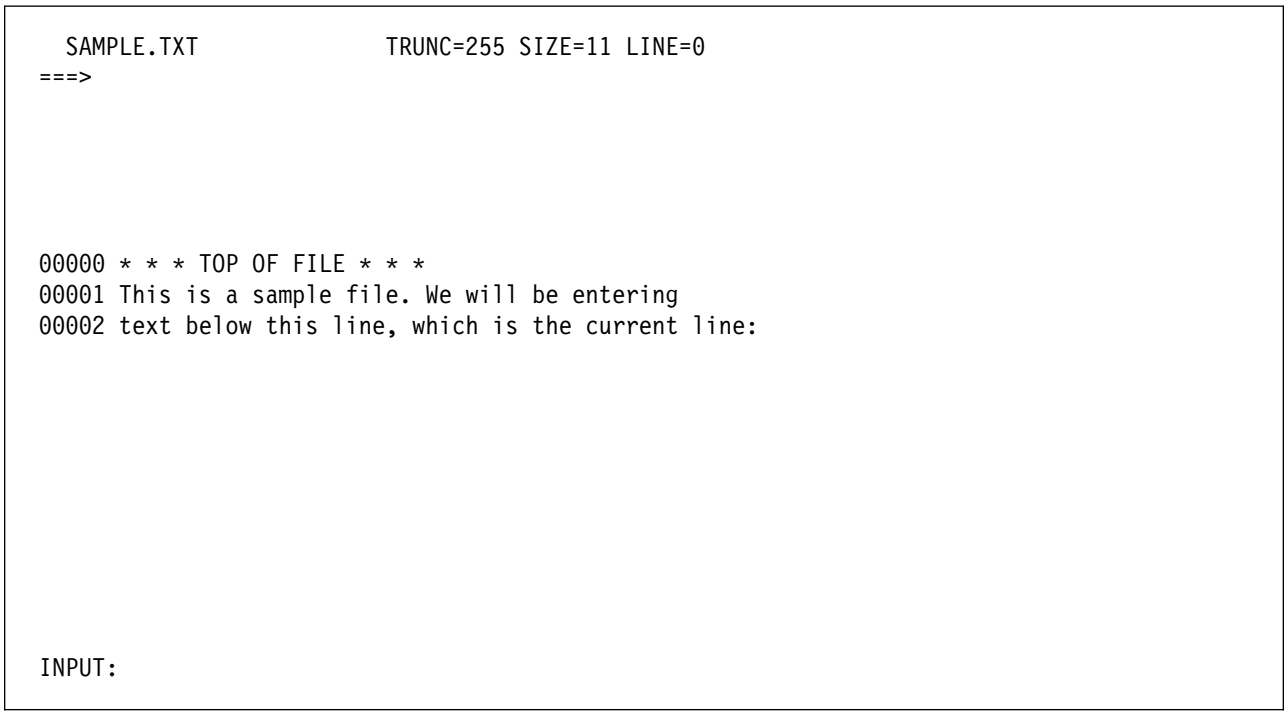


Figure 20. Switching to Input Mode

```
  SAMPLE.TXT              TRUNC=255 SIZE=11 LINE=0
===>



00000 * * * TOP OF FILE * * *
00001 This is a sample file. We will be entering
00002 text below this line, which is the current line:
00003 We have now entered some text and pressed Return
00004 once to cause the text to be inserted in the file.









                                                Input
```

Figure 21. Entering Text in Input Mode

For entering large amounts of text, you might wish to use Power Typing. This mode is similar to Input mode, but you don't have to worry about line boundaries. You just enter the text without respect to lines. The screen is treated as one long stream of text, with the cursor automatically moving to the next line when you reach the end of a line. Then when you press Return, the program automatically flows lines to word boundaries and inserts the text into the file following the current line.

Power Typing allows you to use the entire screen for entering text except for the first two lines. The first line of the screen shows the name of the file and the Power Typing title. The second line of the screen shows the current line of the file. See Figure 22 on page 38 for the screen format.

You may enter as much text as you like until the screen is full. When you press Return, the stream of text is split into even lines and inserted into the file following the current line. The last line of the inserted text becomes the current line and is displayed on the second line of the screen. When you have finished entering text, press Return twice to exit from Power Typing mode. See Figure 23 on page 38 to see a sample of the text on exit from Power Typing.

In Power Typing, insertion works differently from normal editing or Input mode. Since the entire screen is treated as a stream of text, insertions wrap across multiple lines. When you insert characters, the entire stream of text shifts to the right.

CP-Edit keeps track of where you have entered text on the screen, even blanks. Only those characters you have entered on the screen are inserted into the file. If you want blanks to be in your text, you must type them in.

If you want to cause a break in your text, you can do it in either of two ways: You can just press Return when you have entered the end of a section, or you can use the Linend character to cause a break in the text. Wherever you type in a Linend character in Power Typing mode, a new line is started when the text is inserted in the file. The default Linend is a **Pound Sign (#)**. If you need to use this character in your text, you can set the Linend to some other character, or set it off entirely using the **SET LINEND** command. You can find out the current Linend character using the **QUERY LINEND** command.

```
  SAMPLE.TXT          * * * P o w e r  T y p i n g * * *
once to cause the text to be inserted in the file.
This is some text which has been entered in Power Typing mode. The t
ext can wrap around across lines and the program will take care of s
plitting the stream of text on word boundaries.#We can use the Linen
d character to force a break in the text if we want to.
```

Figure 22. Entering Text in Power Typing Mode

```
  SAMPLE.TXT              TRUNC=255 SIZE=11 LINE=0
===>
00002 text below this line, which is the current line:
00003 We have now entered some text and pressed Return
00004 once to cause the text to be inserted in the file.
00005 This is some text which has been entered in Power Typing
00006 mode. The text can wrap around across lines and the program
00007 will take care of splitting the stream of text on word
00008 boundaries.
00009 We can use the Linend character to force a break in the text
00010 if we want to.
00011 * * * END OF FILE * * *









F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 23. Text on Exit from Power Typing

Using Function Keys

The Function keys (sometimes referred to as Program Function keys or PF keys) give you a handy way to quickly enter frequently-used CP-Edit commands. CP-Edit allows you to use 40 Function keys, all of which are settable.

Function keys 1 through 10 are obtained by just pressing the keys marked F1 through F10. Function keys 11 through 20 are obtained by holding down the Shift key and a Function key (for example, for F13, press Shift-F3). Function keys 21-30 are obtained by holding down the Ctrl key and a Function key (for example, for F25, press Ctrl-F5). And Function keys 31-40 are obtained by holding down the Alt key and a Function key (for example, for F40, press Alt-F10).

On the Enhanced (101-key) Keyboard, function keys 11 and 12 may be obtained just by pressing the keys marked F11 and F12.

The default Function key settings are shown in Appendix B, “Default Settings” on page 219. The first 10 are set for you. The rest are set to CP-Edit comments.

You can set the Function keys using the **SET PF**n command. You can use this command in the CP-Edit profile (PROFILE.MAC) so the Function keys will be automatically customized for you. For example, your profile might contain these commands:

```
SET PF11 TOP
SET PF12 BOTTOM
SET PF13 SPLIT CURSOR
SET PF14 QUIT
SET PF15 SET CURLINE ON 3
SET PF16 SET CURLINE ON 11
SET PF17 UP 22
SET PF18 NEXT 22
```

Or if you were doing some repetitive editing operation you might set some Function keys just for that operation. For example with these settings

```
SET PF9 locate /PROCEDURE/
SET PF10 change /PROCEDURE/FUNCTION/
```

you could repetitively locate the field "PROCEDURE" and selectively change it to "FUNCTION" just by pressing a couple of keys.

You may query the setting of a single PF key or of all the PF keys using the Query PF command:

```
Query PF14
Query PF *
```

The **Esc** key behaves like a Function key, but you cannot change its meaning. It always is set to a QUIT command. It is useful for exiting from HELP files and exiting from other files without saving them (like a QUIT command issued on the command line, the Esc key will not allow you to lose changes you have made to your file, however).

The **PgUp** and **PgDn** keys behave like Program Function keys also. They are used to scroll through your file. You can set the scroll amount to either a half-page (half the file area height) or a full page (the number of lines in the file area) using the **SET PGUP** and **SET PGDN** commands. Their initial settings are half-page, which is the same as the settings of the **PF7** and **PF8** keys.

The Function key settings for the first 10 Function keys are normally displayed on the bottom two lines of the screen. If you wish, you may turn this display off using the **SET DSPF** command, (allowing one additional line of text on the screen) or you may change the range of Function keys displayed using this command.

CP-Edit Macros

REXX Macros

CP-Edit includes an interface to REXX88. The full power of the REXX language is available to you in a CP-Edit macro written in REXX. REXX itself will not be described here. See the REXX88 Programmer's Guide for a description of the version of REXX implemented in REXX88. REXX88 Version 0.64 or later is required in order for CP-Edit's REXX interface to work properly. You must install the resident version of REXX88 (REXXSYS.SYS and REXXIBMR.EXE). See the REXX88 Programmer's Guide for information on installation of REXX88.

Macros may be invoked by just specifying the name of the macro and any parameters to be passed to the macro on the CP-Edit command line; or you may explicitly invoke a macro using the MACRO command. If your macro has the same name as a CP-Edit command, you **must** invoke it using the MACRO command.

CP-Edit normally searches for macros the same way DOS searches for commands to execute: it first searches the current directory, then if the macro is not found there, it searches the DOS PATH. You may supply an explicit path to be used in searching for the macro, whether you use the MACRO command or not. If you supply an explicit path, the normal path search is bypassed.

The implicit search for macros means that if you type in an invalid command, CP-Edit will first attempt to find an internal command with that name, and failing that, will search for a macro with that name. This is usually fairly fast, but if you are running from diskette, or have an unusually long PATH, this may take some time.

A REXX macro must begin with a REXX comment. If a macro does not begin with a REXX comment, CP-Edit assumes that it is a "native" macro (see below).

Several commands have been added to CP-Edit to exploit the REXX interface. The EXTRACT command allows you to access some CP-Edit internal parameters and information about the file you are editing. The MSG and EMSG commands give you a way (in addition to the REXX "say" command) to output messages on the screen. These three commands are **only** allowed in REXX macros.

Several options are available to you for writing messages from REXX macros. The SET MSGMODE and the SET PAUSE command control the way screen output behaves. You may output messages using the CP-Edit MSG and EMSG commands, or using the REXX "say" statement.

REXX output, including the trace output and output from the "say" statement, always goes to a separate screen. When (if) a macro first produces output directly from REXX, the screen is cleared and the output is written to that screen. The default is for that information to just scroll off the screen, and for the screen to be cleared and the file to be re-displayed when the macro is done executing. If you wish to be able to view the display while the macro is executing, you may want to use the SET PAUSE command. If you SET PAUSE ON, the display will pause after every screen full of information, and at the end of the macro, so you can view the information before it scrolls off the screen or is cleared. The SET PAUSE ON/OFF commands can be issued from the macro itself to control the behavior of the screen as required by the macro.

The location of the output from the CP-Edit MSG and EMSG commands, as well as the location of CP-Edit informational and error messages, is controlled by the second operand of the SET MSGMODE command. The default is SET MSGMODE ON IMMEDIATE. In this mode, the output from MSG and EMSG and CP-Edit error and informational messages goes to the separate scrolling screen along with any output from REXX "say" and trace, and is

affected in the same way by SET PAUSE. EMSG causes a bell character (beep) to be output, but is otherwise identical to MSG.

If you specify SET MSGMODE ON DELAY, the output from MSG and EMSG and CP-Edit error and informational messages produced during macro execution are saved and, upon completion of the macro, overlay the bottom few lines of the editing screen. This is the same as CP-Edit messages during normal editing. Output from MSG is written to the screen in the status area color, and output from the EMSG command is accompanied by a bell character and is written in the error message color. This mode may be preferable for macros that produce small amounts of output or normally produce no output and which you want to look transparently like a CP-Edit command.

It is possible to use the SET MSGMODE command within the macro itself to change the routing of messages during macro execution. You might, for example, SET MSGMODE ON DELAY as the normal case for a normal completion message, and SET MSGMODE ON IMMEDIATE to output a detailed error message. For commands which you did not want to product output (like TOP, for example), you might SET MSGMODE OFF.

In addition to MSG and EMSG, the CMSG command can be used to put a message or a command to be executed in the command line upon completion of the macro. If you wish to use the command line for a message which is not an executable command, it might be a good idea to start it with an asterisk ("*"), which is a CP-Edit comment, so that if the user presses Enter, the message on the command line will be ignored by CP-Edit.

Macros can invoke other macros. Due to the non-reentrant nature of DOS (and hence REXX88), you cannot just specify the name of the macro or use the CP-Edit MACRO command in a REXX macro. In fact, the MACRO command is disallowed from a REXX macro. However, you can use the REXX CALL command to invoke another macro. With the explicit CALL, REXX88 can provide nested macros.

If it is possible that a command you wish to execute from a REXX macro is has a synonym that is a macro, you may want to use COMMAND. This forces CP-Edit to bypass synonym resolution. This avoids the possibility of a command being disallowed because it is really a macro. This may be desirable in non-REXX macros also to avoid an unintentional recursion loop.

Note that this condition can arise easily if you write a macro to "front-end" a CP-Edit command and then invoke the command itself inside the macro. In this case you **must** use COMMAND.

Unfortunately, due to the non-reentrant nature of DOS (and hence REXX88), if you invoke CP-Edit from a REXX batch file you cannot execute REXX macros (including the profile). You can, however, invoke REXX macros from CP-Edit if it has been invoked from an ordinary DOS batch file. The return code operand of the the QUIT and QQUIT commands may be used to set the DOS ERRORLEVEL.

Another restriction due to the non-reentrant nature of DOS and REXX88 is that the CP-Edit DOS command may not be used from a REXX macro. DOS commands may be executed from REXX macros using the REXX **address** '**DOS**' command.

If you are executing a series of DOS commands from a REXX macro, you can use the 'address DOS' command with no operand to switch the default environment to DOS. When you are done executing DOS commands, you can use the 'address CPEDIT' command with no operand to switch the default environment back to CP-Edit. In the midst of a series of commands implicitly directed to DOS by the 'address DOS' command, you may use 'address CPEDIT cpeditcommand' to execute single CP-Edit commands.

The first time a REXX88 macro is executed, REXX88 tokenizes the macro and appends the tokenized form to the macro following the DOS end-of-file character. If you subsequently edit the macro, CP-Edit stops reading the macro at the end-of-file mark, so you should not ordinarily ever see the tokenized information. You should be aware of it, however, because

1. The first time a macro is executed, it will take longer because of the extra processing for the tokenization and writing the tokenized information;
2. The size of the macro file as shown by the DOS DIR command increases after it is executed, and will be larger than you might expect from the size of the text of the macro;
3. If you edit the macro and FILE it, the tokenized form is effectively removed; the length will be back to just that of the text; and REXX88 will re-tokenize the macro again the next time it is executed;
4. If you create or edit the macro with an editor which does not write the DOS end-of-file mark at the end of the file when storing the file on disk (including versions of CP-Edit prior to V3.10), the tokenized form will be visible if you subsequently edit the macro using current versions of CP-Edit; (In this case, you must delete the tokenized lines before you FILE.)
5. If you edit the macro with another editor which does not recognize the DOS end-of-file mark, or with a disk sector editor such as the NU program supplied as part of the Norton Utilities, you will see the tokenized information.

A MACRO command may be assigned to a Program Function key. This allows you to invoke a macro you use frequently with just one keystroke.

See the sample macros provided with CP-Edit for examples of REXX macros.

CP-Edit Native Macros

For compatibility with previous versions, CP-Edit still allows "native" macros. REXX88 is not required in order to use these macros. They are simple lists of commands. No branching or looping is possible with native macros.

CP-Edit distinquishes a REXX macro from a native macro by examining the first line of the macro. If the line begins with a REXX comment, CP-Edit assumes that it is a REXX macro. If not, it assumes that it is a native macro.

Ordinarily, if you need to write macros for use in CP-Edit, you would use REXX. However, you can still use the native macros even if REXX is not installed. Also, the native macros execute slightly faster than REXX macros, so if a simple sequential list of commands will perform the function you require you might use a native macro.

The Profile Macro

There is one special macro, the Profile macro, which is automatically executed for you just after loading each file you edit. You may provide the name of this macro on the DOS command line when you invoke CP-Edit, or if you do not provide it, CP-Edit looks for a file named PROFILE.MAC.

The search for the profile is the same as the search for any other macro. If an explicit path is given, that is used. If not, first the current directory is searched, then the DOS PATH.

You can modify the profile to turn CP-Edit into a batch editor (the last statement of the profile macro might be FILE to save the results of the changes made -- or see the END macro for an example of the use of the EXTRACT /ALT/ command to decide whether to FILE or QUIT.) Be sure to leave PAUSE set to OFF for batch editing.

Putting it All Together

Now let's put together the commands we have discussed into a typical editing session. First we insert the diskette containing CP-Edit, its Help files, the PROFILE.MAC file, and our text files into the default DOS drive. Then we begin editing a sample file using CP-Edit:

```
C:> CPEDIT sample.txt
```

After reading in the file into storage, CP-Edit executes the commands in the PROFILE.MAC file. Then it displays the first few lines of the file on the screen. Suppose we want to add some text to the end of the file. We move the current line to the bottom of the file and go into Input Mode:

```
bo
I
```

After entering some text, we press Return twice to get out of Input Mode. Now suppose we want to make some changes to a section of our file named "Chapter 3". We know the only place that string appears in the file is at the beginning of that section (it's unique), so we look for that string (backwards, since we are at the bottom of the file):

```
locate -/Chapter 3/
```

CP-Edit finds the line containing this string for us, and positions the current line there. We then use some full-screen editing to make some changes to this part of the file, possibly moving up or down in the file a screen or two using PgUp and PgDn. We may also use some prefix commands to insert or delete lines or move a paragraph around.

Now suppose we wish to refer to the contents of another file. We use the Edit command to bring the file into storage, browse through it using Locate, PgUp, and PgDn, then go back to our original file by just entering E on the command line.

Now suppose we want to change section 3.1 of the file to section 3.2 and change all references accordingly. We go to the top of the file:

```
TOP
```

We use a couple of global change commands to change all the references:

```
c /Section 3.1/Section 3.2/ * *
c /section 3.1/section 3.2/ * *
```

Now we want to insert a new section 3.1 in the file. We have already built that file on another diskette, which we now place in the A: drive. We happen to know that the section title starts in the first column of a line and it is the second such occurrence of that string in the file. We are at the top of the file still, so we enter the following commands:

```
f Section 3.2
=
up
get a:sect31.txt
```

The = command repeats the Find to find the second occurrence of the string. These commands find the section header for the new section 3.2 and bring in the new section 3.1 just before it.

We may make some more changes to our file, possibly referring to other files in the process. We might then issue the FILE command to save the file on disk. Then, since we have not changed any of the other files we have referred to, we get out of all of them at once and return to DOS by entering

```
cancel
```

If we had previously removed all the other files from storage, the FILE command would have returned us to DOS.

At any point, if the command we wanted to execute was assigned to a Function key, we could have executed the command just by pressing that Function key. We could also execute both a command on the command line and a Function key command together by typing in the command on the command line and pressing the Function key instead of Return. In this case, the Function key command is executed first, then the command on the command line.

CP-Edit Commands

This section contains the details of the use of the CP-Edit commands. These commands are entered on the command line of the screen (the second line from the top of the screen) following the arrow (===>), or assigned to PF keys.

Syntax

The syntax of the commands is specified in the following way: The name of the command is given in upper and lower case letters. The upper case portion gives the minimum abbreviation of the command name. The command may be entered using any number of characters of the name from the minimum to the full name of the command. The command name may be entered in upper or lower case.

Parameters for the command are shown in lower case following the command. Optional parameters are shown in square brackets ([]). The brackets may be nested. This is meant to imply that if an earlier parameter is not specified, later ones cannot be specified. If you have a choice of forms for a parameter, the choices are shown in square brackets below the first form.

Example:

```
Change /string1[/string2/  [target  [ p ]]]
                                    [ * ]
```

This is the Change command. Its minimum abbreviation is "C". The slash ("/") or some other unique delimiter and "string1" are a required parameter. The rest are optional parameters. If the second parameter "/string2/" is not specified, the following parameters may not be either. A number may specified for the "p" parameter or an "*" may be specified.

File Names

Filenames on commands that use them are in the normal DOS format: an optional drive letter, followed by an optional path, followed by the filename name part, followed by an optional extension.

Examples:

```
C:\CPEDIT\MACROS\PROFILE.MAC
A:MYFILE
B:\BACKUP\MYPROG.PAS
XYZ.TXT
ABC
\USR\BIN\MAKEFILE.BAT
```

For some commands (EDIT, GET, PUT, PUTD, FILE, SAVE), CP-Edit also allows you to use an equal-sign ("=") at the beginning of a filename in place of the drive and path to mean "the drive and path of the current file". For example if you were currently editing the file

```
C:\MYDIR\MYDATA.TXT
```

you could edit the file MYOTHER.TXT in the same directory by entering the following command:

```
E =MYOTHER.TXT
```

This would edit the file

```
C:\MYDIR\MYOTHER.TXT
```

Another example is the case where you want to GET a file from a subdirectory of the directory containing the current file. You could enter the following command:

```
GET =MYSUBDIR\MORESTUF.TXT
```

which would read in the file

```
C:\MYDIR\MYSUBDIR\MORESTUF.TXT
```

This shorthand notation is useful with the EDIT, GET, PUT, PUTD commands, etc. for working with several files at a time spread over two or more directories.

CP-Edit currently uses DOS 3.xx (and DOS 4.xx) rules when checking filenames for legal characters. DOS 3.xx and 4.xx are much less restrictive in this respect than was DOS 2.xx. CP-Edit will still run with DOS 2.xx, but if you attempt to use characters that DOS 2.xx considers illegal, the results are unpredictable, since the error-checking in DOS 2.xx itself is rather poor, and it may allow you to create a file that you cannot read or erase.

ADD COMMAND

```
Add   [n]
```

You may use the Add command to add one or more blank lines to your file. If no number is specified as the operand of the Add command, one line is inserted. The blank lines are inserted following the current line. The last added line becomes the current line.

Examples:

```
A 5
```

(Add 5 blanks lines following the current line. The last one becomes the current line.)

```
ADD
```

(Add a single blank line to the file.)

AUTOIN COMMAND

```
AUTOIn
AI
```

The AUTOIn command is used to switch to automatic indent mode. This mode is similar to Input mode, except that the cursor is automatically set to the column of the first non-blank character in the current line. Using the NEWLINE key positions the cursor to the same column. This mode is useful for entering text with indentations, and for entering structured programming source which uses indentation to show program structure. (This mode is similar to the input mode of the ISPF editor.)

When CP-Edit switches to automatic indentation input mode, the screen display is different from the usual editing screen. The portion of the screen below the current line becomes the input area. It is a blank area where text may be entered. Lines entered there are added to the file immediately following the current line. Only lines on which you have entered text (even a blank) are added to the file. If you skip lines (don't type in anything on them) they are not added to the file.

See the description of the Input command for additional information on Input mode.

Pressing the Return key once in input mode causes the text entered in the input area to be added to the file and a new blank input area to be displayed. The cursor is automatically positioned to the column which contains the first non-blank character on the line above the first line of the input area. Pressing the Return key twice (without entering text) causes CP-Edit to exit from input mode and go back to normal screen editing.

Examples:

```
AI
```

(Switch to automatic indent input Mode.)

BACKWARD COMMAND

```
BAckward  [ Half ]
```

The BAckward command moves the current line backward (toward the top of the file) by either a whole screen or half a screen.

Note that this implementation is different from that in XEDIT: the Half operand is added for CP-Edit, and it is not possible to move backward by multiple screens as it is with XEDIT.

The number of lines moved is adjusted to the size of the current logical screen. This command, however, does not take into account the fact that lines of a file may occupy more than one physical line on the screen.

See also the SET PGUP command.

Examples:

```
BACKWARD
```

(Moves the current line backward one screen.)

```
SET PF7 BACKWARD HALF
```

(Set PF7 to its default value, to move the current line backward by half the screen height.)

BOTTOM COMMAND

```
BOttom
```

The BOttom command moves the current line to the bottom of the file. The bottom of the file is a pseudo-line which follows the last real line of the file. Any insertions made when the current line is the bottom of the file go after the last real line of the file.

Example:

```
BOTTOM
```

CANCEL MACRO

```
CANCEL
```

The CANCEL macro does a QUIT from each file which is currently being edited and exits from CP-Edit to DOS.

Notice that a QUIT (a protected QUIT and not a QQUIT) is done for each file, so that if any file in storage has been changed, CP-Edit will not lose the changes. If any file has been changed since it was last saved, a FILE or QUIT for that file must be done before CP-Edit will exit to DOS.

Example:

```
CANCEL
```

CAPPEND MACRO

```
CAPPEND  text
```

The CAPPEND macro is used to append some text to the end of the current line. The column pointer is left at the first character of the appended text.

The CAPPEND macro is supplied as one of the sample macros with CP-Edit. Since it is written in REXX, you must have REXX installed in order to use it.

Example:

```
CAPPEND some text
```

(Append "some text" to the end of the current line.)

CDELETE COMMAND

```
CDelete [column-target ]
        [ 1 ]
```

The CDelete command is used to delete text from the current column to the column target. If no column target is specified, one column is deleted.

Examples:

```
CLOCATE :13
CDELETE 23
```

(Delete columns 13 through 22 on the current line.)

```
CDELETE
```

(Delete a single character at the current column position.)

CFIRST COMMAND

```
CFirst
```

The CFirst command is used to reset the column pointer to the left zone.

Examples:

```
SET ZONE 5 25
CFIRST
```

(Position the column pointer to column 5.)

```
SET ZONE 1 *
CFIRST
```

(Position the column pointer to column 1.)

CHANGE COMMAND

```
Change /string1[/string2/  [target  [ p ]]]
                                    [ * ]
```

The Change command is used to change parts of one or more lines of the file. String1 is changed to string2. Either string may be null. If string1 is null, string2 is just inserted at the beginning of each line. If string2 is omitted or null, string1 is deleted.

The delimiter for the strings is shown as a slash, but any character not appearing in the strings may be used. This allows changing any arbitrary string to any other arbitrary string.

The last two fields of the Change command can be used to specify the range of the changes to be made. The first of the two fields, the target, specifies the lines of the file over which the changes are to be made. Changes always begin with the current line of the file. If the target is a string expression, changes are made to all lines up to, but not including, the line which first satisfies the string expression. If the target is an absolute line number, changes are made up to, but not including, that line. If the target is a relative line number, changes are made over that many lines. If the target is omitted, the default is one line (the current one).

See the discussion of CP-Edit targets in Section “CP-Edit Targets” on page 33 for further details on the use of targets.

The last operand of the Change command may be used to specify the maximum number of occurrences of string1 which are changed to string2 on each line. If this number is omitted, 1 is assumed.

The current line pointer is not changed by the Change command.

Examples:

```
C /abc/xyz/
```

(Change the first occurrence of the string "abc" to "xyz" on the current line.)

```
C /This/That/ ~/bird/ *
```

(Change every occurrence of the string "This" to "That" on every line up to, but not including, the first line which does not contain the string "bird".)

```
CHANGE ;//;//**; *
```

(Change the first occurrence of the string "//" to "//**" on every line of the file starting with the current line.)

```
CH//%1 %2 / 3
```

(Insert the string "%1 %2 " at the beginning of 3 lines, starting with the current line.)

```
C '$$'' :22 3
```

(Delete the first 3 occurrences of the string "$$" on each line starting with the current line and continuing up to, but not including, line 22 of the file.)

```
Change /a/a/ * *
```

(Does nothing to the file, but does result in a message which will tell you how many occurrences of the letter "a" there are in the file and on how many lines it occurs.)

CINSERT COMMAND

```
CInsert text
```

The CInsert command is used to insert text at the current column.

Examples:

```
CLOCATE :13
CINSERT Put it here.
```

(Insert "Put it here." at column 13 of the current line.)

CLAST COMMAND

```
CLAst
```

The CLAst command is used to reset the column pointer to the end of the zone.

Examples:

```
SET ZONE 5 25
CLAST
```

(Position the column pointer to column 25.)

CLOCATE COMMAND

```
CLocate column-target
```

The CLocate command is used to position the current column pointer to the column target.

Column targets may be specified in the following ways:

1. An absolute column number, which is an integer preceded by a colon:

   ```
   CLOCATE :13
   ```

2. A relative column number, which is an integer preceded by a plus or minus, or just an integer (for which plus is assumed):

   ```
   CLOCATE +5
   CLOCATE 5
   CLOCATE -3
   ```

3. A simple string expression consisting of a string within delimiters optionally preceded by a plus ("+") or minus ("-") sign and/or a not-sign (the ASCII tilde, represented here by "~"):

   ```
   /string1/

   +/string1/

   -/string1/

   ~/string1/

   +~/string1/

   -~/string1/
   ```

The plus sign is always optional. It specifies movement forward (toward the end) of the line (or toward the end of the file, if SET STREAM ON has been specified). The minus sign specifies backward movement (toward the beginning) of the line or file.

The "~" is used to mean "not". A string expression containing "~" means "up to, but not including, the first column NOT containing the string". If both a plus or minus and a not-sign are used, the plus or minus must precede the not-sign.

Examples:

```
/BEGIN/

/PROCEDURE/

-~/*/
```

Examples:

```
CLOCATE :13
```

(Position the current column to column 13.)

```
CLOCATE +2
```

| (Position the column pointer 2 characters to the right.)

```
|     CLOCATE /XXX/
```

| (Position the column pointer to the beginning of the string "XXX".)

CMSG COMMAND

```
CMsg anytext
```

The CMsg command is used to put a message on the command line. It may be used to put a command on the command line to be executed just by pressing Enter. It is most useful from a macro, but may be used from the command line also.

Examples:

```
CMSG MACRO REDOIT
```

(This would put 'MACRO REDOIT' on the command line. The user could execute this command just by pressing Enter.)

```
CMSG  * This is a command line message.
```

(This would put the message on the command line. The message would be ignored if the user pressed Enter since it begins with an asterisk, which makes the message a CP-Edit comment.)

```
cmsg cmsg cmsg cmsg * Press Enter 4 times to remove this.
```

(Try it!)

COMMAND

```
COMMAND   command
```

Use COMMAND to force CP-Edit to bypass synonym resolution and not search for a macro. The command is executed only if it is a built-in CP-Edit command. This is useful for overriding synonyms and avoiding unintentionally invoking a macro from a macro.

COVERLAY COMMAND

```
COVerlay text
```

The COVerlay command is used to overlay text at the current column. Only non-blank characters in the text replace corresponding characters in the original line contents. The underline character ("_") may be used to overlay blanks onto the line.

Examples:

```
CLOCATE :13
COVERLAY Put it here.
```

(Overlay "Put it here." at column 13 of the current line.)

CREPLACE COMMAND

```
CReplace text
```

The CReplace command is used to replace text at the current column. The text may contain blanks, which will replace existing characters in the line.

Examples:

```
CLOCATE :13
CREPLACE Put it here.
```

(Put "Put it here." at column 13 of the current line.)

CURSOR COMMAND

```
CURsor   CMdline [colno | 1 ] [Priority n]
         File    lineno   [colno] [Priority n]
         Home    [Priority n]
         Screen  lineno   [colno] [Priority n]
```

The CURsor command is used to position the cursor on the screen. It is most useful in macros, or when assigned to a PF Key, or when used with the SET ENTER command.

The CMdline operand positions the cursor to the command line on the screen.

The File operand positions the cursor within the file if the line and column specified is on the screen.

The Home operand positions the cursor to the command line if the cursor is in the file area, or positions the cursor to the file area (generally the current line) if the cursor is on the command line.

The Screen operand positions the cursor to the specified position on the screen.

The priority setting determines the priority of cursor positioning compared to other operations which would position the cursor. The default if the priority operand is not specified is absolute positioning (the CURsor command overrides any other source of cursor positioning). If multiple CURsor commands with identical priorities are specifed, the last one issued is the one which is effective.

DEFINE COMMAND

```
DEFine  keyname [ = ] function [ ; ]
```

The DEFine command is used to redefine the use of keys on the keyboard. The keyname parameter specifies the key to be redefined, and the function parameter specifies the editing function or character to be assigned to the key.

The key being reassigned may be a single ASCII character or a keyname.

Key names are based on the US English keyboards.

Case

Case is significant for the single alphabetic characters representing either keystrokes or key definitions. Otherwise case is not significant on "define" statements. For example, you must use "T" if you mean uppercase T and "t" if you mean lowercase t, but you may use DEFINE, or Define, or define to mean "define", and you may use Asterisk, or ASTERISK or asterisk (or AsTeRiSk, for that matter) to mean the asterisk ("*") keystroke or key definition.

Keynames on Left Side of Define Statement

These are the keynames which can appear on the left side of a define statement:

Keys for Enhanced Keyboard

Top Row of Keys

```
Esc   f1  f2  f3  f4  f5  f6  f7  f8  f9  f10  f11  f12  PrtSc
```

Typewriter Keyboard Keys

```
` 1 2 3 4 5 6 7 8 9 0 - = Backspace
tab q w e r t y u i o p [ ] \
     a s d f g h j k l ; ' Enter
      z x c v b n m , . /
         Spacebar
```

Shifted Keyboard Keys

```
~ ! @ # $ % ¬ & * ( ) _  +
Backtab                { } ¦
                     : "
                   < > ?
```

Keys Between Typewriter Keyboard and Pad

```
  Insert Home PageUp
  (Ins)       (PgUp)

  Delete End  PageDown
  (Del)       (PgDn)


        up
 left  down  right
```

| Note that the Insert, Home, PageUp, PageDown, Delete, and End keys return identical scan codes to the similarly named ones on the numeric pad, and CP-Edit cannot distinguish between them. The different names are only for convenience.

Keys on Numeric Pad

```
        padslash padstar padminus
  home  up       pgup    padplus
  left  pad5     right
  end   down     pgdn    padenter
```

Keys at Lower Right of Keyboard

```
  ins  del
```

Keys for Original (PC-1) Keyboard

Function Keys

```
f1  f2
f3  f4
f5  f6
f7  f8
f9  f10
```

Typewriter Keyboard Keys

```
Esc 1 2 3 4 5 6 7 8 9 0 - = Backspace
tab  q w e r t y u i o p [ ]
      a s d f g h j k l ; ' ` Enter
     \ z x c v b n m , . /
         Spacebar
```

Shifted Keyboard Keys

```
     ! @ # $ % ¬ & * ( ) _  +
Backtab                 { }
                       : " ~
     ¦              < > ?
```

Keys on Numeric Pad

```
         home  up       pgup    padminus
PrtSc    left  pad5     right
padstar  end   down     pgdn    padplus
```

Keys at Lower Right of Keyboard

```
ins  del
```

Key Name Aliases

Sorted By Key Name

!	exclamation	
"	doublequote	
#	pound	poundsign
$	dollar	dollarsign
%	percent	
&	ampersand	
'	quote	singlequote
(	lparen	leftparen
)	rparen	rightparen
*	asterisk	star
+	plus	
,	comma	
-	minus	
.	period	point
/	slash	
:	colon	
;	semicolon	
<	less	lessthan
=	equal	
>	greater	greaterthan
?	question	questionmark
@	at	atsign
backspace	bksp	
[	lbracket	leftbracket
\	backslash	
]	rbracket	rightbracket
¬	caret	
_	underline	
`	backquote	grave
{	lbrace	leftbrace
¦	bar	
}	rbrace	rightbrace
~	tilde	

Figure 24. Key Name Aliases (Sorted By Key Name)

Sorted By Alias

```
&           ampersand
*           asterisk       star
@           at             atsign
`           backquote      grave
\           backslash
¦           bar
backspace   bksp
¬           caret
:           colon
,           comma
$           dollar         dollarsign
"           doublequote
=           equal
!           exclamation
>           greater        greaterthan
{           lbrace         leftbrace
[           lbracket       leftbracket
<           less           lessthan
(           lparen         leftparen
-           minus
%           percent
.           period         point
+           plus
#           pound          poundsign
?           question       questionmark
'           quote          singlequote
}           rbrace         rightbrace
]           rbracket       rightbracket
)           rparen         rightparen
;           semicolon
/           slash
~           tilde
_           underline
```

Figure 25. Key Name Aliases (Sorted By Alias)

Key Name Usage Notes

Some key names may be modified using shift- for the shifted state, ctrl- for the control shift state, or alt- for the alt shifted state. Many of these combinations are invalid (for example, shift-ampersand, since the ampersand appears on the shifted 7 key), or represent keystrokes which are just not returned by the BIOS keyboard interrupt (such as shift-prtsc), or which are not unique (for example, the BIOS returns exactly the same value for the space bar, shift-space, ctrl-space, and alt-space, so this command allows you to specify only the unshifted space).

See the table of key combinations below for precise documentation on what key combinations are accepted by this command.

Note that there are a number of keystrokes that are allowed by this command that are not returned by the BIOS for the original PC-1 keyboards and the old AT keyboards, such as the pad5, and alt-home; and of course there are some keystrokes, such as f11 and f12, that are not produced by the older keyboards because the keys do not exist.

If this command does not provide enough flexibility to remap the keyboard to your liking, the use of a general-purpose keyboard remapping utility, such as KBDX, is recommended. Because CP-Edit interfaces with the key-

board through the BIOS (for DOS) or using KbdXxx functions (for OS/2), it is not possible to see all keystrokes or to distinguish between some keystrokes produced by different shift states of the same key.

Keynames on Right Side of Define Statement

These are the functions which can appear on the right side of a define statement:

ASCII Characters

```
blank (space)
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ¬ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { ¦ } ~
```

Character Aliases

The following are character aliases which may be used:

!	exclamation	
"	doublequote	
#	pound	poundsign
$	dollar	dollarsign
%	percent	
&	ampersand	
'	quote	singlequote
(	lparen	leftparen
)	rparen	rightparen
*	asterisk	star
+	plus	
,	comma	
-	minus	
.	period	point
/	slash	
:	colon	
;	semicolon	
<	less	lessthan
=	equal	
>	greater	greaterthan
?	question	questionmark
@	at	atsign
[	lbracket	leftbracket
\	backslash	
]	rbracket	rightbracket
¬	caret	
_	underline	
`	backquote	grave
{	lbrace	leftbrace
¦	bar	
}	rbrace	rightbrace
~	tilde	

Figure 26. ASCII Character Aliases

Editing Functions

Name	Aliases	Function performed
f1 - f40	(pf1 - pf40)	Program Function keys.
esc	(escape)	
ins	(insert)	
del	(delete)	
home		
end	(rightmost)	Goes to end of line
pgup	(pageup)	
pgdn	(pagedown)	
up	(uparrow, cursorup)	cursor up
down	(downarrow, cursordown)	cursor down
left	(leftarrow, cursorleft)	cursor left
right	(rightarrow, cursorright)	cursor right
backspace		same as cursor left
newline		
enter	(Return)	
eraseeof	(eraseeol)	
center	(pad5 or curline)	
bottom		Goes to bottom of screen.
leftmost		
desbksp		Destructive Backspace
xtab		XEDIT-style tab
xbacktab		XEDIT-style backtab
xtabcmd		tab to command line
xtabcmdf		tab to next screen command line
xtabcmdb		tab to previous screen cmd line
wordtab	(wordtabf, ctrlright)	forward word tab
wordtabb	(ctrlleft)	backward word tab
clear		XEDIT Clear key
dead	(nothing)	Ignored

Figure 27. Editing Functions

In addition, an arbitrary hex character (x'01'-x'ff') may be used. Example:

```
define alt-q = x'DA' ;    /* upper left box char. */
```

Any hex character from x'01' through x'FF' may be used, but note that some hex characters may cause problems. In particular, be careful of x'1a', which is the DOS end-of-file character; putting this character into the file will cause the file to be truncated at that point next time the file is read by CP-Edit or any other program that recognizes the DOS end-of-file character. Note also that x'00' is reserved to mean a null character and is ignored on input by CP-Edit.

There are some functions, such as the destructive backspace (desbksp), the XEDIT Clear key (clear), and the XEDIT-style tabs (xtab and xbacktab) which are by default not assigned to any key in CP-Edit. In order to use them at all, you must assign them to a key with the DEFINE command.

Summary of Keys and Characters

ASCII Collating Sequence

On Left Side of a Define Statement: The following is a summary in ASCII collating sequence of all the keys that can appear on the left side of a define statement:

	!	U	f11	pf2
	"	V	f12	pf3
	#	W	f2	pf4
	$	X	f3	pf5
	%	Y	f4	pf6
	&	Z	f5	pf7
	'	[	f6	pf8
	(	\	f7	pf9
	)	]	f8	pgdn
	*	¬	f9	pgup
	+		g	plus
	,	`	grave	pound
	-	a	greater	poundsign
	.	ampersand	greaterthan	prtsc
	/	asterisk	h	q
	0	at	home	question
	1	atsign	i	questionmark
	2	b	ins	quote
	3	backquote	insert	r
	4	backslash	j	rbrace
	5	backspace	k	rbracket
	6	backtab	l	right
	7	bar	lbrace	rightarrow
	8	bksp	lbracket	rightbrace
	9	blank	left	rightbracket
	:	c	leftarrow	rightparen
	;	caret	leftbrace	rparen
	<	colon	leftbracket	s
	=	comma	leftparen	semicolon
	>	cursordown	less	singlequote
	?	cursorleft	lessthan	slash
	@	cursorright	lparen	space
	A	cursorup	m	spacebar
	B	d	minus	star
	C	del	n	t
	D	delete	o	tab
	E	dollar	p	tilde
	F	dollarsign	pad5	u
	G	doublequote	padasterisk	underline
	H	down	padenter	up

| Figure 28. ASCII Collating Sequence (Left Side of Define Statement - Part I)

```
|       I             downarrow      padminus       uparrow
|       J             e              padplus        v
|       K             end            padslash       w
|       L             enter          padstar        x
|       M             equal          pagedown       y
|       N             equals         pageup         z
|       O             esc            percent        {
|       P             escape         period         ¦
|       Q             exclamation    pf1            }
|       R             f              pf10           ~
|       S             f1             pf11
|       T             f10            pf12
```

Figure 29. ASCII Collating Sequence (Left Side of Define Statement - Part II)

On Right Side of a Define Statement

The following is a summary in ASCII collating sequence of all the key definitions that can appear on the right side of a define statement:

!	atsign	f32	pf3
"	b	f33	pf30
#	backquote	f34	pf31
$	backslash	f35	pf32
%	backspace	f36	pf33
&	backtab	f37	pf34
'	bar	f38	pf35
(	bell	f39	pf36
)	bksp	f4	pf37
*	blank	f40	pf38
+	bottom	f5	pf39
,	c	f6	pf4
-	caret	f7	pf40
.	center	f8	pf5
/	clear	f9	pf6
0	colon	g	pf7
1	comma	grave	pf8
2	curline	greaterthan	pf9
3	cursordown	h	pgdn
4	cursorleft	home	pgup
5	cursorright	i	plus
6	cursorup	ins	point
7	d	insert	pound
8	dead	j	poundsign
9	del	k	q
:	delete	l	question
;	desbksp	lbracket	questionmark
<	dollar	left	quote
=	dollarsign	leftarrow	r
>	doublequote	leftbrace	rbracket
?	down	leftbracket	return
@	downarrow	leftmost	right
A	e	leftparen	rightarrow
B	end	lessthan	rightbrace
C	enter	m	rightbracket
D	equal	minus	rightmost
E	equals	n	rightparen
F	eraseeof	newline	s
G	eraseeol	nothing	semicolon
H	esc	o	singlequote
I	escape	p	slash
J	exclamation	pad5	space
K	f	pagedown	splat
L	f1	pageup	star
M	f10	percent	t
N	f11	period	tab
O	f12	pf1	tilde
P	f13	pf10	u
Q	f14	pf11	underline
R	f15	pf12	up

Figure 30. ASCII Collating Sequence (Right Side of Statement) - Part I

```
|       S             f16             pf13            uparrow
|       T             f17             pf14            v
|       U             f18             pf15            w
|       V             f19             pf16            wordtab
|       W             f2              pf17            wordtabb
|       X             f20             pf18            wordtabf
|       Y             f21             pf19            x
|       Z             f22             pf2             xbacktab
|       [             f23             pf20            xtab
|       \             f24             pf21            xtabcmd
|       ]             f25             pf22            xtabcmdb
|       ¬             f26             pf23            xtabcmdf
|       _             f27             pf24            y
|       `             f28             pf25            z
|       a             f29             pf26            {
|       ampersand     f3              pf27            ¦
|       asterisk      f30             pf28            }
|       at            f31             pf29            ~
```

| Figure 31. ASCII Collating Sequence (Right Side of Statement) - Part II

| Examples:

```
|     define
```

| (Define

```
|     def
```

| (Define

DELETE COMMAND

```
DELete [ target | 1 ]
```

The DELete command is used to delete lines of the file. The number of lines specified by the target are deleted, starting with the current line. If the target is omitted, one line (the current one) is deleted.

The DELete command deletes only in the forward direction. Targets preceded by minus signs are not allowed.

The prefix delete commands may also be used to delete lines of the file. See the prefix commands in Section "CP-Edit Prefix Commands" on page 193 below.

DOS COMMAND

```
DOS [,]   [ doscommand ]
```

The DOS command is used to execute DOS commands from CP-Edit. The DOS command to be executed may be entered on the command line following "DOS ". If no DOS command is entered, DOS command mode is entered, in which you may enter any number of DOS commands. To return to CP-Edit from this mode, enter the DOS EXIT command.

After executing the DOS command and before re-displaying the current editing screen, CP-Edit normally displays the message "Press any key to continue". This is to allow you to view any messages that may have been output by the DOS command. When you press any key, the most recent editing screen is re-displayed. If, however, you enter a comma (",") following "DOS" and preceding the DOS command, CP-Edit does not pause. This is useful if you are executing DOS commands from CP-Edit macros. CP-Edit does not pause when returning to CP-Edit from DOS command mode after the DOS EXIT command has been entered.

Most DOS in-storage and disk-resident commands and application programs may be executed from within CP-Edit. But since some programs may not be compatible with CP-Edit, it is possible that some program will not execute properly from within CP-Edit.

In general, programs which make themselves resident extensions of DOS will not work well from CP-Edit. These programs should be started before starting CP-Edit. For example, FileCommand does not start up properly from within CP-Edit (and in fact, causes storage chains to be destroyed so that a re-boot may be necessary, although CP-Edit files are not lost; do not attempt to start FileCommand from within CP-Edit). Using FileCommand with CP-Edit works nicely if you start FileCommand first, then invoke CP-Edit from the FileCommand menu. If DOS command mode is then entered from CP-Edit, the FileCommand menu is re-displayed.

One type of DOS command executes with no errors from within CP-Edit, but ultimately has no effect. Any DOS command which affects only the current environment will have no permanent effect, since a copy of the current environment is passed to the new invocation of COMMAND.COM by CP-Edit, but there is no provision for making the changes to the current environment known to CP-Edit. Because of this, when executed using the CP-Edit DOS command, the effect of such commands as PATH, SET, and PROMPT is lost upon return to CP-Edit.

What programs execute from within CP-Edit depends in part on the amount of memory available. Some minimum amount of storage must be available to execute any DOS command at all. The amount required depends on the DOS version -- see the table below for minimum values for the various DOS versions. If this minimum amount is not available, an error message results. If there is not enough memory to load the particular program you are trying to execute, DOS will give you an error message.

The following table lists the minimum free storage for DOS command execution:

DOS Version	Minimum	Default
DOS 2.x	21KB	24KB
DOS 3.x	26KB	30KB
DOS 4.x	38KB	40KB
SWAPEXEC	8KB	(DOS minimum)

If more storage is available, more will be available for execution of DOS commands.

You may use SWAPEXEC to execute commands instead of DOS's COMMAND.COM. The default is to *not* use SWAPEXEC. You may change this setting using the SET SWAPEXEC command:

```
  set swapexec on
or
  set swapexec off
```

See the SET SWAPEXEC command for details.

Obviously, SWAPEXEC must be installed on your system to take advantage of this option -- it must be present in some directory in your DOS PATH (or present in the current directory on the current drive). With this option selected, you will have a lot more memory in which to execute DOS commands: you will have the amount you had before loading CPEDIT, less about 8KB which is required for the resident portion of SWAPEXEC.

Note that there are some disadvantages to using SWAPEXEC: you must have enough space on some drive to store the swapped-out storage. Swapping to VDISK is recommended -- you can swap to any DOS disk, but swapping to anything other than a VDISK will be much slower. There is still a small delay even when swapping to VDISK, but it's probably not really noticable compared to invoking COMMAND.COM. Putting SWAPEXEC itself on a VDISK is highly recommended also (the VDISK should be in your DOS PATH).

When the SWAPEXEC option is used, the minimum free storage for executing any DOS command is about 8KB.

Another problem is that REXX doesn't know to use SWAPEXEC. If you want to switch permanently to using SWAPEXEC, you can modify all your REXX macros which execute DOS commands by preceding each command with "SWAPEXEC":

```
change
  address 'DOS' 'CHKDSK C:'
to
  address 'DOS' 'SWAPEXEC CHKDSK C:'
```

Because of the way DOS commands are executed from a Borland C program (which is the compiler currently used to compile CP-Edit Version 5 and later for DOS), the amount of memory available to execute DOS commands is unpredictable, and depends on the pattern of memory useage in CP-Edit prior to invocation of the DOS command. It is possible, for example, to fill memory then empty it (load in a large file and FILE or QUIT from it) and have memory in such a state that it is impossible to execute a DOS command. Use of SWAPEXEC is highly recommended to minimize the possibility that this will happen.

When CP-Edit is *not* using SWAPEXEC, it executes DOS commands by loading a secondary copy of the DOS command processor, COMMAND.COM. It attempts to load the command processor using the information in the COMSPEC= parameter of the current environment. If this information is incorrect (COMMAND.COM is not present in the specified disk or directory), an error message results and no DOS commands will be executed. If there is no COMSPEC= parameter in the current environment, CP-Edit assumes that it should load COMMAND.COM from the current directory of the current drive. If no copy of COMMAND.COM is present there, an error message results. See the DOS reference manual for further details on the current environment and the DOS SET command.

Examples:

```
DOS
```

(Go into DOS command mode. Enter DOS commands. Enter "EXIT" to return to CP-Edit.)

```
DOS , ERASE ABC.XYZ
```

(Erase the file "ABC.XYZ". Do not pause to display the DOS screen before re-displaying the current CP-Edit screen.)

```
DOS DIR B: | SORT > B.DIR
```

(Produce a sorted directory listing of the B: disk in a file. The file may then be edited with CP-Edit if desired.)

DOWN COMMAND

```
Down    [ n ]
        [ * ]
```

The Down command moves the current line forward (toward the end of the file) one or more lines. If the number of lines is omitted, 1 is assumed.

Down * is equivalent to the BOttom command. The Down command is indentical in function to the Next command.

Examples:

```
DOWN
```

(Moves the current line forward one line.)

```
D 20
```

(Moves the current line forward 20 lines.)

EDIT COMMAND

```
Edit   [[d:][path]fname[.ext]]
```

The Edit command is used to begin editing another file.

The filename is an optional parameter of the Edit command. If you supply a filename as part of the Edit command, that file will be edited. If you don't supply one, the next file in the ring will be edited. If there is only one file in storage, Edit with no filename does nothing. If the file specified on the Edit command is in storage, CP-Edit will switch to it immediately. If it is not in storage, CP-Edit attempts to read it in from disk. If it is not found on the disk, a new file will be created in storage.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

If only one diskette drive is installed, the drive should be used only as the A: drive with CP-Edit. The logical B: drive should not be used.

If no filename is specified, the next file in the ring of files in storage is edited. (The files are chained together in the order in which they are edited. When the end of the chain is reached, CP-Edit starts over at the beginning of the chain.)

The number of files in storage can be determined at any time by using the **Q**uery **NBF**ile command. The number and names of files in storage can be determined by using the **Q**uery **RING** command.

If the function key display is turned of **(SET DSPF OFF)**, the status line at the bottom of the screen displays the number of files currently in storage.

Examples:

```
E autoexec.bat
```

(Edit the autoexec.bat file on the default drive in the default directory.)

```
EDIT
```

(Switch to the next file in storage.)

```
E C:\CPEDIT1\CPEDIT.PAS
```

(Edit the file CPEDIT.PAS in the CPEDIT1 directory on the C: fixed disk.)

EMSG COMMAND

```
EMSG anytext
```

The EMSG command is used in REXX macros to output an error message. It outputs an ASCII bell character and writes the text of the message directly to the screen (if MSGMODE is ON IMMEDIATE), or saves it for display overlaying the editing screen upon macro completion (if MSGMODE is ON DELAY). In DELAY mode, the error message is displayed in the error message color. The EMSG command may only be used in a REXX macro.

Example:

```
EMSG 'Not a valid target:' target1
```

END COMMAND

```
END
```

The END command performs the equivalent of a FILE command for the current file if the file has been changed since it was last saved, or a QUIT command if the file has not been changed since it was last saved.

Example:

```
END
```

(FILE if the current file has been changed, otherwise QUIT.)

EXTRACT COMMAND

```
EXTract /operand [/operand [/operand ... ]]
```

The operand may be any of the keywords below:

```
ALT       FName     NUMber    TABS
AUtosave  HEX       PF        TARGet
CASE      IMAGE     PREfix    TOL
COLUMN    LENgth    RING      TRUNC
CURLine   LIne      SCReen    Verify
CURSor    LScreen   SIZe      WRap
EOL       MSGMode   STREAM    Zone
FLscreen  NBFILE    SYNonym   =
```

The EXTract command is used in REXX macros to directly set REXX variables to the value of various CP-Edit editing parameters or file values. The EXTract command may only be used in a REXX macro.

The following are the values of the variables returned to the REXX macro by the EXTract command. The variables are all REXX compound variables of the form "name.n", where "name" is the same as the EXTract command operand, and "n" is an integer subscript. "name.0" is always the number of other subscripted variables returned. See the REXX88 Programmer's Guide or other REXX reference for further information on the use of REXX and REXX compound variables.

ALT

Returns the number of alterations that have been made to the file since the last AUTOSAVE and SAVE, or as specified in the last SET ALT command.

```
ALT.0    number of variables returned (2)
ALT.1    number of changes since last AUTOSAVE
ALT.2    number of changes since last SAVE
```

AUtosave

Returns the current SET AUTOSAVE count, the filename of the current AUTOSAVE file and the number of alterations that have been made to the file since the last AUTOSAVE.

```
AUTOSAVE.0 number of variables returned (3)
AUTOSAVE.1 OFF or autosave count
AUTOSAVE.2 autosave filename
AUTOSAVE.3 number of alterations since last autosave
```

CASE

Returns the current CASE settings.

```
CASE.0    number of variables returned (2)
CASE.1    MIXED or UPPER
CASE.2    RESPECT or IGNORE
```

COLumn

Returns the current column number.

```
COLUMN.0    number of variables returned (1)
COLUMN.1    current column number
```

CURLine

Returns line number on the screen and contents of the current line of the file.

```
CURLINE.0    number of variables returned (5)
CURLINE.1    M ff‹+n|-n“ ff‹-“ n (M = middle of screen)
CURLINE.2    actual line number on the screen
CURLINE.3    contents of current line
CURLINE.4    OFF (not implemented)
CURLINE.5    OLD (not implemented)
```

CURSor

Returns the current and original position of the cursor on the screen (line number and column number), the current and original position of the cursor in the file and the priority. The current position is where the cursor will be displayed the next time the screen is refreshed. The original position is where the cursor was when the screen was last read.

```
CURSOR.0    number of variables returned (9)
CURSOR.1    current line number of cursor on screen
CURSOR.2    current column number of cursor on screen
CURSOR.3    current line number of cursor in file
CURSOR.4    current column number of cursor in file
CURSOR.5    original line number of cursor on screen
CURSOR.6    original column number of cursor on screen
CURSOR.7    original line number of cursor in file
CURSOR.8    original column number of cursor in file
CURSOR.9    priority
```

EOL

Returns the current EOL state. EOL is ON when the column pointer reaches the right zone + 1.

```
EOL.0    number of variables returned (1)
EOL.1    ON or OFF
```

FLscreen

Returns the first and last lines of the file currently being displayed on the screen. If the returned line numbers are zero, no file is displayed in the current screen or there is no valid current screen.

```
FLSCREEN.0   number of variables returned (2)
FLSCREEN.1   file line number of first file line on screen
FLSCREEN.2   file line number of last file line on screen
```

The EXTRACT /FLscreen/ command is not included in the DOS version of CP-Edit.

FName

Returns the name of the file currently being edited in two forms: as the exact filename as specified by the user (and as it appears on the display screen), and parsed, with the actual drive letter, actual path, 1 to 8 character name part of the filename, and the 0 to 3 character extension.

```
FNAME.0    number of variables returned (5)
FNAME.1    user-specified fileid
FNAME.2    actual driveletter
FNAME.3    actual path
FNAME.4    name part
FNAME.5    extension
```

Note that the values returned by EXTRACT /FNAME/ in CP-Edit are somewhat different from those returned by XEDIT, reflecting the different operating system environment.

HEX

Returns the current HEX setting.

```
HEX.0    number of variables returned (1)
HEX.1    ON or OFF
```

IMAGE

Returns the current IMAGE setting.

```
IMAGE.0    number of variables returned (1)
IMAGE.1    ON or OFF
```

LENgth

Returns the length of the current line.

```
LENGTH.0    number of variables returned (1)
LENGTH.1    length of current line
```

LIne

Returns the line number (in the file) of the current line.

```
LINE.0    number of variables returned (1)
LINE.1    line number of current line
```

LScreen

Returns the dimensions of the current logical screen and the size of the display (DOS character-mode) or window (OS/2, X-windows).

```
LSCREEN.0    number of variables returned (6)
LSCREEN.1    Height of logical screen in lines
LSCREEN.2    Width of logical screen in characters
LSCREEN.3    Line number of top left corner
LSCREEN.4    Column number of top left corner
LSCREEN.5    Number of lines in display or window
LSCREEN.6    Number of columns in display or window
```

MSGMode

Returns the current MSGMode setting.

```
MSGMODE.0  number of variables returned (2)
MSGMODE.1  ON or OFF
MSGMODE.2  IMMEDIATE or DELAY
```

NBFILE

Returns the number of files currently being edited (number of files in the ring).

```
NBFILE.0   number of variables returned (1)
NBFILE.1   number of files being edited.
```

NUMber

Returns the state of the NUMBER setting of the file (as set by the SET NUMBER command).

```
NUMBER.0   number of variables returned (1)
NUMBER.1   "ON" or "OFF"
```

```
PF n
PF *
PF
```

Returns one or all PF key settings.

The PF n (or PFn) form returns the following information for the single PF key:

```
PFn.0      number of variables returned (0 or 2)
PFn.1      BEFORE
PFn.2      PFn setting
```

The PF, PF *, or PF* form returns the following information for all 40 CP-Edit PF keys:

```
PF1.0      number of variables returned (0 or 2)
           for PF1
PF1.1      BEFORE
PF1.2      PF1 setting
PF2.0      number of variables returned (0 or 2)
           for PF2
PF2.1      BEFORE
PF2.2      PF2 setting
  .
  .
  .
PF40.0     number of variables returned (0 or 2)
           for PF40
PF40.1     BEFORE
PF40.2     PF40 setting
```

If a PF key is undefined, PFn.0 is set to 0 and no other variables are set for that PF key.

PREfix

Returns the state of the prefix setting of the file (as set by the SET PREFIX command). (The PREFIX SYNONYM form of EXTRACT /PREFIX/ is not currently supported.)

```
PREFIX.0   number of variables returned (1)
PREFIX.1   "ON" or "OFF"
PREFIX.2   "RIGHT" or "LEFT"
```

RING

Returns the number files that are currently in the ring and the names and attributes of the files.

```
RING.0       number of variables returned (= RING.1 + 1)
RING.1       number of files in the ring.
RING.2       file identification line of first file in ring.
 ...
RING.n       file identification line of last file in ring.
```

The file identification line returned is the same as that displayed at the top of the screen when the file is displayed on screen, except that the file name will always be the full user-specified name of the file (equivalent to FNAME.1).

SCReen

Returns the attributes of screens that have been defined using the SET SCREEN command.

```
SCREEN.0     number of variables returned (1)
SCREEN.1     screen definition: SIZE, WIDTH, or DEFINE
```

Note that the result of this extract will always be in terms of SIZE (if a single logical screen or horizontal splits are defined), WIDTH (if vertical splits are defined), or DEFINE (if the SET SCREEN DEFINE is used to define arbitrary splits). For example, for a 25-line screen, if SET SCREEN 2 was the command issued, SCREEN.1 would be

```
SIZE 13 12
```

SIZe

Returns the number lines in the file currently being edited.

```
SIZE.0     number of variables returned (1)
SIZE.1     number lines in current file
```

STReam

Returns the state of the STREAM setting of the file (as set by the SET STREAM command).

```
STREAM.0   number of variables returned (1)
STREAM.1   "ON" or "OFF"
```

SYNonym name

Returns the synonym, its minimum abbreviation length, and the synonym definition.

```
SYNONYM.0  number of variables returned (3)
SYNONYM.1  synonym name
SYNONYM.2  length of minimum abbreviation
SYNONYM.3  definition
```

The form EXTRACT /SYNONYM */ is not supported.

TARGet

Returns the line and column number of the most recent target found.

```
TARGET.0  number of variables returned (normally 4)
TARGET.1  line number of start of target
TARGET.2  column number of start of target
TARGET.3  line number of end of target (always = TARGET.1)
TARGET.4  column number of end of target
```

If no target has been used in this file, TARGET.0 = 0.

Unlike XEDIT, *any* target used in any command will change the contents of EXTRACT /TARGet/.

TABS

Returns the current tab column numbers.

```
TABS.0    number of variables returned (1)
TABS.1    tab columns
```

TOL

Returns the current TOL state. TOL is ON when the column pointer reaches the left zone - 1.

```
TOL.0    number of variables returned (1)
TOL.1    ON or OFF
```

TRUNC

Returns the current truncation column number.

```
TRUNC.0    number of variables returned (1)
TRUNC.1    current truncation column
```

Verify

Returns the current verify column settings and VERIFY ON/OFF setting.

```
VERIFY.0    number of variables returned (2)
VERIFY.1    OFF (VERIFY ON not implemented in CP-Edit)
VERIFY.2    verify columns
```

WRap

Returns the current WRap setting.

```
WRap.0      number of variables returned (1)
WRap.1      ON or OFF
```

Zone

Returns the current zone column settings.

```
ZONE.0    number of variables returned (2)
ZONE.1    left zone
ZONE.2    right zone
```

= (Equal)

Returns the current contents of the equal buffer (normally the name of the last macro which was invoked from the command line.)

```
EQUAL.0    number of variables returned (1)
EQUAL.1    contents of = buffer
```

FILE COMMAND

```
FILE   [ [d:][path]filename[.ext] ]
```

The FILE command saves the current file and exits the current file. If the current file is the only file in storage, CP-Edit ends and exits to DOS. If one or more other files are in storage, the current file is removed from storage and CP-Edit switches to the next file in the chain of files in storage.

A DOS filespec may be specified on the FILE command. If it is, the file is saved on disk with that name. If it is not, the current name of the file as displayed on the ID line of the screen is used.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

Examples:

```
file
```

(Save the current file and exit from it. If it is the only file, exit to DOS.)

```
file C:\tedit2\jnewx.pas
```

(Save the current file to the file named "JNEWX.PAS" in the tedit2 directory on the C: fixed disk.)

FIND COMMAND

```
Find text
```

The Find command does a column-dependent search forward in the file for the first line containing the specified text in the specified position in the line. The search begins with the line following the current line. If the text is found, the line containing it is made the new current line. If the text is not found, the current line remains the same.

Note that the text begins after a single blank which ends the Find command. (If the minimum abbreviation of F is used, the text on the CP-Edit command line exactly aligns with the text displayed on the screen.) Matching of the text against the lines of the file begins in the column of the file corresponding to the first tab column. Normally, this is column 1, but the tabs can be set to any column.

This command is useful for finding information in particular columns of a file.

Examples:

```
f echo off
```

(Find the next line in the file containing "echo off" beginning in the first column of the line.)

```
FIND  (*
```

(Find the next line in the file containing the beginning of a Pascal comment in column 2 of the line.)

FORWARD COMMAND

```
FOrward  [ Half ]
```

The FOrward command moves the current line forward (toward the end of the file) by either a whole screen or half a screen.

Note that this implementation is different from that in XEDIT: the Half operand is added for CP-Edit, and it is not possible to move forward by multiple screens as it is with XEDIT.

The number of lines moved is adjusted to the size of the current logical screen. This command, however, does not take into account the fact that lines of a file may occupy more than one physical line on the screen.

See also the SET PGDN command.

Examples:

```
FORWARD
```

(Moves the current line forward one screen.)

```
SET PF8 FORWARD HALF
```

(Set PF8 to its default value, to move the current line forward by half the screen height.)

GET COMMAND

```
GET    [ [d:][path]filename[.ext] ] [ firstrec [ numrec ]]
```

The GET command is used to read in lines from another file. With it you can read all or part of a file from disk. If you do not specify the number of records to be read, all the records of the file are read. If you do not specify the "firstrec" parameter, the GET begins with the first record of the file. If you do not specify the filename, the temporary file created by the PUT or PUTD command is assumed.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

The current line pointer is moved to the last line of the file read in.

Examples:

```
GET
```

(Reads in the entire contents of the temporary file and inserts it into the current file at the current line position.)

```
get template.pas
```

(Brings in all of a file named "TEMPLATE.PAS" following the current line.)

```
get a:\sampdir\sample.txt 3 10
```

(Gets 10 lines of text starting with line 3 from file "SAMPLE.TXT" in the directory SAMPDIR on the A: drive and inserts it in the current file following the current line.)

HELP COMMAND

```
Help    [d:][path][cmdname]
```

With the Help command you may display information on the use of CP-Edit commands during an editing session. If no parameter is entered, a general Help file with summary information on all commands is displayed. Or the name of a CP-Edit command may be specified for specific information on the use of a particular command. The command name may be abbreviated.

Function key 1 (F1) has the Help command assigned to it by default.

The CP-Edit Help files must be present on a disk in order for the Help command to work. If they are not, an error message is displayed when the Help command is entered. If the help files are not on the default drive, CP-Edit searches for the help files using the DOS PATH from the current environment. (See the DOS reference manual for information on the PATH command.) If the help files are not in the current DOS PATH either, you must specify the drive and path information.

Actually, the Help command merely edits the proper Help file in read-only mode, and you may browse through it using the PgUp and PgDn keys. You can change things on the screen in the Help file (although you should not), but CP-Edit will not let you FILE or SAVE the Help file, so the copy on disk will not be changed. You must QUIT from the Help file when you are done displaying it (or you may press the Esc key). The REXX version of CP-Edit is now shipped with the HELP command as a synonym for MACRO HELP. The HELP macro adds additional capabilities to the REXX version; in several of the help files there is a full-screen menu of additional help file names. You can get additional help on any of the commands in the menu just by putting the cursor over the name of the command in the menu and pressing PF1.

Examples:

```
HELP
```

(Display the general Help file.)

```
HELP A:
```

(Display the general Help file. The general Help file is on the A: disk.)

```
help ch
```

(Display the Help file for details of the use of the Change command.)

```
HELP HELP
```

(Display information on the Help command itself.)

HEXTYPE COMMAND

```
HEXType [target]
```

The HEXType command displays lines in hexadecimal. It may be useful if non-text characters are in your file. You may display them using the HEXType command, then manipulate them using hexadecimal strings with the Change command (you must SET HEX ON to use hexadecimal strings).

Lines are output beginning with the current line. The range of lines to be output are specified using a CP-Edit target. See the discussion of targets in Section “CP-Edit Targets” on page 33. Targets containing a minus sign ("-") may not be used. If the target is omitted, only the current line is output.

Examples:

```
HEXType
```

(Display the current line in hexadecimal.)

```
HEXT :25
```

(Display in hexadecimal lines starting with the current line and ending with line 24.)

INPUT COMMAND

```
Input   [line]
```

You may use the Input command in two different ways to enter text into your file. If text is specified on the command itself, the line of text is inserted into the file following the current line. The inserted line becomes the current line. If no line of text appears on the command, CP-Edit switches to Input mode.

When CP-Edit switches to Input mode, the screen display is different from the usual editing screen. The portion of the screen below the current line becomes the Input area. It is a blank area where text may be entered. Lines entered there are added to the file immediately following the current line. Only lines on which you have entered text (even a blank) are added to the file. If you skip lines (don't type in anything on them) they are not added to the file.

The size of the Input Area depends on where you have set the Current Line (CURLINE) on the screen. The default CURLINE is line 11 of the screen, which leaves an input area of 14 lines. If you want a larger input area, you can set CURLINE higher up on the screen. For example, you might

```
SET CURLINE ON 3
```

to provide the largest possible input area of 21 lines. On the other hand, the response of CP-Edit is fast enough that you might choose to use a small Input Area and press the Return key after entering each line of text. Using a smaller Input Area has the advantage that more of the old text is displayed above the CURLINE on the screen.

Pressing the Return key once in input mode causes the text entered in the input area to be added to the file and a new blank input area to be displayed. Pressing the Return key twice (without entering text) causes CP-Edit to exit from Input mode and go back to normal screen editing.

Examples:

```
I This is an inserted line.
```

(Insert the line of text in the file following the current line. Make the new line the current line.)

```
INPUT
```

(Switch to Input Mode.)

JOIN COMMAND

```
Join   [ CURSOR ]
       [ colno ]
```

The Join command may be used to join two succesive lines of the file. When no operands are supplied, the current line and the following line are combined into one line. A single blank is added to the end of the current line, then the following line is concatenated to it, and the following line is deleted.

The Join CURSOR form of the command may be used to join two lines at the cursor position. (The line following the line containing the cursor overlays the cursor line at the cursor position, and the following line is deleted.)

The Join CURSOR command is most useful if it is assigned to a Program Function key. Let us suppose that you have assigned the Join CURSOR command to Program Function key 11...

```
SET PF11 JOIN CURSOR
```

and you want to join two lines into one line. Position the cursor to the position where the line is to be join and press Shift-F1 (for PF11) to join the lines.

The JOIN colno form of the command joins the current line and the following line, with the following line overlaying the current line beginning at the colno column. The following line is deleted.

Caution: text under and to the right of the cursor (in the case of JOIN CURSOR) or under and to the right of the specified column number (in the case of JOIN colno) is overlayed and lost when the JOIN CURSOR or JOIN colno form of the command is used.

Text may not be joined to a line beyond the truncation column. The JOIN command does not lose text if the joined line extends beyond the truncation column however. The text which would not fit within the truncation column is spilled to a following line. Text beyond the truncation column of the first of the two lines to be joined does not participate in the joining.

Examples:

```
JOIN
JOIN CURSOR
JOIN 10
```

LEFT COMMAND

```
LEft    [ n ]
        [ * ]
```

The LEft command moves the screen display to the left in the current file.

The number of columns to move may be specified. If it is not, the default is 1 column. Left * means move to the left-hand side margin of the file. Left 0 means the same thing.

Examples:

```
L 20
```

(Move the display window left 20 columns.)

```
LEFT *
```

(Move the display window all the way to the left.)

LOCATE COMMAND

```
Locate   target
```

The Locate command is used to position the current line to the line in the file as specified by the target. The Locate command is a general search and positioning command. See the discussion of CP-Edit targets in Section “CP-Edit Targets” on page 33. The Locate command accepts targets which move both forward and backward.

For a few specific forms of the target, the Locate command itself may be omitted. That is, only the target need be entered. Those forms are:

1. an absolute line number:

   Example:

   ```
   :27
   ```

   (Moves to line 27 of the file.)

2. A relative displacement from the current line -- either a number alone or a number preceded by a plus sign or minus sign:

   Examples:

   ```
   +18
   ```

   (Moves forward 18 lines. Equivalent to Next 18 command.)

   ```
   -10
   ```

   (Moves backward 10 lines. Equivalent to Up 10 command.)

3. A simple string expression which uses slashes ("/") as the delimiter character, optionally preceded by a plus ("+") or minus ("-") sign and/or a not-sign (the ASCII tilde, represented here by "~"):

   ```
   /string1/

   +/string1/

   -/string1/

   ~/string1/

   +~/string1/

   -~/string1/
   ```

   The plus sign is always optional for a string expression. It specifies movement forward (toward the end) of the file. The minus sign specifies backward movement (toward the beginning) of the file.

   Examples:

   ```
   /BEGIN/
   ```

   (Search forward for the first line containing the string "BEGIN". If found, make it the current line.)

   ```
   -/PROCEDURE/
   ```

   (Search backward for the first line containing the string "PROCEDURE".)

```
-~/(*/
```

(Search backward for the first line NOT containing the beginning of a Pascal comment, "(*".)

For any other form of the target, such as a string with some character other than the slash ("/") as the delimiter, the Locate command must be used explicitly.

Examples:

```
Locate ;/**/;
```

(Looks forward from the current line for the line containing the string "/**/". If found, it becomes the current line.)

```
l -~/END/
```

(Beginning with the line before the current line, search backward for the first line NOT containing the string "END".)

LOWERCAS MACRO

```
LOWercas [ target ]
```

The LOWercas macro translates alphabetic characters in lines of the file from upper case to lower case. The LOWercas macro translates beginning with the current line over the range of lines specified by the target. If the target is omitted, one line is translated.

The LOWercas macro translates without regard to the current CASE setting. (Target search is, however, affected by the current setting of CASE Respect or Ignore.)

Example:

```
TOP
LOWERCAS *
```

(Translate the entire file to lower case.)

```
lower /end of section/
```

(Translate lines of the file beginning with the current line and continuing up to but not including the line containing the string "end of section".)

```
low 3
```

(Translate 3 lines of the file beginning with the current line.)

MACRO COMMAND

```
MACRO   [[d:][path]filename[.ext]]
                           [.MAC]
```

The MACRO command is used to execute a CP-Edit macro, either a REXX macro or a CP-Edit "native" macro (simple list of commands). See the Section “CP-Edit Macros” on page 40 for discussion of both REXX and native macros. Macro files may of course be created using CP-Edit.

It is not necessary normally to explicitly use the MACRO command to execute a macro. Normally, macros may be invoked just by entering the name of the macro on the command line (optionally followed by parameters for REXX macros). However, if you wish to execute a macro with the same name as a CP-Edit command or synonym, you must either use the MACRO command, or set a synonym for the macro.

The MACRO command assumes a file extension of .MAC if no extension is supplied. (For this reason, if you want to use a filename for a macro file which does not have an extension at all, you must append a period to the end of the filename.)

A DOS drive letter and/or path may be supplied. If they are not, CP-Edit looks for the macro file first in the current directory on the DOS default drive, then it uses the DOS PATH from the current environment to search for the macro file. This is similiar to the way DOS searches for commands to be executed. See the DOS PATH command in the DOS reference for information on how to set the DOS PATH. The PATH command must be issued before entering CP-Edit, since a PATH command issued using the CP-Edit DOS command will not affect the environment in which CP-Edit is running.

Native CP-Edit Macro files may contain the MACRO command. That is, macros may invoke other macros. Some care must be exercised in doing this. Since CP-Edit native macros contain no facility for conditional execution of commands it is possible to cause CP-Edit to fill up memory by (directly or indirectly) invoking a macro from itself. For example, if macro A invokes macro B which in turn invokes macro A, a recursion loop occurs.

CP-Edit contains code to detect an out-of-memory condition when executing a macro, so recursion should eventually terminate, but it might take a long time if you have a lot of memory to fill up.

REXX macros may not contain the MACRO command. REXX macros can invoke other macros using the REXX 'call' command. Recursion is possible with REXX macros also, but recursion in REXX macros will most likely be limited by the size of the REXX workspace.

See the sample macros provided with CP-Edit for examples of how to write REXX macros.

MERGE MACRO

```
MERGE  target1 target2 [col]
```

The MERGE macro is used to combine two sets of lines. The first set of lines is deleted and the second set is modified.

The target1 operand is used to specify the first set of lines to be merged onto the second set, starting with the current line and continuing up to but not including the line described by target1.

The target2 operand defines the beginning of the second set of lines upon which the first set of lines is to be overlayed.

The col operand defines the starting column in the second set of lines where the first set of lines will be overlayed. If this operand is not specified, the default is column 1.

This command may be used to produce multicolumn text from single column text. You might use the TF or TJ prefix commands along with the SET ZONE command to format the text into a single narrow column, then set the zone back to the default setting and use the MERGE command to move blocks of text over on the page to produce multicolumn text.

The MERGE macro is supplied as one of the sample macros with CP-Edit. Since it is written in REXX, you must have REXX installed in order to use it.

Example:

```
MERGE * :42 40
```

(Overlay the rest of the file from the current line to the end of the file on top of lines beginning with line 42, at column 40.)

MSG COMMAND

```
MSG anytext
```

The MSG command is used in REXX macros to output a message.

If MSGMODE is set to IMMEDIATE, it just writes the text of the message to the screen. In that mode, the MSG command is exactly equivalent to the REXX 'say' command.

If MSGMODE is set to DELAY, the message text is saved and displayed overlaying the editing screen upon macro completion. The message is displayed in the status area color along with any other CP-Edit error or informational message.

The MSG command may only be used in a REXX macro.

Example:

```
MSG 'This is an information message'
```

NEXT COMMAND

```
Next    [ n ]
        [ * ]
```

The Next command moves the current line forward (toward the end of the file) one or more lines. If the number of lines is omitted, 1 is assumed.

Next * is equivalent to the BOttom command. The Next command is indentical in function to the Down command.

Examples:

```
NEXT
```

(Moves the current line forward one line.)

```
N 20
```

(Moves the current line forward 20 lines.)

OS COMMAND

```
OS [,]   [ oscommand ]
```

(This command is identical to the DOS command.)

The OS command is used to execute operating system commands from CP-Edit.

After executing the OS command and before re-displaying the current editing screen, CP-Edit normally displays the message "Press any key to continue". This is to allow you to view any messages that may have been output by the OS command. When you press any key, the most recent editing screen is re-displayed. If, however, you enter a comma (",") following "OS" and preceding the command, CP-Edit does not pause.

See the DOS command for documentation of characteristics of this command that are unique to the DOS environment.

OVERLAY COMMAND

```
Overlay overlaystring
```

The Overlay command is used to overlay portions of the current line. Characters of the overlaystring replace corresponding columns of the current according to the following rules:

1. If the column in the overlaystring is blank, the corresponding column of the current line is unchanged.
2. If the column in the overlaystring contains an underline (_) character, the corresponding column of the current line becomes a blank.
3. If the column in the overlaystring contains any other non-blank character, it replaces the contents of the corresponding column of the current line.
4. If the current line is shorter than the overlaystring, it is extended to contain the overlaystring characters.

The Overlay command consists of the command (or its abbreviation) and a single blank following it. That is, the overlaystring begins immediately following the Overlay command and one blank. The command line in CP-Edit is arranged such that if the minimum abbreviation O is used for the Overlay command, the overlaystring contents exactly line up with the data lines of the file. It may be useful to move the current line near the command line when using Overlay:

```
SET CURLINE ON 3
```

Repeated overlays may be made using the REPEat command. In fact, single overlays are usually easier using the full-screen facilities of CP-Edit.

Example:

```
Overlay                    (* Version 1.10 *)
REPEat  9
```

(Overlay a version number comment on 10 lines of a Pascal source file.)

```
O *
```

(Overlay an asterisk in column 1 of the current line.)

POWERINP COMMAND

```
POWerinp
```

When this command is entered, CP-Edit switches to Power Typing mode. This mode is similar to Input mode, but you don't have to worry about line boundaries. You just enter the text without respect to lines. The screen is treated as one long stream of text, with the cursor automatically moving to the next line when you reach the end of a line. Then when you press Return, the program automatically flows lines to word boundaries and inserts the text into the file following the current line.

In Power Typing mode, the screen display is different from the usual editing screen. Power Typing allows you to use the entire screen for entering text except for the first two lines. The first line of the screen shows the name of the file and the Power Typing title. The second line of the screen shows the current line of the file. See Figure 22 on page 38 for the screen format.

In Power Typing, insertion works differently from normal editing or Input mode. Since the entire screen is treated as a stream of text, insertions wrap across multiple lines. When you insert characters, the entire stream of text shifts to the right.

CP-Edit keeps track of where you have entered text on the screen, even blanks. Only those characters you have entered on the screen are inserted into the file. If you want blanks to be in your text, you must type them in.

If you want to force a break in the text you are entering, you can use the Linend character. Whenever you enter a Linend character, CP-Edit forces a new line to begin. The Linend character itself does not appear in the resulting text. The default Linend character is the Pound Sign (#), but you can set it to any other character, or set it off entirely, using the SET LINEND command.

Pressing the Return key once in Power Typing mode causes the text entered on the screen to be added to the file and a new blank screen to be displayed. The last line entered into the file becomes the current line and is displayed on the second line of the screen. Pressing the Return key twice (without entering text) causes CP-Edit to exit from Power Typing mode and go back to normal screen editing.

The truncation column affects how text is divided into lines. If the truncation column is set to less than 71, text is divided into lines which are less than or equal to the truncation value in length. Otherwise, the lines of text are less than or equal to 71 characters long.

Power Typing mode is not affected by the Left or Right commands. Function keys cannot be used in Power Typing mode. Pressing a function key in Power Typing mode is equivalent to pressing the Return key. The scale and tab lines are not displayed in Power Typing mode.

Examples:

```
POWER

pow
```

(Begin Power Typing.)

PUT COMMAND

```
PUT target  [ [d:][path]filename[.ext] ]
```

The PUT command writes selected lines of the current file out to a file on disk. The PUT command (along with the GET command) may be used to copy lines among files. The lines may be written to a named file (if the DOS filespec is entered on the command) or to a CP-Edit temporary file.

If a named file is used, and if the file does not already exist on disk, it is created. If the file does exist, the selected lines are appended to the end of the file.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

If no filespec is used, the lines are written to a CP-Edit temporary file on the default DOS drive. If a temporary file existed before, it is overwritten.

The contents of the temporary file may be retrieved using the GET command with no filespec specified. The temporary file is erased when exiting CP-Edit.

The range of lines to be written out are specified using a CP-Edit target. See the discussion of targets in Section “CP-Edit Targets” on page 33. Targets containing a minus sign ("-") may not be used, since the PUT command only looks forward.

Examples:

```
PUT
```

(Writes out the current line to the temporary file. If a temporary file existed before this command was issued, this line overwrites it.)

```
put 10 x.xxx
```

(Writes 10 lines beginning with the current line to the file named X.XXX. If the file existed before the command was issued, the 10 lines are appended to the end of it.)

```
Put /END/ c:\pasdir\saveit.pas
```

(Writes the lines beginning with the current line up to, but not including, the first line containing "END" to the file SAVIT.PAS in the PASDIR directory on the C: drive.

```
put *
```

(Writes the rest of the file beginning with the current line to the temporary file.)

PUTD COMMAND

```
PUTD target  [ [d:][path]filename[.ext] ]
```

The PUTD command writes selected lines of the current file out to a file on disk and deletes them from the current file. The line following the last line deleted becomes the current line. The PUTD command (along with the GET command) may be used to move lines among files. The lines may be written to a named file (if the DOS filespec is entered on the command) or to a CP-Edit temporary file.

If a named file is used, and if the file does not already exist on disk, it is created. If the file does exist, the selected lines are appended to the end of the file.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

If no filespec is used, the lines are written to a CP-Edit temporary file on the default DOS drive. If a temporary file existed before, it is overwritten.

The contents of the temporary file may be retrieved using the GET command with no filespec specified. The temporary file is erased when exiting CP-Edit.

The range of lines to be written out are specified using a CP-Edit target. See the discussion of targets in Section “CP-Edit Targets” on page 33. Targets containing a minus sign ("-") may not be used, since the PUTD command only looks forward.

The PUTD command is very similar to the PUT command, except the PUTD command deletes the range of lines from the current file, while the PUT command does not.

Examples:

```
PUTD
```

(Writes out the current line to the temporary file. Deletes the line from the current file. Makes the next line of the file the current line. If a temporary file existed before this command was issued, this line overwrites it.)

```
putd 10 x.xxx
```

(Writes 10 lines beginning with the current line to the file named X.XXX and deletes them from the current file. The line following the last of the 10 lines becomes the current line. If X.XXX existed before the command was issued, the 10 lines are appended to the end of it.)

```
Putd /END/ saveit.pas
```

(Writes the lines beginning with the current line up to, but not including, the first line containing "END" to the file SAVIT.PAS and deletes them from the current file. The line containing "END" becomes the current line.)

```
putd *
```

(Writes and deletes the rest of the file beginning with the current line to the temporary file. The current line is set to the bottom of the file.)

QUERY CASE COMMAND

```
Query CASE
```

The Query CASE command is used to display the current CASE settings.

See the SET CASE command for how to change the CASE settings and their meanings.

Examples:

```
q case
```

(Query the current CASE setting. The result could be any of

```
Case Mixed Respect
CASE UPPER Respect
Case Mixed IGNORE
CASE UPPER IGNORE
```

depending on the actual current CASE setting.)

QUERY FREE COMMAND

```
Query FREE
```

The Query FREE command is used to display amount of free storage currently available to CP-Edit. The total amount of free storage and the largest block of free storage are displayed. The size of the largest free block may give some idea of the degree of storage fragmentation.

The amount of file text that can be stored is less than the total free storage because of the overhead associated with each line of the file. See Section “Invocation of CP-Edit” on page 7 for a method of estimating the amount of file text that can actually be edited.

Example:

```
query free
```

QUERY HEX COMMAND

```
Query HEX
```

The Query HEX command is used query the setting of HEX mode. See the SET HEX command for the meaning of the HEX setting.

Example:

```
query hex
```

(Queries the setting of HEX mode -- CP-Edit responds with either

```
Hex Off
```

which is the default, or

```
Hex On .)
```

QUERY IMAGE COMMAND

```
Query IMage
```

The Query IMage command is used query the setting of IMage. See the SET IMage command for the meaning of the IMage setting.

Example:

```
query image
```

(Queries the setting of IMage -- CP-Edit responds with either

```
Image On
```

which is the default, or

```
Image Off .)
```

QUERY LINEND COMMAND

```
Query LINENd
```

The Query LINENd command is used to display the Linend character used to force separate lines in Power Typing (POWerinp) mode.

Examples:

```
query linend
```

(Display the linend setting -- CP-Edit responds with

```
Linend on #
```

which is the default, or

```
Linend on $
```

where the linend character has been set to, for example, the dollar-sign character, or

```
Linend off .)
```

QUERY NBFILE COMMAND

```
Query NBFile
```

The Query NBFile command is used query the number of files in storage. See also the Query RING command.

Example:

```
query nbf
```

(Queries the number of files in storage - CP-Edit responds with a message indicating the number of files.)

QUERY PF COMMAND

```
Query PF [ nn ]
         [ *  ]
Query FK [ nn ]
         [ *  ]
```

The QUERY PF (QUERY FK) command is used to query the settings of one or all Program Function keys (Function Keys).

The settings are displayed on the screen.

If the PF key number is omitted or "*" is specified, all the settings are displayed.

Examples:

```
Q PF *
```

(Display the settings of all the Program Function keys.)

```
query FK9
```

(Query the setting of Program Function key 9.)

QUERY PGUP/PGDN COMMANDS

```
Query PGUP
Query PGDN
```

The Query PGUP/PGDN commands are used to query the settings of the PgUp and PgDn keys.

The settings are displayed on the screen.

Examples:

```
Q PGUP
Q PGDN
```

(Display the settings of both the PgUp and PgDn keys.)

QUERY RING COMMAND

```
Query RING
```

The Query RING command is used to display the names and characteristics of all the files currently in storage. The number of files in storage is also displayed. See also the Query NBFile command.

Example:

```
query ring
```

(CP-Edit displays the number of files, and the name, the Trunc, the Size, and current line number for each file currently in storage.)

Example output:

```
2 files in storage
abc.xyz                    Trunc=255 Size=25 Line=10
test.txt                   Trunc=40 Size=550 Line=40
```

QUERY STREAM COMMAND

```
Query STREAM
```

The Query STREAM command is used query the setting of STREAM mode. See the SET STREAM command for the meaning of the STREAM setting.

Example:

```
query stream
```

(Queries the setting of STREAM mode -- CP-Edit responds with either

```
Stream Off
```

which is the default, or

```
Stream On .)
```

QUERY TABS COMMAND

```
Query TABS
```

The QUERY TABS command is used to query the settings of the CP-Edit tab stops.

The tab stops may be set using the SET TABS command.

Example:

```
Q TABS
```

(Display the settings of all the tab stops.)

QUERY TRUNC COMMAND

```
Query TRunc
```

The QUERY TRunc command is used to query the setting of the CP-Edit truncation column.

The truncation column may be set using the SET TRunc command.

The truncation column is always displayed on the top line of the CP-Edit screen.

Example:

```
Q TRUNC
```

(Display the settings of the truncation column.)

QUERY VERIFY COMMAND

```
Query Verify
```

The QUERY Verify command is used to query the settings of the CP-Edit starting and ending verify columns.

The starting and ending verify columns may be set using the SET Verify command.

Example:

```
Q Verify
```

(Display the settings of the starting and ending verify columns.)

QUERY VERSHIFT COMMAND

```
Query VERShift
```

The QUERY VERShift command is used to query the shift in the CP-Edit verify columns produced by the LEft and RIght commands.

Example:

```
Q VERSHIFT
```

(Display the verify shift.)

QUERY VERSION COMMAND

```
Query VERSIon
```

The Query VERSIon command is used query the current CPEDIT version. The output message gives the name of the .EXE file, the version number, and, if a virtual version, the paging device (disk file name or "EMS").

Example:

```
query version
```

(Queries the current CPEDIT version.)

QUERY WRAP COMMAND

```
Query WRap
```

The Query WRap command is used query the setting of WRAP. See the SET WRap command for the meaning of the WRAP setting.

Example:

```
query wrap
```

(Queries the setting of WRap -- CP-Edit responds with either

```
Wrap On
```

which is the default, or

```
Wrap Off .)
```

QUERY ZONE COMMAND

```
Query Zone
```

The QUERY Zone command is used to query the settings of the CP-Edit left and right zone columns.

The left and right zone columns may be set using the SET Zone command.

Example:

```
Q Zone
```

(Display the settings of the left and right zone columns.)

QQUIT COMMAND

```
QQuit [n]
```

The QQuit command unconditionally exits from the current file. If the current file is the only one in storage, CP-Edit exits to DOS.

The "n" operand may be used to set the DOS ERRORLEVEL if the file is the last one in storage and CP-Edit exits to DOS.

The QQuit command is an "unprotected" quit. That is, it exits even if changes made since the file was last saved would be lost. See also the QUIT ("protected" quit) command. Normally you should use the QUIT command to exit from a file to make sure changes are not lost. You should only use the QQuit command when you are sure you do not want to save changes you have made to the current file.

Examples:

```
QQ
```

(Exits the current file. If the current file is the only file in storage, causes CP-Edit to exit to DOS.)

```
QQUIT 16
```

(Exits the current file. If the current file is the last one in storage, exits to DOS and sets the DOS ERRORLEVEL to 16.)

QUIT COMMAND

```
QUIT [n]
```

The QUIT command exits from the current file if no changes have been made to it since editing of it began, or since the last time a SAVE command was executed for it. If the current file is the only one in storage, CP-Edit exits to DOS.

The "n" operand may be used to set the DOS ERRORLEVEL if the file is the last one in storage and CP-Edit exits to DOS.

The QUIT command exits only if no changes would be lost. It is thus a "protected" quit command. See also the QQuit ("unprotected" quit) command.

Example:

```
QUIT
```

(Exits the current file. If the current file is the only file in storage, causes CP-Edit to exit to DOS.)

```
QUIT 1
```

(Exits the current file. If the current file is the last one in storage, exits to DOS and sets the DOS ERRORLEVEL to 1.)

RECOVER COMMAND

```
RECover    [ n ]
           [ * ]
```

The RECover command restores lines which were deleted. The lines are added following the current line. If no operand is specified, one line is restored.

Currently, CP-Edit saves a maximum of 10 lines.

Lines may have been deleted in any of several ways:

- The DELETE command
- The D and DD prefix commands
- The PUTD command
- The TF and TJ prefix commands delete lines before reinserting them in formatted form
- Some macros may delete lines (the MERGE macro, for example).

Examples:

```
RECOVER
```

(Put back the last line deleted, following the current line.)

```
Recover 3
```

(Recover the 3 most recently deleted lines.)

```
Recover *
```

(Recover all the deleted lines which have been saved by the program. Currently, the most that can be recovered is 10 lines.)

REPEAT COMMAND

```
REPEat [ target ]
```

The REPEat command repeats the previous command beginning with the line following the current line over the range of lines specified by the target. If the target is omitted, one repetition occurs.

The REPEat command increments the current line pointer, then reexecutes the previous command. If the target line has not been reached, the current line pointer is incremented and the command is executed again.

The REPEat command is most useful when used with commands which act only on the current line, such as Overlay. Repeating commands which themselves increment the current line pointer may not give the desired results.

Example:

```
Overlay *
REPEat /SET PF/
```

(Overlay records in the file beginning with the current line and continuing up to but not including the first line containing the characters "SET PF".)

```
SET HEX ON
HEXTYPE
REPEAT ~/X'1B'/
```

(Type out in hexadecimal the contents of lines starting at the current line up to the first line not containing an ASCII Escape character.)

RESET COMMAND

```
RESet
```

The RESet command clears any pending prefix commands.

Example:

```
RESET
```

RIGHT COMMAND

```
RIght    [ n ]
```

The RIght command moves the screen display to the right in the current file.

The number of columns to move may be specified. If it is not, the default is 1 column.

Examples:

```
R 20
```

(Move the display window right 20 columns.)

SAVE COMMAND

```
SAVe   [ [d:][path]filename[.ext] ]
```

The SAVE command saves the current file on disk.

A DOS filespec may be specified on the SAVE command. If it is, the file is saved on disk with that name. If it is not, the current name of the file as displayed on the ID line of the screen is used.

An equal sign ("=") may be used in place of the normal DOS drive and path to mean "the drive and path of the current file".

Examples:

```
SAVE
```

(Saves the current file on disk under its current name.)

```
save b:xyz.txt
```

(Saves the current file on the B: drive with filename XYZ.TXT. If an old file with name XYZ.TXT exists on the B: drive, it is replaced.)

SET ALT COMMAND

```
SET ALT n [p]
```

The SET ALT command is used to set the number of changes which CP-Edit believes have been made to the current file since the last AUTOSAVE and, optionally, since the last SAVE command.

The "n" operand is a integer (0-32767) which represents the number of changes since the last automatic save of the file by the AUTOSAVE facility. The optional "p" operand is an integer representing the number of changes since the last SAVE of the file (or since editing of the file began, if no SAVE has been done during this editing sessiong).

If the alteration count is set to a value less than or equal to the AUTOSAVE value which has been specified, an immediate AUTOSAVE will be done.

The number of changes which have been made to the file since the last AUTOSAVE or SAVE of the file is displayed on the i.d. line at the top of the screen: See also the EXTRACT /ALT/ command, which allows accessing both alteration counts from a REXX macro.

Using the EXTRACT /ALT/ and the SET ALT commands in a macro could allow a macro to cause the AUTOSAVE facility to ignore any changes to the file made in the macro, or condense the number from many to one.

Examples:

```
set alt 10
```

(Set the alteration count since the last AUTOSAVE to 10.)

```
SET ALT 20 100
```

(Set the number of alterations since the last AUTOSAVE to 20. Set the number of alterations since the last SAVE to 100.)

SET AUTOSAVE COMMAND

```
SET AUtosave n
            OFF
```

The SET AUtosave command is used to turn automatic saving of the current file on or off and specify the number of changes which must be made to the file before the file is saved. Each file in storage may have a unique setting of AUtosave.

The file is saved to a temporary file of the form nnnnn.@SV in the same directory as the file being edited. CP-Edit selects the next unused number beginning with 10001 for the filename of the file in order to generate a unique filename and not destroy any other such temporary files. When the file is SAVEd or FILEd successfully, the temporary file is erased. If a QUIT or QQUIT is done, the temporary file is *not* erased. Obviously, if some disaster should befall your editing session, the file would not be erased either. This allows you to recover from some mistakes in editing a file, or from a catastrophic crash of your Personal Computer (such as one caused by a power outage).

See also the SET ALT command and the EXTRACT /ALT/ command. The number of changes which have been made to the file since the last SAVE or autosave of the file is displayed on the i.d. line at the top of the screen:

```
Alt=nnnnn
```

Examples:

```
set au 20
```

(Autosave the file after every 20 changes.)

```
SET AUTOSAVE OFF
```

(Sets AUtosave back to the default of OFF.)

SET CASE COMMAND

```
SET CASE [ Mixed | Upper]  [ Respect | Ignore ]
```

The SET CASE command is used to change the way input is treated by CP-Edit and the way string searches are treated.

The Mixed/Upper setting affects CP-Edit input. If CASE is set Mixed, input to the file and command input are entered as typed, in upper and lower case. If CASE is set Upper, alphabetic characters input to the file or entered in CP-Edit commands are translated to upper case. If case is set to Upper, any change made to a line in full-screen editing by overtyping any character of the line will result in the entire line being translated to upper case.

The Respect/Ignore setting affects the way string targets are treated. If CASE is set to Respect, string targets must match file text exactly. If CASE is set to Ignore, any combination of upper and lower case characters in the string target will match any combination of upper and lower case characters in the file text. For example, with CASE set to Respect,

```
l /Farkle/
```

will only match the string "Farkle" in the text of the file. But with CASE set to Ignore, the same Locate command would match "FARKLE", "fArklE", or "farkle" as well. In fact, here are some other example locate commands that would match any of these text strings:

```
l /farkle/
L /FARKLE/
l /fARKLe/
```

The Respect or Ignore setting affects the operation of the SORT command. If Respect is the current setting, keys sort in normal ASCII collating sequence, with lower case letters sorting higher than upper case letters. If Ignore is the current setting, upper case and lower case letters sort identically (lower case is "folded" into upper case).

The default Case setting is Mixed Respect. The setting of CASE can be displayed using the Query CASE command.

For making mass changes of lines from lower case to upper, the UPPercas command may be used. For making mass changes of lines from upper case to lower, the LOWercas command may be used.

Examples:

```
set case mixed ignore
```

(Set CASE to Mixed (the default) and target string searches to Ignore.)

```
SET CASE U
```

(Set CASE to Upper -- lines input to the file are translated to upper case.)

```
SET CASE U I
```

(Set CASE to Upper and target string searches to Ignore.)

```
set case m r
```

(Set CASE back to Mixed and Respect (the defaults).)

SET CMDLINE COMMAND

```
SET CMDline Top
            Bottom
```

The SET CMDline command is used to set the command line to the top or bottom of the screen.

Examples:

```
set cmdline bottom
set cmdline top
```

(Set the command line to the bottom of the screen. Set the command line back to the default position at the top of the screen.)

SET COLOR COMMAND

```
SET COLOR area  [ color ] [ exthi ] [ High ]
          [ * ]                    [ Nohigh ]
```

The SET COLOR command is used to set the color of various areas of the CP-Edit screen. The following areas may be set:

```
Arrow
Cmdline
CUrline
Filearea
Idline
Msgline
Pending
PFkey
PRefix
Scale
SHadow
STatarea
Tabline
TOfeof
BACKgrnd
```

As usual, the minimum abbreviation of the area name is shown in capital letters above. If "*" is entered for the screen area, all the above areas of the screen are changed except the background. The background color setting is ignored for a monochrome monitor.

The following colors may be used:

```
BLAck
BLUe
Green
Cyan
Red
Magenta
Yellow
White
Underlin    (Supported for DOS character-mode version only)
```

The minimum abbreviation of the color name is shown in upper case letters above. (Underline is actually the same color as Blue: if either is specified, underlining occurs on a monochrome display, and the color blue is produced on a color display.) If the color is omitted, but the highlighting parameter is entered, the highlighting is changed but the color is not. (This is useful for monochrome displays.) If the highlighting parameter is omitted, the highlighting is not changed.

The following extended highlighting (exthi) attributes may be specified:

```
REVvideo
BLInk
NONe
```

If the extended highlighting is omitted, the extended highlighting setting is not changed. Setting a color with the NONe attribute resets any extended highlighting that may have been set previously.

Highlighting and extended highlighting are ignored if the screen background (BACKgrnd) color is set (except for the NONE extended highlighting attribute). To set the background color back to black, enter

```
SET COLOR BACKGRND NONE
```

Note that some colors may look better highlighted than not highlighted on certain displays. Also some color monitors may not recognize the intensity (highlight) bit. Some experimentation with the various colors may be required to produce the best color combinations.

On monochrome displays connected to color adapter cards, the color White would normally be used for all the areas of the screen. The following commands, which produce the same colors as are used with the monochrome adapter card, might be placed in the PROFILE.MAC file so that they would be executed automatically:

```
set color * white nohi
set color arrow white high
set color curline white high
set color idline white high
set color tabline white high
set color scale white high
set color pending white high
set color msgline white high
```

Examples:

```
SET COLOR FILEAREA BLUE
```

(Set the color of the file area to blue.)

```
set color cmdline nohigh
```

(Change the highlighting of the cmdline to no highlighting. Preserve the previous color.)

```
SET COLOR PENDING MAGENTA HIGH
```

(Set the color of pending prefix commands to magenta, highlighted.)

```
set color statarea rev
```

(Set the color of the status area at the bottom of the screen to reverse.)

```
set color msgline red blink hi
```

(Set the color of error messages to red, blinking, and highlighted. This is guaranteed to get your attention.)

SET CURLINE COMMAND

```
SET CURLine [ON]  [ n ]
                  [M[n|+n|-n]]
                  [n|+n|-n]
```

The SET CURLine command is used to set the position of the current line of the file on the screen. The position may range between lines 3 and 23. Each file and each logical screen may have a unique setting of CURLine.

The line where the current line is to be set may be specified as an absolute line number; or as a number preceded by a plus sign or minus sign, signifying a offset from the top of the screen (+) or the bottom (-); or "M" may be specified to mean the Middle of the screen, with or without an offset. The offset may be preceded by by a plus or minus sign for a positive or negative offset from the middle of the screen. If no sign is specified, the offset is assumed to be positive.

The setting for CURLine controls the size of the input area in input mode. The size of the input area is maxline - CURLine lines (where maxline is the last line of the logical screen). If CURLine is set to maxline the input area contains no lines and input mode becomes very difficult to use.

Examples:

```
set curline 3
```

(Sets the current line position to line 3 of the screen.)

```
SET CURL ON M-1
```

(Sets CURLine back to the default position one line above the middle line.)

SET DSPF COMMAND

```
SET DSPF   ON   [ n ]
           OFF  [ 1 ]
```

The SET DSPF command is used to set the Program Function Key display in the status area at the bottom of the screen ON or OFF. It may also be used to change the range of Program Function keys displayed.

The number (n) is the starting PF key to be displayed. 10 key settings are displayed. The number n must be between 1 and 40. If a number greater than 31 is specified, the display wraps around to the PF keys starting with 1. (For example if SET DSPF ON 35 was specified, keys 35 through 40, then 1 through 4 would be displayed.)

The default setting is ON 1.

You can set up function keys to cycle the display through all 40 of the PF keys by assigning a SET DSPF command to one of each set of 10. For example:

```
set pf10 set dspf on 11
set pf20 set dspf on 21
set pf30 set dspf on 31
set pf40 set dspf on  1
```

Examples:

```
SET DSPF ON 1
```

(Sets PF key display on, with the first 10 keys displayed.)

```
SET DSPF OFF
```

(Sets PF key display off. CP-Edit status line is displayed.)

```
SET DSPF ON
```

(Sets PF key display on. Starting key is unchanged from when the display was last on.)

SET ENTER COMMAND

```
SET ENTer   BEFORE      string
            AFTER
            IGNORE
```

The SET ENTer command specifies what action occurs when the Enter (Return) key is pressed. The string may be any CP-Edit command. If the string is null, no action occurs.

If BEFORE is specified, the command is executed before the contents of the command line (or exclusively, if the command line is empty).

If AFTER is specified, the command is executed after the contents of the command line (or exclusively, if the command line is empty).

If IGNORE is specified, the command is executed only if the command line is empty.

'SET ENTER IGNORE CURSOR CMDLINE 1' with various priorities has been tested, as have a few other commands with 'BEFORE', 'AFTER', and 'IGNORE', but it is possible that there are some commands which could at least be very disruptive when used with SET ENTER, particularly with the 'BEFORE' and 'AFTER' forms. Use SET ENTER with care.

Examples:

```
SET ENTER IGNORE CURSOR CMDLINE PRIORITY 30
```

(Set the Enter key action to the XEDIT default.)

```
SET ENTER IGNORE
```

(Set the Enter key action to the CP-Edit default.)

```
SET ENTER AFTER QUERY FREE
```

(Set the Enter key to display the amount of free storage any time it is pressed.)

SET FNAME COMMAND

```
SET FName  [d:][path]filename[.ext]
```

The SET FName command is used to change the filename of the current file in storage. The filename must be supplied.

Examples:

```
SET FNAME ABC.XYZ
```

(Sets the filename of the current file to "ABC.XYZ".)

```
set fn readme
```

(Changes the filename of the current file to "README".)

SET FONT COMMAND

```
SET FONT  font-name
```

The SET FONT command is used to change the current font. The font-name must be supplied.

Legal fonts in the DOS BGI version currently are "default", "bitmap" (same as "default"), "goth.chr", "sans.chr", "trip.chr", and "litt.chr". The font files must either be in the current directory or an environment variable must be set to point to the directory containing the font files (and the BGI driver files). Add a SET command to your autoexec.bat file to set the environment variable to the proper directory:

```
set cbgipath=d:\borlandc\bgi
```

Legal fonts for the AIX X Windows version are any of the X installed fonts.

Legal values for the DOS character-mode version are "CO40" and "CO80".

Examples:

```
SET FONT trip.chr
```

(DOS BGI version: Sets the font to the Borland triplex font.)

```
set font 8x13
```

(AIX X Windows version: Sets the current font to "8x13".)

SET HEX COMMAND

```
SET HEX OFF
        ON
```

The SET HEX command is used to turn hexadecimal mode on or off.

If HEX is set ON, string targets in hexadecimal are allowed. A hexadecimal target string has the form

```
X'nn...nn'
```

where nn are pairs of hexadecimal characters (0-9, A-F). When HEX is ON and CP-Edit finds a target or change string of this form, it converts it into its internal representation. This allows the representation of arbitrary values in the file which could not be entered through the keyboard otherwise.

If HEX is set OFF, strings in the form of hexadecimal strings are treated literally (not converted into their internal representation).

Some hexadecimal values (such as X'0D' -- carriage return -- and X'1A' -- DOS end-of-file character) can cause strange (and usually unpleasant) things to happen if they get into your file. Be sure you know what you are doing when you use this facility.

The HEXType command may be used to look at a line in hexadecimal form.

Examples:

```
set hex on
change /x'0c'// * *
change /x'0A'// * *
```

(Set HEX to ON -- hex change and target strings are allowed; then delete all ASCII Form Feed and Line Feed characters from the file starting with the current line and continuing through the end of the file.)

```
SET HEX OFF
```

(Set HEX OFF -- hexadecimal change and target strings are not interpreted.)

SET IMAGE COMMAND

```
SET IMage OFF
          ON
```

The SET IMage command is used to change the way files are written on disk by CP-Edit. If IMage is set ON, files are written on disk just the way they appear on the screen while being edited. This is the default setting for CP-Edit. If IMage is set OFF, files are written on disk using ASCII tab characters to replace strings of blanks where possible. Each file in storage may have its own setting of IMage.

CP-Edit can read files stored with or without tab characters. The SET IMage command allows you to set whether CP-Edit writes files with tab characters (OFF) or not (ON). Files stored with tab characters may occupy less space on disk, and are compatible with most DOS commands and many other text editors and language processors which run under DOS.

Examples:

```
set image on
```

(Sets IMage to ON -- the current file will be written with blanks just as it appears on the screen.)

```
SET IM OFF
```

(Sets IMage OFF -- the current file will be written with tab characters substituted for strings of blanks where possible.)

SET KEYBOARD COMMAND

```
SET KEYBOARD ENHANCED
             PC1
             ORIGINAL
```

The SET KEYBOARD command is used to force the CP-Edit keyboard code to use only the original PC/1 (PC1 or ORIGINAL) BIOS calls or to attempt to use the 101/102-key (ENHANCED) BIOS calls when reading keystrokes. The default is ENHANCED.

It should normally not be necessary to use this command. The CP-Edit keyboard code should normally detect when the enhanced keyboard BIOS calls are not present and use the old ones, but on some older clone BIOSes and with some programs that replace the keyboard BIOS interrupt handler, the attempt to detect which BIOS calls are present causes problems. Typical symptoms are continuous beeping from the program or other hangs.

Examples:

```
set keyboard PC1
set keyboard enhanced
```

(Set keyboard handling to the normal 101/102 key keyboard BIOS calls or restricts keyboard calls to only the original BIOS calls.)

SET LINEND COMMAND

```
SET LINENd   ON  [ c ]
             OFF
```

The SET LINENd command is used to set the Linend character used to force separate lines in Power Typing (POWerinp) mode. In CP-Edit, it affects only Power Typing.

The default setting is ON #. If you do not specify the character when you set LINENd ON, it is set to whatever it was when it was turned off. If you have never set it, the default used is the pound-sign ("#") character.

Examples:

```
set Linend on $
```

(Sets Linend on and Linend character to dollar-sign -- "$".)

```
SET LINEN off
```

(Sets Linend off -- no Linend character.)

```
SET LINEND ON
```

(Sets Linend on and Linend character to the default -- "#".)

SET MSGMODE COMMAND

```
SET MSGMode ON   [Delay]
            OFF  [Immediate]
```

The SET MSGMODE command is used to set error and information message output on or off and to change where error and informational messages are displayed from a macro.

This command is most useful in a macro. The OFF operand is used to get rid of messages which are incidental to the operation of the macro, but which might be distracting or misleading to the user of the macro. If Immediate is specified, error and information messages are displayed as they are generated on the macro (scrolling) screen during macro execution. If Delay is specified, error and information messages generated during macro execution are saved until execution of the macro is complete, then they are displayed overlaying the bottom lines of the editing screen, the same as error messages and information messages generated during normal full-screen editing.

SET MSGMODE does not affect REXX trace output or output from the REXX "say" statement.

If you SET MSGMODE in a macro, you normally would want to preserve the current setting (using the EXTRACT /MSGMODE/ command) and restore it upon exit from the macro.

The SET MSGMODE command is global in scope; that is, it affects all files being edited.

The default value for MSGMODE is ON DELAY.

Examples:

```
set msgmode on
msgm off
```

```
set msgmode on delay
```

(Delay messages until after macro execution is complete. Display them overlaying the editing screen.)

SET NUMBER COMMAND

```
SET NUMber ON
           OFF
```

The SET NUMber command is used to the prefix area to numbers or to equal-signs ("="). If NUMber is set ON, the prefix area contains the line numbers of the file lines. If NUMber is set OFF, the prefix area contains "=====" for each line.

The effect of this command is purely esthetic. It does not change the way CP-Edit operates.

Examples:

```
set number on
SET NUM OFF
```

(Set NUMber to ON or OFF.)

SET PALETTE COMMAND

```
SET PALette [ p0 p1 p2 .. p15 ]
```

The SET PALETTE command currently is implemented only in the OS/2 flavor of CP-Edit.

The SET PALETTE command is used to load values into the EGA/VGA palette registers. This allows complete flexibility in setting colors in CP-Edit. The palette values p0-p15 are expressed as decimal numbers between 0 and 63.

Examples:

```
set palette 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

(Set the palette registers to the CGA default values.)

```
set palette 0 9 2 3 36 5 54 31 56 15 18 11 44 61 55 63
```

(Set the palette registers to the CP-Edit default values.)

SET PAUSE COMMAND

```
SET PAUSE ON
          OFF
```

The SET PAUSE command is used to set pausing for screen output in REXX macros. If PAUSE is set ON, CP-Edit pauses after every screen full of output and requests that you press the Space Bar to continue. It also pauses at the end of macro execution. If PAUSE is set OFF, output from a macro may disappear from the screen before it can be read.

The default is PAUSE OFF. This is similar to the behavior of XEDIT, and is useful if you don't really want to see the output from a macro.

Examples:

```
set pause on
set pause off
```

(Set PAUSE to ON or OFF.)

SET PF COMMAND

```
SET PFnn    command
SET FKnn    command
  or,
SET PF nn   command
SET FK nn   command
```

The SET PF (SET FK) command may be used to set the specified Program Function key (Function Key) to the desired command. CP-Edit supports 40 PF keys, so the value of nn may range between 1 an 40.

The command may be any legitimate CP-Edit command, or a CP-Edit comment, or the special commands NEWLINE and ERASEEOL. (See below for a discussion of these special commands.) The command is not checked when it is assigned to the Function key, however. It is not checked until the Function key is pressed and CP-Edit attempts to execute the command. If the command is in error, an appropriate error message will be issued when the Function key is pressed. The Function key should then be set to a correct CP-Edit command using the SET PF command.

The command may be omitted. If it is, the Program Function key is set to a CP-Edit comment ("*").

The settings of one or all Program Function keys may be queried using the QUERY PF command.

See Section “Using Function Keys” on page 39 for a discussion of the use of Program Function keys. See Appendix B, “Default Settings” on page 219 for the default settings.

Examples:

```
SET PF13 SPLIT CURSOR
set pf 11 top
set fk12 bottom
```

(Set Program Function key 13 to the SPLIT CURSOR command. Set Program Function key 11 to the TOP command. Set Program Function key 12 to the BOttom command.)

SET PGUP/PGDN COMMANDS

```
SET PGUP Half
         Page

SET PGDN Half
         Page
```

The SET PGUP and SET PGDN commands may be used to set the PgUp and PgDn scrolling amounts to either a full page or a half page.

The default value for both is Half.

The keys are actually set to BACKWARD [HALF] and FORWARD [HALF] commands respectively.

The settings of the PgUp and PgDn keys may be queried using the QUERY PGUP/PGUP commands.

Examples:

```
SET PGUP P
SET PGDN PAGE
```

(Set both page keys to full-page mode.)

```
set pgup half
set pgdn h
```

(Set both page keys to half-page mode.)

SET POINT COMMAND

```
SET POINT  .symbol [ OFF ]
```

The SET POINT command sets a symbolic name for the current line, or turns off the symbolic name.

The symbolic name may be any eight characters. Case is significant; that is, the following are all different names:

```
.start
.Start
.START
```

A maximum of 5 names per file are supported currently.

More than one name may be associated with a line. A name remains associated with the line while the file is being edited, and as long as the line is not deleted or moved, or until the SET POINT .xxxxxxxx OFF command is issued for it.

The symbol may be used as the operand of any command accepting a target. See the section “CP-Edit Targets” on page 33.

Symbols may also be set using the .xxxx prefix command.

Examples:

```
SET POINT .START
set point .end
set point .1111 off
```

SET PREFIX COMMAND

```
SET PREFIX  ON LEFT
            ON RIGHT
            OFF

SET PREFIX  SYNONYM newname oldname
```

The SET PREFIX command may be used to put the prefix area on the left or right side of the screen or to turn it off altogether.

The SET PREFIX SYNONYM form may be used to set a synonym for a prefix command. There are a few constraints on this command:

1. There are only 10 spare entries for synonyms currently.
2. A prefix synonym cannot be redefined, only overridden. So one of the spare entries is used up for every SET PREFIX SYNONYM command, except that exact duplicate synonyms are ignored.
3. Prefix commands are always effectively upper case. You cannot define a lower case synonym.
4. You should not define a synonym 'ZZZZ'. This name is reserved for the spare entries.

A synonym of a previously-defined synonym is permitted.

Examples:

```
set prefix on right
```

(Set PREFIX on the right side of the screen.)

```
SET PREFIX OFF
```

(Set PREFIX off. Use the entire width of the screen for text.)

```
SET PREFIX ON LEFT
```

(Set PREFIX to the default values.)

```
set cmdline bottom
set prefix on left
set number off
```

(Set screen configuration to resemble the XEDIT defaults.)

```
set prefix synonym a f
```

(Set a synonym of A for After for the F for Following prefix command. This overrides the use of A for Add.)

```
SET PREFIX SYNONYM DEL D
```

(Add a synonym of DEL for the D prefix command.)

SET READONLY COMMAND

```
SET READONLY ON
             OFF
```

Allows changing the Read-only status of a file. Read-only mode is set for HELP files and for any file which is not read in completely when loading the file (usually because it filled up memory). Be cautious about using this command, since it could allow unintentionally modifing a HELP file or storing a file which was truncated because it was not read in completely.

Examples:

```
set readonly on
set readonly off
```

(Set READONLY to ON or OFF.)

SET SCALE COMMAND

```
SET SCALe ON  [ n ]
              [M[n|+n|-n]]
              [n|+n|-n]

          OFF
```

The SET SCALe command is used to turn on, turn off, and set the position of the scale line on the screen. The line number may range between 3 and 23. It may not be the same as the CURLine, and it may not be the same as the TABLine if it is turned on. The default is OFF, line 10.

The line where the scale line is to be set may be specified as an absolute line number; or as a number preceded by a plus sign or minus sign, signifying a offset from the top of the screen (+) or the bottom (-); or "M" may be specified to mean the Middle of the screen, with or without an offset. The offset may be preceded by by a plus or minus sign for a positive or negative offset from the middle of the screen. If no sign is specified, the offset is assumed to be positive.

If the line number is omitted from a SET SCALe ON command, the last setting, if any is used. If the line number has never been set, the default of 10 is used.

Examples:

```
set scale on 3
```

(Set the scale display on line 3 of the screen.)

```
SET SCALE OFF
```

(Turn off the scale display.)

See Figure 32 on page 163 for a sample screen with the scale on.

```
  SAMPLE.PAS              TRUNC=255 SIZE=11 LINE=0
===>









      <...+....1....+....2....+....3....+....4....+....5....+...
00000 * * * TOP OF FILE * * *
00001 PROCEDURE SAMPLE (INPUT,OUTPUT);
00002   (* This is a sample Pascal program *)
00003
00004 VAR
00005   greeting : LSTRING(255);
00006 BEGIN
00007   greeting := 'Hello there!';
00008   Writeln (greeting);
00009
00010 END (* of the SAMPLE program *) .
00011 * * * END OF FILE * * *

F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 32. CP-Edit Screen with Scale On

SET SCREEN COMMAND

```
SET SCReen n  [ Horizontal | Vertical ]

           Size   s1 [s2 [s3... [sn]]]

           Width  s1 [s2 [s3... [sn]]]

           Define h1 w1 sl1 sc1 [h2 w2 sl2 sc2]...
```

The SET SCReen command is used to define the logical screens used for editing. You may split the physical screen into multiple logical screens, which may hold different files or different views of the same file.

There are a number of different ways in which you can specify the logical screens:

- n -- specifies the number of equal-sized logical screens into which the physical screen is to be split. The screen may be split either Horizontally (the default) or Vertically.
- Size -- the screen is split horizontally into logical screens containing the specified numbers of lines.
- Width -- the screen is split vertically into logical screens containing the specified numbers of column.
- Define -- the screen is split into logical screens having the specified number of lines and columns, whose upper left corners are at the specified position on the physical screen. These quantities are specified like this:
  - hn -- the height of the logical screen in lines
  - wn -- the width of the logical screen in columns
  - sln -- the start line (uppermost line) of the logical screen
  - scn -- the start column (leftmost column) of the logical screen

  Any number of screens can be specified, provided that the restrictions below are met.

  Note that it is possible to specify screens in some other order than left-to-right, top-to-bottom with Define. If you do this, the screens will be logically ordered in the way they are entered on the command. This affects the order of processing of different views of the same file, and also affects the operation of SOS TABCMDF and SOS TABCMDB.

The default setting is SET SCREEN 1.

Restrictions: Screens must be at least 5 lines high. Screens must be at least 20 columns wide. Screens specified using the Define option may not overlap, or extend beyond the boundaries of the physical screen.

There are no restrictions on the number of screens defined, except that each screen takes additional storage (Each screen requires approximately 800 bytes of storage). If insuffient storage is available for the number of screens requested, a single screen (the default) is displayed. Since a CP-Edit command may be no longer than 255 characters, the practical upper limit is about 24 screens. (Issuing a command longer than the command line requires that the command be put into a macro.)

CP-Edit understands only two physical screen widths: 80 columns and 40 columns. If the display is in a 40-column video mode, it assumes that the display is 40 columns wide; otherwise it assumes that the display is 80 columns wide. (Other display widths -- i.e., wider than 80 columns -- are certainly possible, but currently the program has no way of discovering that it has other widths available.)

CP-Edit understands display heights in the range 25 lines to 50 lines. If the indication of height that it receives from the display is outside this range, it assumes an error of some sort and reverts to a 25-line display. Displays other than 25 lines high are possible only with EGA/VGA displays or displays that appear to be an EGA or VGA.

If multiple logical screens are specified and there are multiple files in storage, as many of the files as possible will be displayed. Generally, the files will be displayed starting at the upper left of the physical screen with the file which was current when the SET SCREEN command was issued, and continuing displaying files in the order in which they appear in the ring. (Note that this is slightly different from XEDIT, which takes into account the previous relations of files to screens.)

Of course you can immediately change which file is displayed in any logical screen by issuing an Edit command from that screen.

When the width of a logical screen is less than the total width of the physical screen, lines are displayed only up to the width of the logical screen; lines do not wrap; changing the SET VERIFY setting will not affect this.

CP-Edit attempts to display the CURLINE, SCALE, and TABLINE in the same relationship to the logical screen as it was before the SET SCREEN command was issued. However, if those quantities are specified as absolute numbers, and if the resulting setting would fall outside the logical screen, then the default value is used. This means that if you intend to use other than the default values for the position of CURLINE, the SCALE, and the TABLINE, then specifying them relative to the top, bottom, or middle of the screen (rather than absolute) would be a good idea.

When multiple views of the same file are displayed in multiple logical screens, changes to the file are processed in the following order:

1. Changes to lines made on the screen: changes are made (normally) starting in the upper left logical screen, continuing to the right and down. If the same line is displayed and also changed in multiple views, the change made in the last logical screen will be the only one which takes effect.
2. Prefix commands: all prefix commands for a single file are processed at once, with the normal CP-Edit restrictions on prefix commands: no overlapping block commands, and only one pending prefix command or block pair allowed per file. Prefix commands are processed in order from top to bottom of the file. Note that it is possible to issue part of a block prefix command from one view and the rest of it from another, allowing, for example, moving or copying lines from one view to another.
3. PF key command: only a single PF key command can be issued at once, and the PF key is associated with the file in the logical screen which contains the cursor at the time the PF key is pressed.
4. Command line commands: since each logical screen has its own command line, multiple commands may be processed. Each is associated with the logical screen from which it is issued, and with the file in that logical screen.

The normal order of processing of logical screens is left-to-right, top-to-bottom. If you have used the Define operand, the order of processing is the order in which the logical screens were defined in the command (operands are not sorted before processing).

Note that the XEDIT restriction that vertical splits cover the entire screen width does not apply in CP-Edit.

Examples:

```
SET SCREEN 2
```

(Split the screen horizontally into two equal-sized logical screens. The logical screens between them fill the entire height of the physical screen, and extend across the entire width of the physical screen.)

```
screen 2 v
```

(Split the screen vertically into 2 equal-sized logical screens. Each spans the full height of the physical screen and each is 40 columns wide -- assuming an 80-character wide display.)

```
SCR 4 V
```

(Split the screen vertically into four equal-sized logical screens. Assumes an 80-column display.)

```
Screen size 10 15
```

(Split the screen horizontally into logical screens 10 and 15 lines high. Assumes a 25-line display.)

```
SET SCR define  20 40 1 1    20 40 1 41    23 80 21 1
```

(Split the screen into two vertical logical screens in the upper part of the screen and a single logical screen covering the full width of the lower part of the screen. Assumes a 43-line display.)

```
SET SCR define 20 40 1 1    25 40 1 41      18 80 26 1
```

(Similar to previous example, but the right upper screen is taller than the left one, and the bottom screen has to be moved down to compensate.)

```
SET SCR define 43 20 1 1    43 20 1 61      18 40 14 21
```

(Defines two logical screens which are full-height 20 columns wide, and then an third 40-column screen in between them which is not full-height. Produces a sort of H-shaped display. Note that this would not be possible with XEDIT, since it requires that the full width of the screen be covered.)

```
'scr d 7 20 1 1 7 20 1 21 7 20 1 41 7 20 1 61',
      '7 20 8 1 7 20 8 21 7 20 8 41 7 20 8 61',
      '7 20 15 1 7 20 15 21 7 20 15 41 7 20 15 61',
      '7 20 22 1 7 20 22 21 7 20 22 41 7 20 22 61',
      '7 20 29 1 7 20 29 21 7 20 29 41 7 20 29 61',
      '8 20 36 1 8 20 36 21 8 20 36 41 8 20 36 61'
```

(Define 24 tiny splits. Note that the above assumes a 43-line EGA or VGA display. Also note that the command as shown is formatted for entry via a REXX macro.)

```
SCREEN 1
```

(Set the screen back to the startup default. For a 25-line display, the command

```
SCREEN SIZE 25
```

produces the same effect. For a 43-line display,

```
SCREEN SIZE 43
```

produces the same effect. For a 50-line display,

```
SCREEN SIZE 50
```

produces the same effect.)

SET SOUND COMMAND

```
SET SOUND ON
          OFF
```

The SET SOUND command is used to sound on or off. Setting SOUND to OFF gets rid of the beep output along with error messages.

Examples:

```
set sound on
set sound off
```

(Set SOUND to ON or OFF.)

SET STREAM COMMAND

```
SET STREAM OFF
           ON
```

The SET STREAM command is used to turn stream mode on or off. If STREAM is off, CLOCATE and CDELETE string searches are limited to the current line. If STREAM is on, these string searches may span line boundaries.

Examples:

```
set stream on
```

(Set STREAM to ON.)

```
SET STREAM OFF
```

(Set STREAM OFF. Column searches do not span line boundaries.)

SET SWAPEXEC COMMAND

```
SET SWAPEXEC [ OFF | ON ]
```

The SET SWAPEXEC command is implemented only in the DOS flavor of CP-Edit.

The SET SWAPEXEC command is used to turn on or off the use of the SWAPEXEC command (see the SWAPEXEC PACKAGE on the PCTOOLS conferencing disk) when executing DOS commands. Use of SWAPEXEC allows larger DOS commands to be executed. Typically, only 10-12K of storage is retained by SWAPEXEC out of the total storage available before loading CP-Edit.

The default is SWAPEXEC OFF.

Examples:

```
set swapexec on
```

(Turn on use of SWAPEXEC.)

```
set swapexec off
```

(Turn off use of SWAPEXEC.)

SET SYNONYM COMMAND

```
SET SYNonym  newname [n] oldname
```

The SET SYNonym command may be used to set a synonym for a command. The minimum abbreviation length (n) for the command may be set also. If the minimum abbreviation is not specified, the default is the length of newname.

The "oldname" may include multiple tokens.

Synonyms to previously-defined synonyms should not be used, since a synonym always resolves to a command.

The number of synonyms is limited only by available memory. However, if an excessive number of synonyms is defined, synonym resolution may become so slow as to noticably affect execution of commands.

If a new synonym is defined with the same "newname" as a previously defined synonym, the new synonym overrides the old one. To turn off a synonym, you must define it to its original value.

To bypass synonym resolution for one command, prefix the command with COMMAND. See COMMAND for details.

Examples:

```
set synonym erase 2 delete
```

(Set a synonym of ERASE for the DELETE command. The minimum abbreviation is ER.)

```
set synonym quit 3 macro pasquit
set synonym qquit 2 macro pasqquit
set synonym file 2 macro pasfile
set synonym end 3 macro pasend
```

(Set synonyms for the commands which exit a file to macros to do post-processing for the file before exiting.)

```
SET SYNonym cut macro cut (copy
```

(Add a synonym for the CUT macro with the "(copy" operand.)

```
SET SYNONYM qring 2 QUERY RING
SET SYN ALLCAPS SET CASE U R
SET SYN ALLLOWER SET CASE M I
```

(Set some synonyms for short forms for frequently-used commands.)

```
SET SYNONYM QUIT 4 COMMAND QUIT
SET SYNONYM QUIT 4 QUIT
```

(Turn off a previously defined synonym for QUIT. The two forms given are equivalent.)

SET TABLINE COMMAND

```
SET TABLine ON  [ n ]
                [M[n|+n|-n]]
                [n|+n|-n]

            OFF
```

The SET TABLine command is used to turn on, turn off, and set the position of the tab line on the screen. The line number may range between 3 and 23. It may not be the same as the CURLine, and it may not be the same as the SCALe if it is turned on. The default is OFF, line 9.

The line where the tab line is to be set may be specified as an absolute line number; or as a number preceded by a plus sign or minus sign, signifying a offset from the top of the screen (+) or the bottom (-); or "M" may be specified to mean the Middle of the screen, with or without an offset. The offset may be preceded by by a plus or minus sign for a positive or negative offset from the middle of the screen. If no sign is specified, the offset is assumed to be positive.

If the line number is omitted from a SET TABLine ON command, the last setting, if any is used. If the line number has never been set, the default of 9 is used.

Examples:

```
set tabl on 3
```

(Set the tab display on line 3 of the screen.)

```
SET TABLINE OFF
```

(Turn off the tab display.)

See Figure 33 on page 172 for a sample screen with the tab line display on.

```
  SAMPLE.PAS              TRUNC=255 SIZE=11 LINE=0
===>




      T-T-T-T-T-T-T-T-T-T----------T---------T---------T-------

00000 * * * TOP OF FILE * * *
00001 PROCEDURE SAMPLE (INPUT,OUTPUT);
00002   (* This is a sample Pascal program *)
00003
00004 VAR
00005   greeting : LSTRING(255);
00006 BEGIN
00007   greeting := 'Hello there!';
00008   Writeln (greeting);
00009
00010 END (* of the SAMPLE program *) .
00011 * * * END OF FILE * * *


F1 = HELP   F2 = FILE     F3 = QUIT     F4 = EDIT     F5 = =
F6 = ?      F7 = UP 10    F8 = DOWN 10  F9 = LEFT 10  F10= RIGHT 10
```

Figure 33. CP-Edit Screen with Tab Line On

SET TABS COMMAND

```
SET TABS n1 [n2...n28]
```

The SET TABS command is used to set the CP-Edit tab stops. The tab stops are used much as they would be on a typewriter. They control the positions to which the tab key moves the cursor. From 1 to 28 tab stops may be set. They need not be specified in order.

The tab stop positions may range between 1 and 255, since that is the range of column positions allowed by CP-Edit.

Examples:

```
SET TABS 1 3 5 7 9 11 13 15 17 19 30 40 50 60
```

(Set the tabs for structured programming indentation with Pascal. See Figure 33 on page 172 for a sample screen with the tab line display on and these values set.)

```
set tabs 1 5 10 15 20 25 30 35 40
```

(Set the tabs for general text processing.)

SET TRUNC COMMAND

```
SET TRunc n
          *
```

The SET TRunc command is used to set the truncation column.

No text may be entered or changed beyond the truncation column. If changes would expand text beyond the truncation column, the part of the line which would have extended beyond the truncation column is lost. When this occurs, a message is output.

The truncation column is set by default to the maximum length of a line (255 characters). Unless you specifically need to have text lost, or want to change text in the left part of the line without affecting text in the right part of the line (beyond the truncation column) it is best to leave it there. The truncation value may be set to the default value using the "*" operand.

Examples:

```
set trunc 60
```

(Sets truncation column to column 60.)

```
SET TR *
```

(Sets truncation column to the default of 255.)

SET VERIFY COMMAND

```
SET Verify [OFF] startcol endcol
           [ON ]    *        *
```

The SET Verify command is used to change the portion of each record which is displayed on the screen.

The default verify settings are columns 1 to 255. The verify start column may not be set less than 1. The verify end column column may not be set greater than 255 (the maximum width of a line of text). The verify start column column may not be set beyond the verify end column. To display just as much of each line as will fit on one screen line, set the verify columns to 1 and 74.

The verify columns may be set to their maximum values (1, 255) by using the "*" instead of a number.

The ON and OFF operands are provided for compatibility with the XEDIT SET Verify command. The ON or OFF is optional in CP-Edit, and has no effect. CP-Edit always behaves as XEDIT would if SET Verify OFF had been issued.

If the width of the displayed portion of lines is set wider than can be displayed on one line (more than 74 columns), the lines are wrapped on the screen.

The LEFT or RIGHT setting is reset by the SET Verify command.

Examples:

```
set verify 1 255
```

(Displays all columns of every line of the file.)

```
set verify off 10 40
```

(Displays only columns 10 through 40.)

```
SET VERIFY OFF 1 74
```

(Sets verify to one physical line on the screen, 1 to 74.)

SET WRAP COMMAND

```
SET WRap ON
         OFF
```

The SET WRAP command is used to specify whether or not CP-Edit "wraps around" to the top of the file if a LOCATE or FIND command reaches the bottom (or top, depending on the search direction) of the file without encountering its target.

If WRAP is set OFF, when the bottom (or top) of the file is reached, the FIND or LOCATE terminates with a "not found" message. If WRAP is set ON, when the bottom (or top) of the file is reached, the command wraps around to the top (or bottom) of the file and continues searching through the line preceding (or following) the current line.

The SET WRAP command affects only the current file being edited.

Note for macro writers: See also EXTRACT /WRAP/.

The default value for WRAP is OFF.

Examples:

```
set wrap on
wr off
```

(Set WRAP to ON or OFF.)

SET ZONE COMMAND

```
SET Zone leftzone rightzone
          *        *
```

The SET Zone command is used to set the zone of each record in which target string searches operate. It affects the Change, Locate, PUT, and PUTD commands and any others which use target strings. Searches for the Change command old string take place only within the zone also.

The default zone settings are 1 and 255 for the left and right zone columns respectively. The right zone column may not be set beyond the truncation column. Obviously the left zone column may not be set beyond the right zone column.

The zone columns may be set to the default values by using the "*" instead of a number.

Examples:

```
set zone 10 40
```

(Sets zone to the area from column 10 through 40. Target searches occur only in columns 10 through 40.)

```
SET Z 1 *
```

(Sets zone back to the defaults, 1 to 255.)

SI COMMAND

```
SI
```

The SI (structured input) command is used to enter lines into the file with automatic indentation. A new blank line is added to the file following the line where the cursor is positioned, 5 dots are placed in the prefix of the blank line, and the cursor is set to the column of the first non-blank character in the line. If you enter text on the new line provided and press 'Enter', another blank line is provided. If you do not enter any text on the blank line and press 'Enter', the blank line is deleted.

This command is useful for entering text with indentations, and for entering structured programming source which uses indentation to show program structure. (This command is similar to the input prefix command of the ISPF editor.)

See also the AUTOIN command and the SI prefix command.

Example:

```
SI
```

SORT COMMAND

```
SORT target [A] col1 col2 [col3 col4 ... ]
            [D]
```

The SORT command may be used to sort part or all of the file. Sorting takes place beginning with the current line and ending under control of the target.

The sort may be in ascending (A) or descending (D) order. If neither A nor D is specified, ascending order is the default.

The fields on which to sort are specified as pairs of columns. These pairs are the start and end columns of the fields. At least one pair of columns (one field) must be specified. The ending column of a field must be greater than or equal to the starting column (a 1-character-wide field is allowed).

The first field is the primary key, and subsequent fields are secondary keys. That is, the selected records are ordered by the primary key, then if primary keys are equal, by the next key, etc. As many fields may be specified as desired (since the maximum length command accepted by CP-Edit is 255 characters, in practice only about 60 fields may be specified).

Fields may overlap. CP-Edit does not check for this. But the results may not be what you wanted.

The ASCII collating sequence is used for sorting. If a particular record in the file is too short to include a key field, the record is sorted as if the field were blank. Since the blank is the lowest (text) character in the ASCII collating sequence, a blank field will generally sort lower (in an ascending sort) than one which is not blank.

The SET CASE Respect or Ignore setting affects the operation of the SORT command. If Respect is the current setting, keys sort in normal ASCII collating sequence, with lower case letters sorting higher than upper case letters. If Ignore is the current setting, upper case and lower case letters sort identically (lower case is "folded" into upper case). See the SET CASE command for details of its use.

A sorting algorithm is used which preserves the existing order of equal keys. The sorting algorithm is quite fast: most files can be sorted in a few seconds. The algorithm is linear; that is, sorting time is directly proportional to the number of records sorted and the number of columns in the keys.

No prefix commands may be pending anywhere in the current file during a sort. An error message is output if you try to sort with prefix commands pending. (Pending prefix commands may be removed using the RESET command, if desired.)

Example:

```
TOP
SORT * 1 8 10 12
```

(Sort the entire file. The primary key is in columns 1 through 8, and the secondary key is columns 10 through 12. The sort is in ascending order.)

```
:5
sort /File(s)/ d 30 31 24 25 27 28 39 39 34 35 37 38
```

Sort a file output by the DOS DIR command, sorting just the actual directory entry lines in inverse order by date and time. The file could have been produced using the DOS 2 I/O redirection and editing the resulting file with CP-Edit:

```
DIR > directry.fil
CP-Edit directry.fil

sort * 1 255
```

(Sort all columns of a file from the current record to the bottom of the file.)

SOS COMMAND

```
SOS    LINEAdd

       TABCmd

       TABCMDB [ n ]

       TABCMDF [ n ]
```

The SOS (screen operation simulation) command simulates certain screen operations.

The SOS LINEAdd command adds a blank line after the line containing the cursor.

SOS TABCmd moves the cursor to the beginning of the command line in the current logical screen (the one in which the command is issued, or, if the command is issued via a PF key, the one in which the cursor resides).

SOS TABCMDB moves the cursor to the previous logical screen.

SOS TABCMDF moves the cursor to the next logical screen.

Normally, logical screens are defined left-to-right and top-to-bottom on the physical screen, and the "next" logical screen means the next one to the right or the next one down. Similarly, the "previous" screen means the next one to the left or the next one up. However, it is possible to define screens in a arbitrary order (using the SET SCREEN DEFINE command), and in that case, "next" and "previous" refer to the order of definition of the logical screens.

In CP-Edit, the supported functions are mainly included for compatibility with XEDIT, since the editing function keys xtabcmd, xtabcmdf, and xtabcmdb are available, and are much faster.

By default, the xtabcmd editing function is assigned to ctrl-home, xtabcmdf is assigned to ctrl-PgDn, and xtabcmdb is assigned to ctrl-PgUp. These assignments can be changed using KEYDEF.

Examples:

```
SET PF2 SOS LINEADD
```

(Set PF2 to the XEDIT default. When PF2 is pressed, a blank line is added to the file following the line containing the cursor.)

```
SOS TABCmd
```

(Position the cursor at the beginning of the command line of the screen in which the command is issued.)

```
SET PF37 SOS TABCMDB
SET PF38 SOS TABCMDF
SET PF39 SOS TABCMDF 2
```

(Set PF37 and PF38 to move the cursor backward one screen and forward one screen respectively. Set PF39 to move the cursor forward two screens.)

SPLIT COMMAND

```
SPLIT CURSOR

      colno
```

The SPLIT CURSOR form of the command may be used to split a line into two lines at the cursor position.

This form of the command is most useful if it is assigned to a Program Function key. Let us suppose that you have assigned the SPLIT CURSOR command to Program Function key 12 ...

```
SET PF12 SPLIT CURSOR
```

and you want to split a line into two lines. Position the cursor to where the line is to be split and press Shift-F2 (for PF12) to split the line.

The SPLIT colno form of the command is used to split the current line at the specified column number.

Limitation: only one colno operand may be supplied.

Example:

```
SPLIT 10
```

(Splits the current line at column 10. Columns 1-9 form the first new line, and columns 9 through the end of the line form the second new line.)

If the split point is beyond the end of the line, the second line will be blank.

SPLTJOIN COMMAND

```
SPLTJOIN
```

The SPLTJOIN command splits the line containing the cursor or joins the following line to the line at the cursor position depending on where the cursor is in the file line. If the cursor is before or on the last non-blank character of the line, the line is split at the cursor position. If the cursor is beyond the last non-blank character of the line, the following line is joined.

The SPLTJOIN command is most useful when assigned to a PF Key:

```
SET PF11 SPLTJOIN
```

STATUS MACRO

```
STATus
```

The STATus macro may be used to display various settings of the SET command. Only the following settings are displayed:

```
ALT                 LINE
AUTOSAVE            MSGMODE
CASE                NBFILE
CURLINE             SIZE
CURSOR              TABS
FNAME               TRUNC
HEX                 VERIFY
IMAGE               WRAP
LENGTH              ZONE
```

Example:

```
STATUS
```

TOP COMMAND

```
TOP
```

The TOP command moves the current line to the top of the file. The top of the file is a pseudo-line which precedes the first real line of the file. Any insertions made when the current line is the top of the file go in front of the first real line of the file.

Example:

```
TOP
```

UP COMMAND

```
Up    [ n ]
      [ * ]
```

The Up command moves the current line backward (toward the beginning of the file) one or more lines. If the number of lines is omitted, 1 is assumed.

Up * is equivalent to the TOP command.

Examples:

```
UP
```

(Moves the current line backward one line.)

```
u 20
```

(Moves the current line backward 20 lines.)

UPPERCAS MACRO

```
UPPercas [ target ]
```

The UPPercas macro translates alphabetic characters in lines of the file from lower case to upper case. The UPPercas macro translates beginning with the current line over the range of lines specified by the target. If the target is omitted, one line is translated.

The UPPercas macro translates without regard to the current CASE setting. (Target search is, however, affected by the current setting of CASE Respect or Ignore.)

Example:

```
TOP
UPPERCAS *
```

(Translate the entire file to upper case.)

```
upper /end of section/
```

(Translate lines of the file beginning with the current line and continuing up to but not including the line containing the string "end of section".)

```
upp 3
```

(Translate 3 lines of the file beginning with the current line.)

= COMMAND

```
=
```

The equal sign command re-executes the last command entered. The command may have been typed in on the command line or it may have been entered as a result of pressing a PF key.

Please note that pressing F5 (or other function key to which an equal-sign command has been assigned) when a command is displayed on the command line will NOT result in executing the displayed command. It will only execute the previous command.

Example:

```
top
locate /xyz/
=
```

(Locate the first occurrence of the string "xyz" in the file, then locate the second occurrence using the "=" command.)

& (Ampersand)

```
&command
```

The ampersand (&) can be used preceding any command on the command line to cause the command to be redisplayed after execution. This allows you to execute a command multiple times just by pressing Enter.

* (Asterisk)

```
*(any text)
```

A CP-Edit comment is accepted but ignored by CP-Edit. It may be used to document the contents of a CP-Edit "native" macro file. It may also be used to set a PF key to a null value.

Examples:

```
* These are samples of CP-Edit comments.
* They may contain any text.
* They are useful for documenting non-REXX macros.
```

? COMMAND

```
?
```

The question-mark command re-displays on the command line the last command entered. The command may be executed in the usual fashion by pressing the Return key or any Function Key (other than one to which a question-mark command has been assigned). The command could have been typed in on the command line or it could have been entered as a result of pressing a Function Key.

Multiple question marks may be entered on the same line to display commands entered prior to the last.

CP-Edit saves only the last 8 commands. Commands executed from within macros are not saved, nor are "?" or "=" commands. The commands are saved in a ring, so that successive "?" commands can be used to cycle through all of the saved commands.

Successive entry of single question marks has the same effect as entering the same number of question marks on one line, but each saved command is displayed in succession.

It is useful to assign the question-mark command to a function key. The default assignment sets Function Key 6 to the "?" command.

Please note that pressing F6 (or other function key to which a question-mark command has been assigned) when a command is displayed on the command line will NOT result in executing the displayed command. It will only redisplay the previous command. This is true whether the command got on the command line by being typed in or by being re-displayed by the "?" command (or F6) or because it contained an error.

Example:

```
?
```

(Retrieve the last command executed.)

```
??
```

(Retrieve the next to last command executed.)

CP-Edit Prefix Commands

The following prefix commands are implemented in CP-Edit:

```
A         (Add)

I         (Insert)

/         (Set current line)

", ""     (Duplicate)

M, MM     (Move)

C, CC     (Copy)

MX, MXX   (Move external or PUTD)

CX, CXX   (Copy external or PUT)

G, GET    (Get)

SI        (Structured Input)

<, <<     (Shift left)

>, >>     (Shift right)

D, DD     (Delete)

P         (Preceding)

F         (Following)

SCALE     (Display Scale)

TABL      (Display tab line)

CE        (Text Center)

TF        (Text Flow)

TJ        (Text Justify)

.xxxx     (Set Symbolic Name)
```

In addition, some synonyms have been predefined in CP-Edit (These are for compatibility with the ISPF editor, another IBM large-computer editor):

```
B (Before) may be used as a synonym for Preceding

R, RR (Repeat) may be used as a synonym for ", "" (Duplicate)
```

A PREFIX COMMAND

```
A
An
nA
```

The A (Add) prefix command is used to insert one or more blank lines in the file. The line(s) are inserted following the line whose prefix contains the A.

The number of blank lines to be inserted may be specified either before or after the A in the prefix of the line. Af the number of lines is not specified, the default is 1.

I PREFIX COMMAND

```
I
In
nI
```

The I (Insert) prefix command is used to insert one or more blank lines in the file. The line(s) are inserted following the line whose prefix contains the I.

The number of blank lines to be inserted may be specified either before or after the I in the prefix of the line. If the number of lines is not specified, the default is 1.

/ PREFIX COMMAND

/

The / (slash) prefix command is used to set the current line. Place the slash (/) anywhere in the prefix area of the line which you want to be the current line.

" PREFIX COMMAND

```
"
"n
n"
""
""n
n""
```

The " (duplicate) prefix command is used to duplicate single lines or blocks of lines. A duplication factor may be specified either before or after the " or "". If a duplication factor is specified for the block form of the duplicate, it must appear on the first duplicate block prefix command (the one closer to the beginning of the file).

The block form of the command duplicates the range of lines starting with the first line marked and ending with the last line marked.

The duplicated lines are inserted in the file immediately following a single duplicated line or immediately following the end of the duplicated block.

R and RR (repeat) may be used as synonyms for " and "" respectively.

M PREFIX COMMAND

```
M
Mn
nM
MM
```

The M prefix command is used to move single lines or blocks of lines from one location to another in the file. A block of lines to be moved may be specified in two different ways: the beginning and end lines of the block may be marked with MM in their prefix areas, or the beginning of the block may be marked with a single M and a number of lines in the block. A single line to be moved is marked with a single M.

The P (preceding), B (before), F (following), or A (after) prefix command must be used to mark the position to which the lines are to be moved.

C PREFIX COMMAND

```
C
Cn
nC
CC
```

The C prefix command is used to copy single lines or blocks of lines from one location to another in the file. A block of lines to be copied may be specified in two different ways: the beginning and end lines of the block may be marked with CC in their prefix areas, or the beginning of the block may be marked with a single C and a number of lines in the block. A single line to be copied is marked with a single C.

The P (preceding), B (before), F (following), or A (after) prefix command must be used to mark the position to which the lines are to be copied.

MX PREFIX COMMAND

```
MX
MXn
nMX
MXX
```

The MX (Move External) is the prefix equivalent of the The MX prefix command is used to move single lines or blocks of lines from one file to another (or from one place in a file to another, but the M prefix command is more efficient for moving lines in the same file). A block of lines to be moved may be specified in two different ways: the beginning and end lines of the block may be marked with MXX in their prefix areas, or the beginning of the block may be marked with a single MX and a number of lines in the block. A single line to be moved is marked with a single MX. The lines are written to the editor temporary file, then deleted, exactly as with the PUTD command (any previous content of the temporary file is overwritten).

The lines may be inserted into another (or the same) file using either the GET command or the G/GET prefix command.

CX PREFIX COMMAND

```
CX
CXn
nCX
CXX
```

The CX (Copy External) prefix command is the prefix equivalent of the PUT command. The CX prefix command is used to copy single lines or blocks of lines from one file to another (or from one place to another in the same file, but the C prefix command is more efficient for copying lines in the same file). A block of lines to be copied may be specified in two different ways: the beginning and end lines of the block may be marked with CXX in their prefix areas, or the beginning of the block may be marked with a single CX and a number of lines in the block. A single line to be copied is marked with a single CX. The lines are written to the editor temporary file exactly as with the PUT command (any previous content of the temporary file is overwritten).

The lines may be inserted into another (or the same) file using either the GET command or the G/GET prefix command.

G PREFIX COMMAND

```
G
GET
```

The G or GET prefix command is the prefix equivalent of the GET command issued from the command line. It is used to insert into the file the contents of the editor temporary file. Note that no options are supported for the G/GET prefix command; that is, only the temporary file can be read, and the entire contents of the temporary file are inserted following the line on which the G/GET prefix command appears.

The editor temporary file is created using the PUT or PUTD command from the command line with no filename specified, or by using the CX or MX prefix command.

SI PREFIX COMMAND

```
SI
```

The SI (structured input) prefix command is used to enter lines into the file with automatic indentation. This prefix command adds a blank line to the file, puts 5 dots in the prefix of the blank line, and sets the cursor to the column of the first non-blank character in the line. If you enter text on the new line provided and press 'Enter', another blank line is provided. If you do not enter any text on the blank line and press 'Enter', the blank line is deleted.

This command is useful for entering text with indentations, and for entering structured programming source which uses indentation to show program structure. (This command is similar to the input prefix command of the ISPF editor.)

< PREFIX COMMAND

```
<
<n
n<
<<
<<n
n<<
```

The < (shift left) prefix command is used to shift text left within the line. The number of columns to be shifted may be specified before or after the < or <<. If the number of columns is not specified, the default is 1 column.

Single lines or blocks of lines may be shifted. Blocks of lines must be marked by placing a << in the prefix of the first line of the block and another << in the prefix of the last line of the block. If a number of columns to shift is specified, it must be specified on the first line of the block (the one nearest the beginning of the file).

> PREFIX COMMAND

```
>
>n
n>
>>
>>n
n>>
```

The > (shift right) prefix command is used to shift text right within the line. The number of columns to be shifted may be specified before or after the > or >>. If the number of columns is not specified, the default is 1 column.

Single lines or blocks of lines may be shifted. Blocks of lines must be marked by placing a >> in the prefix of the first line of the block and another >> in the prefix of the last line of the block. If a number of columns to shift is specified, it must be specified on the first line of the block (the one nearest the beginning of the file).

D PREFIX COMMAND

```
D
Dn
nD
DD
```

The D (delete) prefix command is used to delete single lines or blocks of lines from the file. A block of lines to be deleted may be marked in two ways: the beginning and end lines may be marked with DD, or the beginning line of the block may be marked with a single D and a line count (number of lines to be deleted).

The number of lines to be deleted may be placed before or after the single D in the prefix.

P PREFIX COMMAND

```
P
```

The P (preceding) prefix command is used to mark the line before which text is to be moved or copied. B (before) may be used as a synonym for P (preceding).

F PREFIX COMMAND

```
F
```

The F (following) prefix command is used to mark the line after which text is to be moved or copied. A (after) may be used as a synonym for F (following).

SCALE PREFIX COMMAND

```
SCALE
```

The SCALE prefix command is used to turn on the scale display on the screen. The scale replaces the line in whose prefix the SCALE prefix command is entered.

The scale can also be turned on using the SET SCALE ON command. To turn the scale off, the SET SCALE OFF command must be used.

TABL PREFIX COMMAND

```
TABL
```

The TABL prefix command is used to turn on the tab line display on the screen. The tab line replaces the line in whose prefix the TABL prefix command is entered.

The tab line can also be turned on using the SET TABLine ON command. To turn the tab line off, the SET TABLine OFF command must be used.

CE PREFIX COMMAND

```
CE
CEn
nCE
```

The CE (text center) prefix command is used to center lines of text. If the number of lines to be centered is not specified, one line is centered.

The margins used for text centering are the current zone columns, if they are set. If they are not (the left and right zone are at their default values of 1 and 255), the current verify columns or the current screen display columns are used. The zone columns may be set using the SET ZONE command.

TF PREFIX COMMAND

```
TF
```

The TF (text flow) prefix command is used to format blocks of text into more or less even-length lines. A block of lines to be flowed is marked by placing TF prefix commands on the the beginning and end lines of the block.

The margins used for text flowing are the current zone columns, if they are set. If they are not (the left and right zone are at their default values of 1 and 255), the current verify columns or the current screen display columns are used. The zone columns may be set using the SET ZONE command.

TJ PREFIX COMMAND

```
TJ
```

The TJ (text justify) prefix command is used to format blocks of text into left- and right-justified lines. A block of lines to be justified is marked by placing TJ prefix commands on the the beginning and end lines of the block.

The margins used for text justifying are the current zone columns, if they are set. If they are not (the left and right zone are at their default values of 1 and 255), the current verify columns or the current screen display columns are used. The zone columns may be set using the SET ZONE command.

PREFIX .xxxx (SET SYMBOLIC NAME)

```
.xxxx
```

The .xxxx prefix command can be used to assign a symbolic name to a line. The symbolic name may be referred to in a CP-Edit target.

Symbolic names may also be assigned using the SET POINT command. Symbolic names set with the .xxxx prefix command may be turned off using the SET POINT .xxxx OFF command.

The symbolic name may be any 4 characters. Case is significant; that is, the following are all different names:

```
.start
.Start
.START
```

A maximum of 5 names per file are supported currently.

More than one name may be associated with a line. A name remains associated with the line while the file is being edited, and as long as the line is not deleted or moved, or until the SET POINT .xxxx OFF command is issued for it.

Appendix A. Differences Between CP-Edit and XEDIT

XEDIT Compatibility

CP-Edit has been designed to mimic XEDIT fairly closely. XEDIT users will feel very much at home with CP-Edit. But inevitably there are a number of differences between CP-Edit and XEDIT. Some of the differences are quite subtle.

First, CP-Edit does not have all of the XEDIT commands. (With the amount of memory and the amount of processing power available on PCs, it is unlikely that CP-Edit will ever have **all** the XEDIT commands.) The set of XEDIT commands was chosen for maximum flexibility in editing and for minimum duplication of function. In many cases, XEDIT provides several ways to perform a function, while CP-Edit provides only the most commonly used way to perform the function. Some of the operands of the commands are more limited. See the lists of commands above for such implementation restrictions.

Next, there are a few differences which are due to the different operating system environments. For example, the form of filenames is that of DOS or OS/2 and not VM/CMS.

CP-Edit has a fair amount of the flavor of the ISPF editor also. The TF prefix commmand of CP-Edit is similar to the TF prefix command of the ISPF editor, except it is bounded by a second TF command, rather than a blank line. Also, the R, RR, and B prefix commands are from the ISPF editor. And the CP-Edit END macro is identical in function to the END command in the ISPF editor.

Then there are a few improvements which are only possible because of the PC environment.

Target Limitations

Targets as implemented in CP-Edit are somewhat restricted compared to those of XEDIT. See Section “CP-Edit Targets” on page 33 for a description of targets as implemented in CP-Edit.

Screen and Keyboard Differences

Most of the screen and keyboard use will be very familiar and natural to anyone who has used both the PC and a 327x terminal. Cursor movement is controlled by the cursor keys on the right side of the PC keyboard and the Tab and function keys on the top of the keyboard (or left side for older keyboards).

The arrow keys have their obvious function of moving the cursor one position up, down, left, or right. The Home key positions the cursor to the screen home position (the beginning of the command row -- line 2). The End key positions the cursor to the bottom of the screen without changing the column position. Ctrl-Home positions the cursor to the command line of the current screen.

The PgUp and PgDn keys are settable to half or full page scrolling. Their default settings are Backward Half and Forward Half respectively, the same as the PF7 and PF8 settings.

The Return key performs its normal PC function, that is, it is the equivalent of the Enter key on a 327x terminal.

The large Plus(+) key on the numeric keypad is used as the Next-line key (the equivalent of the Return key on a 327x). The Minus(-) key on the numeric keypad is used to perform the Erase-to-End-of-line function.

40 function keys are implemented in CP-Edit. See Appendix B, “Default Settings” on page 219 for their default settings. In addition, the Esc key is a "hard-coded" function key which is set to a "QUIT" command. Its setting may not be changed.

The forward Tab key actually works in CP-Edit. If the cursor is in the prefix area, the Tab key moves the cursor to the first column in the file area. If the cursor is to the right of the last prefix column, the Tab key positions the cursor to the first text column or to the next tab stop to the right.

The Back-tab key positions the cursor to the next tab stop to the left if the cursor was in the file area; otherwise, the cursor is positioned to the first prefix column.

If you insist on having the crippled tab key function of XEDIT on a 3270 terminal, you can set the tab keys to behave like XEDIT, either by setting just a single tab in column 1, or by using the xtab and xbacktab functions with KEYDEF. See KEYDEF.DOC for details.

The Backspace key performs a non-destructive backspace. This is identical to its use with XEDIT (it is the equivalent of the Left arrow), but different from its use in some PC DOS programs.

The PC-style destructive backspace is available through use of KEYDEF. See KEYDEF.DOC for details.

The Caps-lock key performs its familiar PC function. The word "CAPS" is displayed on the ID line of the screen in Caps-lock mode.

The Insert key toggles CP-Edit into and out of INSERT mode. The letters "INS" are displayed on the ID line of the screen in INSERT mode. In INSERT mode, characters are inserted on the screen prior to the character under the cursor. Non-blank characters may not be shifted off the right-hand edge of the screen. A beep results if you attempt this. INSERT mode does not function anywhere on the screen except the data area and the command line. INSERT mode is ignored in the prefix area. A beep results if you try to insert in one of the areas other than these. This is different from the behavior of a 327x terminal (a 327x terminal locks up if you try to overtype a protected field in INSERT mode).

Pressing the Return key or any PF key turns off INSERT mode. This is pretty much the same as the behavior of XEDIT on a 327x terminal.

The Delete key deletes characters under the cursor. The Delete key does not function when the cursor is not in the data area or the command line. If you attempt to delete characters elsewhere, CP-Edit signals its displeasure by beeping. The Delete key auto-repeats on the PC (except the 3270-PC).

The Num Lock key has its normal PC function.

The Ctrl-Break key combination is disabled in CP-Edit.

The distinction between null characters and blanks does not exist in CP-Edit. Advancing the cursor beyond the end of the current data on a line using the space bar or the right-arrow will produce exactly the same results. Advancing the cursor using the tab key will result in multiple spaces being added to the line.

Prefix Differences.

No more than one prefix command may be pending at any time. Unrecognized prefix commands generate an error message and are not left pending. Block commands may not overlap each other. Single prefix commands may not be contained within block commands.

Profile Differences

The default profile macro is PROFILE.MAC. The search for the profile is the same as the DOS command search; that is, it begins in the current directory of the current disk, and if not found there, uses the PATH. The syntax for overriding this name (-p profname) is consistent with the syntax of DOS, OS/2, and AIX commands, but different from that used with XEDIT.

Miscellaneous Differences and Limitations

The maximum length line which may be edited is 255 characters for DOS, 511 for OS/2, and 2047 for AIX.

The total amount of data that can be edited is limited only by the total amount of free memory available after CP-Edit is loaded.

If insufficient storage is available to start CP-Edit at all (enough for the program, one screen, static data, stack, and heap plus storage for function keys and enough to read in one file) CP-Edit will refuse to run, producing a "Not enough storage" message.

When storage fills up CP-Edit will refuse to enter any more data, and may refuse to do much of anything which will use up more storage. You should immediately FILE or QUIT one or more of the current files, or delete text from one or more of the files. The Query FREE command may be used to determine the amount of free storage remaining at any time.

As many files can be edited as will fit in storage.

A older virtual memory version of CPEDIT is available for DOS in CPEDITV PACKAGE on PCTOOLS. It allows editing of files larger than available memory.

Input mode is by file rather than global across all files in storage.

In Input mode, the prefix remains on the screen for the portion of the screen above the input area and prefix commands may be applied to lines in this area. In input mode, the cursor is initially positioned to the first text column (which is not the leftmost column on the screen), and pressing the newline key (keypad minus) positions the cursor to the first text column on the next line.

For PF key commands, BEFORE (the XEDIT default) is the only type of execution which is done; that is, the PF key command is always executed before the contents of the command line.

CP-Edit remembers only the last 9 commands entered. These 9 are all that can be retrieved using the "?" command.

The RECOVER command is rather limited in CP-Edit, due to PC memory limitations. Only the last 10 lines deleted from the file are saved.

The SET SCREEN command is fully implemented in CP-Edit, but the XEDIT restriction that the entire width of the screen be covered does not apply to CP-Edit. Also, the exact assignment of files to screens after a SET SCREEN command may not be identical to that of XEDIT.

The use of the SET IMage command by CP-Edit is different from its meaning in XEDIT. The SET IMage command in XEDIT affects the storing of blanks or tabs on input from the terminal. The SET IMage command in CP-Edit affects the storing of blanks or tabs on disk. Backspace characters are not affected by CP-Edit's SET IMage command, therefore there is no SET IMage CANON operand in CP-Edit.

The ASCII collating sequence is used by the CP-Edit SORT command. It is quite different from the EBCDIC collating sequence used by XEDIT. In particular, numeric characters sort higher than alphabetic characters in EBCDIC, but sort lower in ASCII; lower case letters sort lower than upper case letters in EBCDIC, higher in ASCII; almost all punctuation sorts lower than alphabetic and numeric characters in EBCDIC, whereas in ASCII, some punctuation sorts higher than numerics, some sorts between upper and lower case alphabetics, and some sorts above lower case.

Appendix B. Default Settings

Default Function key settings

```
F1  -- Help
F2  -- File
F3  -- Quit
F4  -- Edit
F5  -- =
F6  -- ?
F7  -- Backward half
F8  -- Forward  half
F9  -- Left 10
F10 -- Right 10
```

All other function keys (F11 - F40) are set to a CP-Edit comment (*).

On the Enhanced (101-key) Keyboard, F11 and F12 are implemented directly. On older keyboards, F11 and F12 are available as the shifted F1 and F2 keys. On either keyboard, Function keys 11-20 can be produced using the shifted function keys F1-F10. Function keys 21-30 are produced using Ctrl and F1-F10. Function keys 31-40 are produced using Alt and F1-F10. For example:

```
Ctrl-F7 produces F27.

Alt-F10 produces F40.

Shift-F4 produces F14.
```

The settings of the function keys can be set using the SET PFn (Set Program Function Key) command. It is often useful to set them in your CP-Edit Profile (PROFILE.MAC).

Default Synonyms

The following synonyms are built in to CP-Edit:

```
SET SYNONYM HELP        1 MACRO HELP
SET SYNONYM MERGE       2 MACRO MERGE
SET SYNONYM REPLACE     1 MACRO REPLACE
SET SYNONYM STATUS      4 MACRO STATUS
```

Any of the above may be overridden using the SET SYNONYM command.

Default SET values

The following are the defaults for the CP-Edit SET values other than the Program Function keys and synonyms:

```
SET AUTOSAVE OFF

SET CASE MIXED RESPECT

SET CMDLINE TOP

SET CURLINE ON 11

SET ENTER IGNORE

SET HEX OFF

SET IMAGE ON

SET LINEND ON #

SET NUMBER ON

SET MSGMODE ON DELAY

SET PREFIX ON LEFT

SET SCALE OFF 10

SET SCREEN 1

SET TABS 1 10 20 30 40 50 60 70

SET TABLINE OFF 9

SET TRUNC 255

SET VERIFY OFF 1 255

SET WRAP OFF

SET ZONE 1 255
```

Any of the above may be changed by using the appropriate SET command. See the SET commands in Section "CP-Edit Commands" on page 45.

On color displays, CP-Edit behaves as if the following SET commands had been issued:

```
SET COLOR ARROW WHITE HIGH

SET COLOR CMDLINE PINK HIGH

SET COLOR CURLINE WHITE HIGH

SET COLOR FILEAREA GREEN

SET COLOR IDLINE WHITE HIGH

SET COLOR MSGLINE RED

SET COLOR PENDING YELLOW HIGH

SET COLOR PREFIX GREEN

SET COLOR SCALE YELLOW HIGH

SET COLOR STATAREA WHITE

SET COLOR TABLINE YELLOW HIGH

SET COLOR TOFEOF WHITE
```

On monochrome displays, CP-Edit behaves as if the following SET commands had been issued:

```
SET COLOR ARROW HIGH

SET COLOR CMDLINE NOHIGH

SET COLOR CURLINE HIGH

SET COLOR FILEAREA NOHIGH

SET COLOR IDLINE HIGH

SET COLOR MSGLINE HIGH

SET COLOR PENDING HIGH

SET COLOR PREFIX NOHIGH

SET COLOR SCALE HIGH

SET COLOR STATAREA NOHIGH

SET COLOR TABLINE HIGH

SET COLOR TOFEOF NOHIGH
```

The above may be changed with the appropriate CP-Edit SET COLOR commands. See the SET commands in Section “CP-Edit Commands” on page 45.

In addition, CP-Edit behaves as if the following XEDIT SET commands had been issued prior to invocation (these values cannot be changed):

```
SET IMAGE ON

SET NULLS ON

SET FULLREAD ON

SET STAY ON

SET RECFM V

SET LRECL 255

SET MSGLINE ON -1 22 OVERLAY

WIDTH 255
```

(Note that the CP-Edit SET IMAGE command does not affect the way the keyboard tab key is interpreted. It is different from the XEDIT SET IMAGE command. See Appendix A, “Differences Between CP-Edit and XEDIT” on page 215 for a discussion of the differences.)

Appendix C. CP-Edit Messages

CP-Edit messages are generally self-explanatory. In some cases, some additional explanation is attached to the messages below.

CP-Edit issues three categories of messages: informational messages, warning, and error messages. Informational messages are intended merely to give information on what is happening, to respond to queries, etc. Warning and error messages indicate that CP-Edit has had some difficulty doing what you have requested or it is warning you of some potentially dangerous condition.

Warning and error messages are output using the MSGLINE color (default is red on color displays and intensified on monochrome displays) and are accompanied by a "beep" or bell character. Information messages are output in the STATAREA color.

Informational Messages:

'(nnn) Occurrence(s) changed on (nnn) line(s)'

Produced by the Change command. Tells you how many changes were made.

'New file'

Produced by Edit command or initial entry into CP-Edit. The file did not previously exist on the disk or in storage.

'End of File'

Produced by BOttom and Next commands.

'Top of File'

Produced by TOP and Up commands.

'Input:'

Produced by Input command. CP-Edit has switched to Input Mode.

'Autoindent:'

Produced by the AUTOIn command. CP-Edit has switched to Auto-indent mode.

'Press Space bar to continue'

Produced whenever full-screen output is displayed (PF key display, DOS command output, etc.).

Warning and Error Messages:

' No room on screen for input'

Produced by the Input or AUTOIn command. The current line has been set too close to the bottom of the screen so that there is no room for the input area on the screen. The current line should be set higher on the screen, or the scale or tabline could be turned off or moved higher on the screen if either of them is in the way.

' Line not inserted'

Produced by the Input command. Usually accompanied by 'Out of memory' message.

' Unable to replace line in file'

Produced by the Change, JOIN, and SPLIT commands. Usually accompanied by a "Out of memory" message. There is insufficient memory available to complete the command.

' Target line does not follow current line'

Produced by the Change, PUT, and PUTD commands. Generally this means you used an absolute line number target (:nn) which did not follow the current line. Since these commands only operate forward (downward) in the file, the target is not allowed. This can also be produced if you try to PUT at the bottom of the file.

' Extra parameter "xxxx", ignored'

Produced when extra information is found on CP-Edit command line. Could also be caused by incorrect specification of filename or profile name on command line (spaces within the filename or profile name, for example).

' File will not fit in memory'

Produced by Edit, GET, or loading of first file. Usually accompanied by 'Out of memory' error message.

' Unable to join line'

Produced the JOIN command. Accompanied by 'Out of memory' error message. There is insufficient memory to do the JOIN operation.

' Maximum macro nesting depth exceeded'

Produced by the MACro command when "native" macros are nested too deeply. Maximum nesting depth is 4.

'File has been changed. Use QQuit to quit anyway'

Produced by the QUIT (protected quit) command. The file has been changed since it was last saved on disk.

' Too many files open: (filename)'

Produced by Edit, GET, PUT, PUTD, SAVE, and FILE. CP-Edit is unable to open a file because the DOS FILES=nn parameter in CONFIG.SYS is set to too low a number. It must be set high enough to allow the five DOS standard files plus at least three for CP-Edit. Since the default number of 8 should allow CP-Edit to operate normally, this error is an unusual condition. See the Configuring section of the DOS manual for further information.

' File is Read-only'

Produced by FILE or SAVE command. You may not FILE or SAVE a HELP file or other read-only file. You must QUIT (or QQuit) to exit a HELP file (or press the Esc key).

' Out of memory'

Produced by any command or full-screen editing function that causes most of the memory available to CP-Edit to be used up. When this message is displayed, you must FILE or QUIT (or QQuit) from one or more of the files in storage or delete lines from one or more files in storage.

When this error message is displayed, the memory available has fallen below the reserve limit built into CP-Edit. In this state, CP-Edit will be unable to perform most editing operations. You should FILE the current files if they have been changed in memory.

If, however, a file has not been read into memory completely, (accompanied by a 'File will not fit in memory' message) you must QUIT. The file is made read-only so as not to save an incomplete file.

'Entering DOS command mode.'
'Enter 'EXIT' to return to CP-Edit'

Produced by the DOS command when you have entered just 'DOS' without a DOS command to execute. You must enter the DOS 'EXIT' command in order to return to CP-Edit.

' COMMAND.COM not found. Cannot execute DOS command'

CP-Edit has been unable to find a copy of the DOS command processor, and therefore cannot execute DOS commands. The DOS command processor should either be on the current disk in the current directory, or be wherever the COMSPEC= parameter of the current environment points.

' Insufficient memory for DOS command execution'

You must allow sufficient free memory for DOS command execution. The minimum amount with DOS 2.x is about 20K bytes.

' Return code nn from DOS command execution'

An unrecognized return code from DOS internal function 4BH has been received by CP-Edit. This should not occur.

'Zone(s) adjusted to truncation column'

Warning issued by the SET TRunc command when the truncation column is set to a value less than the right zone column. The right zone is adjusted to a value equal to the truncation column. If the left zone is then less than the right zone, it is set equal to the right zone.

' Cannot sort. Pending prefix command.'

No prefix commands may be pending while sorting. The prefix commands may be removed by using the RESET command.

' Block command overlap'

Produced by Prefix command processing. Block prefix commands may not overlay. Single Prefix commands may not appear within a block prefix command.

' More than one pending block command'

Produced by Prefix command processing. Only one prefix command may be pending.