intel®

Development Solutions

Debug Editor User's Guide

Order Number: 167098-002

Notational Conventions

The following notational conventions are used throughout this manual:

italics	are for variable expressions. You substitute a value or symbol.
{*items*}	between braces indicate that you must select one and only one item in the enclosed menu.
[*items*]	between brackets are optional items of which you can select one and only one.
device	stands for the letter or number of a disk drive.
dirname	stands for any directory created by a user.
filename	is a valid file name.
pathname	specifies a path to a file. It can include *device*, *dirname*, and *filename*.
shading	is used to indicate user input.
<CNTL>	denotes the host keyboard's control key. For example, <CNTL>C means enter C while pressing the control key.
<RETURN>	denotes the host keyboard's carriage return key.
hlt>	denotes the debugging system's input prompt. On some systems, the prompt is an asterisk (*).

DEBUG EDITOR USER'S GUIDE

Order Number: 167098-002

REV.	REVISION HISTORY	DATE
-001	Original Issue.	6/86
-002	Final Typeset Version.	9/86

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

CAUTION

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. As temporarily permitted by regulation, it has not been tested for compliance with the limits for Class A Computing Devices pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the government, in accordance with the software license agreementas defined in FAR 52.227-7013.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

Intel Corporation retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

Above	im	iSBC	PC BUBBLE
BITBUS	iMDDX	iSBX	Plug-A-Bubble
COMMputer	iMMX	iSDM	PROMPT
CREDIT	Insite	iSXM	Promware
Data Pipeline	Intel	KEPROM	QueX
FASTPATH	intel	Library Manager	QUEST
GENIUS	intelBOS	MAP-NET	Quick-Pulse Programming
Δ	Intelevision	MCS	Ripplemode
i	inteligent Identifier	Megachassis	RMX/80
i	inteligent Programming	MICROMAINFRAME	RUPI
I^2ICE	Intellec	MULTIBUS	Seamless
ICE	Intellink	MULTICHANNEL	SLD
iCEL	iOSP	MULTIMODULE	UPI
iCS	iPDS	ONCE	VLSiCEL
iDBP	iPSC	OpenNET	4-SITE
iDIS	iRMX	OTP	
iLBX			

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

MULTIBUS® is a patented Intel bus.

IBM is a registered trademark of International Business Machines.

CONTENTS

CHAPTER 3 MACRO COMMANDS

PREFACE

About This Manual

This manual provides the information needed to use the debug editor supplied with Intel emulators and debugging tools. The information in this manual pertains to the Intel Intellec® Series III and Series IV development systems, and IBM PC AT and PC XT computer systems.

The following is a brief outline of the chapter contents:

Chapter 1 presents an overview of the editor.

Chapter 2 describes each of the screen editor's commands.

Chapter 3 describes the MACRO command and how to create and use macros to simplify editing routines.

1 OVERVIEW

1.1 Introduction

This chapter provides an overview of the debug editor.

Intel debugging tools have two types of editors:

- Command line editor — The command line editor enables you to edit system commands while you enter them at the keyboard or to recall previous commands for execution.
- Screen editor — The screen editor enables you to edit commands, debug definitions, and disk files without having to exit from the system software.

The following sections describe these editors.

1.2 The Command Line Editor

This editor is available for use while your system is in input mode. The command line editor makes it possible for you to edit commands as you enter them at the keyboard.

The command line editor utilizes a 400-character command history buffer. The history buffer stores commands as you enter them at the keyboard. Use the <↑> and <↓> directional arrow keys to scroll forward and backwards through the history buffer. After you retrieve a command from the history buffer, you can edit the command as if you were entering it from the keyboard.

The command line editor uses the directional arrow keys and special edit keys to alter command lines. After editing a line, you can enter a carriage return regardless of the position of the cursor without losing the information to the right of the cursor.

The command line editor special edit keys are listed in Table 1-1. Under Key Name in the table, when <CNTL> is combined with another key, as in <CNTL>A, you should hold down the <CNTL> key while pressing that key.

CAUTION

Do not press <CNTL>D on Series III hosts. <CNTL>D invokes the ISIS DEBUG-86 debug monitor. Press G <RETURN> to exit the monitor.

Table 1-1 Command Line Editor Special Function Keys

Key Name	Function
<RUBOUT>	Deletes the character to the left of the cursor.
<CNTL>A	Deletes the cursor position and everything to the right of the cursor to the end of the line.
<CNTL>C	Cancels the command in progress. Use <CNTL>BREAK on IBM hosts.
<CNTL>E	Re-executes the last command.
<CNTL>F	Deletes the character at the cursor and adjusts spacing.
<CNTL>X	Deletes the part of the line to the left of the cursor and closes the space.
<CNTL>Z	Deletes the current line.
<ESC>	Invokes the debug editor and places the present command in the edit buffer for editing. If you press <ESC> at the prompt, the previous command is retrieved from the history buffer for editing.
<←>	Moves the cursor left one character.
<→>	Moves the cursor right one character.
<↑>	Retrieves the previous line from the history buffer for editing.
<↓>	Moves to the next line in the history buffer.
<HOME>	Extends the effect of the last arrow key. Causes jumps to the beginning or end of the current line when used with the right or left arrow.

1.2.1 Example of Using the Command Line Editor

The following examples assume that debugging software is involved and that your system supports the commands listed. Perform the following steps to quickly learn how to use the command line editor. (The examples assume a hlt> prompt.)

1. Enter the following command:

   ```
   hlt> MENU = FALSE<RETURN>
   ```

   The system responds with the input prompt. Notice that the syntax menu at the bottom of the screen is now off.

2. Press the <↑> key to retrieve the command entered in step 1. Use the <↓> key to position the cursor at the F in FALSE. Press <CNTL>A to delete FALSE from the command line. Modify the command as shown next but do not press the <RETURN> key until you perform step 3.

   ```
   hlt> MENU = TRUE
   ```

3. TRUE is misspelled. Position the cursor one space beyond the W in the command entered in step 2. Press the <RUBOUT> key to rubout the W. Type in an "E" and press the <RETURN> key.

   The system responds with the input prompt and also returns to the syntax menu.

Experiment with the command line editor as you use your system. You can press the <RETURN> key regardless of the cursor position and execute the command (provided it is syntactically correct).

NOTE

The history buffer is limited in size. When entries exceed the buffer limit, the oldest commands are deleted from the buffer. Use <CNTL>E to execute the last command entered. You do not need to retrieve it from the history buffer prior to entering <CNTL>E.

1.3 The Screen Editor

Intel debugging tools have a built-in screen editor modeled after Intel's Aedit editor. The screen editor is an interactive, screen-oriented editor with menu style command prompts and special function key support, that runs under the development system software.

The screen editor takes advantage of CRT capabilities to enable you to perform the following functions:

- Display and scroll text on the screen
- Move the cursor to any position in the text file or to any point on the screen
- Correct typing mistakes
- Rewrite text by overstriking old characters
- Make insertions and deletions easily
- Insert hexadecimal
- Jump to the beginning, end, or tagged text

To simplify debug editing, the screen editor also provides features which enable you to:

- Find any character string and replace it with another character string
- Move or copy sections of text within a file or to another file
- Create macros to execute several commands at once, thereby simplifying repetitive editing tasks
- Indent text automatically
- View lines over 80 characters in length
- Scan listing files while editing your primary file
- Edit files without leaving the emulation environment
- Use hexadecimal input

The following sections describe the screen editor.

1.4 Entering the Screen Editor

There are two ways you can invoke the screen editor; by pressing the <ESC> key, or by typing EDIT <RETURN>. Both the <ESC> key and EDIT command invoke the same edit function. Use the EDIT command to create or modify previously-defined debug objects or disk files. Use the <ESC> key to display and modify the text of the last command entered. This sequence includes all text entered since the last main level prompt was displayed. The syntax for the two invocation commands is:

EDIT [*edit-item*] <RETURN>

<ESC>

Where:

EDIT	invokes the screen editor.
edit-item	is a valid edit item for your debugging system. Refer to the EDIT command entry and menu entry in your debugging system's user's guide for the exact syntax. What you can initially bring into the editor may vary between systems, however, the editor functions and commands do not vary.
<ESC>	retrieves the last command for editing. Press <ESC> either at the system input prompt or during command entry.

If you press <ESC> immediately after the input prompt, the previous command is retrieved from the history buffer for editing. If you press <ESC> while entering a command, the incomplete command is brought into the edit buffer for editing.

If you press <ESC> immediately after defining a procedure, repeat loop, or other construct, the entire construct is placed in the editor, even if the definition has errors.

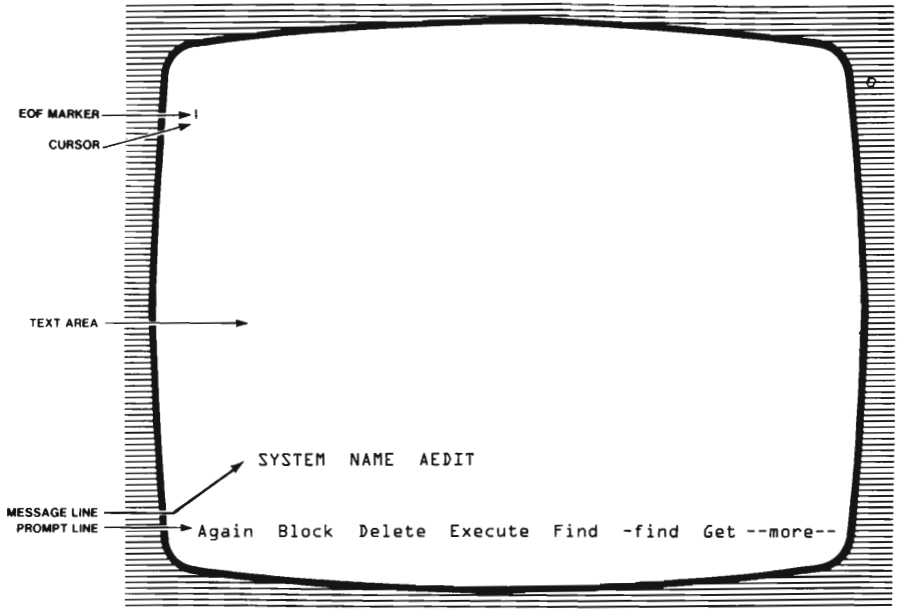
If you enter EDIT immediately after the input prompt and you do not specify a debug object for editing, you get a blank edit screen. A disk file can be called with the G(et) command.

If you enter EDIT and specify the name of a debug object residing in system memory, then that debug object is brought into the edit buffer for editing. Debug objects are LITERALLYs, BRKREGS, TRCREGS, and PROC definitions.

1.5 Screen Display and Menu Format

The screen editor uses a 25-line (24-line for the Series IV development system), 80-column display screen (columns are numbered from 0 to 79). The cursor, a solid, nonblinking block, is the reference point for all screen editor functions. The screen display is divided into three sections (refer to Figure 1-1):

- The prompt line
- The message line
- The text area



121756-1

Figure 1-1 Editor Display Screen

1.5.1 The Prompt Line

The prompt line is the bottom line of the display. The three types of prompts are:

- Menu prompts
- Line-edited prompts (i.e., a prompt that requires the user to type more than one character)
- Yes-no prompts

When the screen editor is invoked, the menu prompt is at the main command level. Menu prompts are a partial list of up to eight words that indicate which editing commands are available.

The word "more" indicates that pressing the <TAB> key displays the next line of menu prompts. Figure 1-2 shows the three menu prompt lines available at main command level.

```
----system-name AEDIT
Again  Block  Delete  Execute  Find  -find  Get    --more--
```

```
----
Hex  Insert  Jump  Macro  Other  Quit  Replace    --more--
```

```
----
?replace  Set  Tag  View  Xchange   --more--
```

Figure 1-2 The Menu Prompts

The menu prompt line offers a selection of main level commands. Specify a selection by pressing the initial letter of the command. At the command level, the upper or lower case of the letter is not significant. For example, press I(nsert) to create new text. Press the ESC(ape) key to discontinue text creation and return to command mode. If an incorrect character is entered, the message line above the menu writes "---illegal command". Some commands result in one or more subcommands. For example, press J(ump) and a new menu appears listing options for the jump command.

Line edited prompts ask for information (such as a filename) requiring more than a single character user response. The response can be up to 32 characters.

The yes-no prompts use the following form:

```
all changes lost? (y or [n])
```

The brackets indicate the default response. For this example, any response other than y or (Y) is considered a negative response.

1.5.2 The Message Line

The message line is located directly above the prompt line and is used to display status messages or to indicate command modes (refer to Figure 1-3).

The message line contains a blank in the first column followed by four hyphens. The word "Macro" or "Other," or both, may follow the hyphens to indicate that a macro is being defined or that the secondary buffer is being displayed. The hyphens in the message line never change unless OTHER or MACRO mode is changed. The remainder of the message line is changeable because it is used for status messages or used to indicate the count (repeat function) of a command.

1.5.3 The Text Area

The text area comprises the top 23 lines of the screen. These 23 lines display the text being edited. When the text is greater than 23 lines, scroll through the file using the <↑> and <↓> keys or the JUMP command (see section 2.11).

1.6 Special Function Keys

Almost every keyboard character can be considered a command because each key causes something to happen. In addition to the keys that select the menu items, the keys listed in Table 1-2 perform special functions.

CAUTION

Do not press <CNTL>D on Series III hosts. <CNTL>D invokes the ISIS debug monitor. Press G <RETURN> to exit the monitor.

1.7 Special Screen Indicators

1.7.1 The EOF Marker

The vertical bar (EOF marker) marks the end of text in the file (refer to Figure 1-1). When you invoke the editor without specifying a command or debug object, the EOF marker appears at the beginning of the file. As you type text into the file, this symbol moves and continues to mark the end of the file.

1.7.2 The Cursor

The cursor is displayed on the screen as a solid non-blinking block. When you invoke the editor and do not load a debug object or system command, the cursor is positioned over the EOF marker. Entering the editor with <ESC> or with a debug object definition places the cursor

```
----
Again  Block  Delete  Execute  Find  -find  Get    --more--
```

Figure 1-3 The Message Line

Table 1-2 Screen Editor Special Function Keys

Key Name	Function
<RUBOUT>	Deletes the character to the left of the cursor and closes the space. If a line feed is deleted, the preceding carriage return is also deleted. (The ← key on the top row of an IBM PC also performs this function.)
<CNTL>A	Deletes the character under the cursor and all characters to the right of the cursor to the end of the line. (You can also use the CHAR DELETE key on the Series IV and the DEL key on an IBM PC.)
<CNTL>C	Aborts the command in progress.
<CNTL>F	Deletes the character under the cursor and adjusts spacing.
<CNTL>U	Restores up to 100 characters deleted by the last <CNTL>A, <CNTL>X, or <CNTL>Z to the current cursor position. Consecutive <CNTL>U's will repeat the restoration.
<CNTL>X	Deletes all characters to the left of the cursor and adjusts the spacing. (The CLEAR LINE key on a Series IV also performs this function.)
<CNTL>Z	Deletes the entire line on which the cursor is positioned. The cursor does not move.
<ESC>	Terminates the editing command in progress and returns the editor to the main command level. For some commands, <ESC> executes the command (for example, FIND).
<RETURN>	Moves the cursor to the beginning of the next line. The end of line/line feed character is not displayed on the screen. The use of the EOL key in examples of input is indicated by <RETURN>.
<TAB>	Rotates the menu prompt line to display the next line of commands. When in mode command level, <TAB> sets tabs as indiated by the tab setting.
<←>	Moves the cursor left one character.
<→>	Moves the cursor right one character.
<↑>	Moves the cursor up one line.
<↓>	Moves the cursor down one line.
<HOME>	Works in conjunction with the arrow keys to page forward or backward through a file or to cause a jump to the beginning or end of a line.

at the beginning of the text. The cursor moves in the direction indicated by the arrows on the cursor control keys (the cursor does not move past the EOF marker). Pressing the <HOME> key after an arrow key allows for fast cursor movement, depending on the prior cursor movement command. (The arrow keys cannot be used to move the EOF marker.)

1.7.3 Lines and Line Terminators

A line of text consists of a character string terminated by an end-of-line/line-feed (EOL). This pair of characters, called the line terminator, is entered in the file when you press the <RETURN> key. The EOL is displayed on the screen as a blank.

If a line is over 80 characters long, an exclamation point (!) is displayed in the last column on the screen and the portion of the line that does not fit is not displayed. To view the portion of the line that is not displayed, use the SET Leftcol command.(see section 2.17.5).

1.8 Printing and Non-Printing Characters

All characters are displayed on the screen except <ESC>, NUL, BELL, backspace, and characters with hexadecimal values equal to or greater than 7F. All characters that are not displayed on the screen print as a question mark (?). The <RETURN> and <TAB> keys display as blanks.

1.9 Tags

Tags identify locations within a file. You can specify up to four locations, A through D, with the TAG command (see 2.18) and use them as destinations for the JUMP command.

1.10 Repeat Function

To repeat certain commands 1 to 65535 times (some commands ignore count or limit count to 32) enter a number prior to entering a command letter. For example, 5R(eplace), indicates that the replace command is to be executed five times. Enter a forward slash (/) to repeat the command from the cursor to the end of text.

1.11 Edit Buffers

The screen editor has three edit buffers:

- The primary buffer
- The secondary buffer (other)
- The BLOCK buffer

The primary buffer is the text area entered when the editor is first invoked. The size of the buffer is constrained by the amount of memory available from the host.

The secondary buffer enables you to examine, search, or borrow from another file while editing the primary buffer. The secondary buffer is accessed with the OTHER command (see section 2.13) and has a fixed maximum size of 7 KB.

The block buffer is used by the BLOCK command (refer to section 2.3) as the storage area for text you move, copy, or delete. The block buffer has a fixed maximum size of 2 KB.

1.12 Example Edit Session

Following is a short editing session illustrating use of the screen editor.

To enter the editor without calling a debug object or previous system command, type:

```
hlt>EDIT<RETURN>

----
Again Block Delete Execute Find -find Get  --more--
```

Press the G(et) key to move text into the primary buffer.

```
G
```

The editor prompts with:

```
Input file:
```

Enter the name of the file to edit using the complete pathname, for example:

```
Input file: C:\ICEDIR\JUNE16.LOG<ESC>
```

NOTE

You can press either the <ESC> key or <RETURN> key to execute the GET command. This is true for any command that accesses disk files (Init, Write, Put).

Once the file is in the buffer, use any of the edit commands available from the menu to view or modify text.

Assume you have a file in the buffer and you are finished editing it. To exit from the editor and return to the system software, press Q(uit). This command gives you a submenu with the option to simply A(bort), losing any changes to the file, E(xecute) the system commands in the file, I(nit) to call another file, or W(rite) the contents of the buffer back to the original file or to another file.

When exiting an editing session in which you want to save the file to disk, use the W(rite) option to save the changes. When the file has been written either to a new disk file, or to the

previous disk file, exit the editor with the A(bort) command. The command asks if you want to lose changes, you can type Y(es), because the changes are stored in the file.

1.13 Editing a BRKREG

This example assumes a BRKREG called brk1 has been created and is in memory, not on a disk file. To edit brk1, enter:

```
hlt> EDIT brk1<RETURN>
```

The BRKREG definition is displayed on the screen for editing.

```
define brkreg brk1 = myprogram#23-1:main.proc1
```

Make any editing changes you want. Assume you want to save the BRKREG definition for future use; press Q(uit). The menu prompts:

```
Abort Execute Init Write
```

Press W; the editor prompts for the output file name. Enter the pathname you want to write the BRKREG definition to. After the file has been written, press A(bort) to return to the system software, E(xecute) to re-execute the BRKREG definition (assuming you made changes to it), or I(nit) to start a new editing session.

Chapter 2 explains each of the screen editor commands.

2 SCREEN EDITOR COMMANDS

2.1 Menu Commands

This chapter explains all commands as they appear on the screen editor menu.

When invoked, the screen editor displays the editor commands menu on the second line from the bottom of the screen. Press the <TAB> key to display additional commands. Entering the first letter of the keyword from each menu invokes that function. Figure 2-1 shows the main menu prompt lines.

The following sections describe each of the screen editor commands listed in Figure 2-1.

2.2 AGAIN (A)

The AGAIN command repeats the last edit command.

If AGAIN is used after a FIND, -find, REPLACE, or ?replace command, the entire command, including arguments, is repeated. This command may be repeated any number of times.

2.3 BLOCK (B)

The BLOCK command enables you to delimit a section of text that can then be deleted, moved, or copied, depending on the subcommand you specify. Text saved in the block buffer can be retrieved with the GET command which copies the contents of the block buffer to the current cursor position in your file (refer to the GET command entry later in this chapter).

To delimit a section of text, position the cursor at the first character in the block of text desired and press B. An at sign (@) will be superimposed over the character, marking one endpoint of the block. (Endpoints can be set in either order, the examples here explain setting the beginning endpoint first.)

The editor prompts with the following subcommands:

```
Buffer Delete Find -find Jump Put
```

Position the cursor one character past the end of the desired block. Select one of the BLOCK subcommands to place the delimited block of text in the buffer. The character under the beginning @ sign is copied into the buffer, but the one under the second @ sign is not.

The following sections describe the BLOCK subcommands.

```
----
Again  Block  Delete  Execute  Find  -find  Get  --more--
```

```
----
Hex  Insert  Jump  Macro  Other  Quit  Replace  --more--
```

```
----
?replace  Set  Tag  View  Xchange  --more--
```

Figure 2-1 The Main Prompt Lines

2.3.1 Buffer

Press B(uffer) to copy the block of text into the buffer. The text is unchanged, the at signs (@) are removed, and the delimited block of text is copied into the buffer. If the block of text to be copied is over 2 KB, the remainder is written to an unnamed temporary work file (the file is erased when the editor is exited). The contents of the 2 KB buffer remain unchanged until you execute another BLOCK command.

Retrieve the contents of the buffer with the GET command (refer to the GET entry later in this chapter).

2.3.2 Delete

Press D(elete) to move a block of text from the screen and from its current place in memory to the 2 KB buffer. If the text to be moved is over 2 KB, the editor prompts:

```
cannot save in memory--save on disk? ( [y] or n )
```

Respond with N if you do not want to save the text; any other key if you want to save the text. If you save the text, it can be retrieved with the G(et) command. (Refer to the GET entry, 2.8).

2.3.3 Find

Press F(ind) to search forward in the text for the character where you wish to place the second @ delimiter. See the FIND command entry later in this chapter for more information.

2.3.4 -find

Press -(find) to search backward in text for the character where you wish to place the second @ delimiter. See the -find command entry later in this chapter for more information.

2.3.5 Jump

Press J(ump) and the editor responds with the following subcommand prompts:

```
Start End Line Position Atag Btag Ctag Dtag
```

Choose an option to position the second @ delimiter. If you choose a tag, that tag must already be set in text. See the TAG command for information on setting tags. See the JUMP command later in this chapter for further explanation of JUMP subcommands.

2.3.6 Put

Press P(ut) to save the delimited block of text to a named output file. The editor prompts:

```
Output file: pathname
```

Enter the output *pathname* and press the <RETURN> key or the <ESC> key to copy the buffer contents. The delimited text is written to this file and also remains in its current place in memory.

The block buffer holds only one block of text. If you execute another BLOCK command before retrieving the contents of the block buffer with the GET command, the original contents are overwritten by the new block of text.

2.4 DELETE (D)

The DELETE command works like the BLOCK command. The subcommands and the manner of execution are identical. However, with the DELETE command, text can be deleted by typing D at both endpoints.

2.5 EXECUTE (E)

The E(xecute) command executes a previously defined macro. Execution means taking all console input (except responses to the ?replace and "all changes lost" prompts) from the macro text instead of from the keyboard. Press E; the editor prompts:

```
Macro name:
```

Type in the macro name followed by <ESC> or <RETURN>. If the macro exists, it is executed. If it does not exist, the message "no such macro" appears on the message line.

Execute a macro until the end of the file by preceding the E(xecute) with a forward slash (/). Execute the macro a specified number of times using the count option. For example:

/E<RETURN>	(Executes the macro forever)
[count]E<RETURN>	(Executes the macro a predefined number of times)

2.6 FIND (F)

The F(ind) command searches forward from the current cursor position for a string of characters. To find a string of characters, press F; the editor prompts:

```
Find'' ''
```

Between the double quotes (" ") is the last string requested (initially NUL). To change the target string, type in a new string. Pressing the <ESC> key before typing in a new target string repeats the last find.

NOTE

Do not use <RETURN> to execute the F(ind) command; <RETURN> becomes part of the target string to be found. Execute the command by pressing the <ESC> key.

When the target string is found, the cursor is placed immediately at the first character following the target string. If the string is not found, the message, "not found: *target string*" is displayed on the message line.

2.7 -find (-)

The -(find) command works like the F(ind) command with the following exceptions:

- -(find) searches backward from the current cursor position to the beginning of the file.
- The Showfind option is ignored. (See section 2.17, the SET command).
- The cursor is positioned on the first character of the matched string.

2.8 GET (G)

The G(et) command retrieves the contents of the block buffer (or a disk file) and places it at the current cursor position in your file.

Press G; the editor prompts

```
Input file:
```

To retrieve the contents of the buffer, press the <RETURN> key or the <ESC> key. To retrieve a disk file, specify the complete *pathname* of the file and enter <ESC> or <RETURN>. The file contents will be copied at the current cursor location.

Using the G(et) command with the BLOCK command facilitates moving or copying text.

G(et) can be used in conjunction with the O(ther) edit buffer to allow the user to move text from one file to another.

2.9 HEX (H)

The HEX command enables you to insert the ASCII equivalents of hexadecimal values in the text or display the hexadecimal values of text contents on the message line.

Press H; the editor prompts:

```
Input  Output
```

Press I(nput) to insert a hexadecimal value at the cursor position. The editor prompts:

```
Hex value:
```

Next to the colon, enter one digit or an even number of digits. If the input is legal, the equivalent characters are inserted in the text at the cursor position. If the values are not legal, the editor displays "invalid hex value" and returns to the main command level.

Press O(utput) to see the hexadecimal value of the character at the actual cursor position. If you want to see the hexadecimal values of a string of characters, precede the HEX command with the number of bytes you wish to display. Up to ten bytes of hexadecimal values can be displayed on the message line. If more bytes are to be displayed, the message line prompts:

```
hit space to continue
```

2.10 INSERT (I)

INSERT mode enables you to create initial text or add text to an existing file. To exit INSERT mode and return to the main command level, press the <ESC> key.

Press I; the editor confirms insert mode with the message:

```
[insert]
```

on the menu line. Each character you enter is then inserted at the cursor until you press the <ESC> key. The new text is displayed as you insert each character.

INSERT mode works differently if a forward slash precedes the I. All text past the cursor in the line where INSERT is specified is moved down. This avoids distracting flashing on the screen (which is caused from text getting pushed out as more text is inserted).

In INSERT mode, you can use the cursor control keys (<←>,<→>,<↑>,<↓>, <HOME>, <RUBOUT>, and <RETURN>) and the <CNTL>delete keys. Pressing <RETURN> after inserting text in the last line scrolls the text up one line.

With the editor you can insert text beyond the screen display (i.e., more than 80 characters per line) with the SET command (see section 2.17). However, only 80 characters can be viewed. The set command must be reissued to view beyond the right margin. The screen does not scroll horizontally.

If the cursor is positioned beyond the end of a line when entering text in insert mode, the cursor moves to the point immediately before the end (or carriage return) of the current line, and the insertion begins there.

<CNTL>C deletes all text inserted since the beginning of INSERT mode (or the last move command) and returns the editor to the main command level.

2.11 JUMP (J)

The JUMP command moves the cursor to a specified location in text. The editor automatically returns to main command level upon completion of the jump.

Press J; the editor responds with the following submenu.

```
Start End Line Position Atag Btag Ctag Dtag
```

2.11.1 Start

The Start subcommand moves the cursor to the start of the file.

2.11.2 End

The End subcommand moves the cursor to the end of the file.

2.11.3 Line

The Line subcommand moves the cursor to a specified line. Press L; the editor prompts:

```
line:
```

Enter any value and press the <ESC> key. The cursor jumps to the start of the designated line or to the EOF marker if the value is greater than the number of lines in the file. (The first line in a file is line 0.)

2.11.4 Position

The Position subcommand moves the cursor to a specified column position (0-79) in the current line. Press P; the editor prompts:

```
column:
```

Enter a value (0-79) and press the <ESC> key. The cursor jumps to the designated column.

- If the current line is too short, the cursor jumps to the last column in the line.
- If there is no character at the specified column (i.e., alphanumeric character, space, carriage return/line feed), the cursor jumps to the next character.

2.11.5 Atag, Btag, Ctag, Dtag

The cursor jumps to the specified tag, which you previously set with the TAG command (refer to the TAG command entry later in this chapter).

2.12 MACRO (M)

With the screen editor MACRO command, you can create a sequence of EDIT commands that you can name, store in a separate file, and have available for execution at any time. Macros are explained in greater detail in Chapter 3.

2.13 OTHER (O)

The screen editor has two distinct edit buffers; only one is available for display at a given time. The text area used at start-up is in the primary buffer; the other edit buffer is called the secondary buffer. The secondary buffer has a fixed size of 7 KB. The OTHER command enables you to switch from one edit buffer to the other.

Press O; the message "Other no input file" is displayed at the start of the message line whenever the secondary edit buffer is accessed:

```
---- Other no input file
Again Block Delete Execute Find -find Get --more--
```

You initially have a blank screen except for the the command menu on the second line from the bottom. All EDIT commands are available in both buffers. Change from one file to the other by pressing O(ther) at main command level.

The secondary buffer enables you to examine, search, or borrow a block of text from a file within the primary buffer. Because the secondary buffer has a fixed size of 7 KB, text that you have already scanned is deleted in order to display new text.

If the text size is greater than 7 KB and text is lost, "some text lost" appears on the message line. Any attempt to initialize an edit session with a file larger than 7 KB in the secondary buffer results in the message "text does not fit," being displayed on the message line. If the file is too large, it cannot be saved with the QUIT Write command. Use the main edit buffer to edit files greater than 7 KB.

2.14 QUIT (Q)

The QUIT command terminates the editing session. Press Q; the editor displays the following subcommands:

```
---- no input file
Abort Execute Init Write
```

2.14.1 Abort

The Abort subcommand returns control to the development system software without writing the contents to a file. If you have made any changes to the file, the editor will prompt:

```
all changes lost? (y or [n])
```

Any response other than Y returns the editor to the main command level.

2.14.2 Execute

The Execute subcommand enables you to redefine a debug object (for example PROC). To execute the debug object, enter its name after control has been returned to the development system software.

2.14.3 Init

The Init subcommand enables you to begin a new editing session without returning to the development system software. If any changes have been made to the current file, the editor prompts:

```
all changes lost (y or [n])
```

Any response other than Y returns the editor to the main command level.

To edit a new file, press I; the editor prompts:

```
enter [file [TO file ]]
```

Enter the *pathname* of the new file you wish to edit. The editing session is then re-started with the new file.

2.14.4 Write

The Write subcommand enables you to write the contents of your file to a disk file. Press W(rite) to save a file to a disk. Specify the complete path name of the file to be created or updated. The QUIT menu is always re-displayed after an Update (see section 2.14.5) or Write subcommand. Press A to abort from the editor; <ESC> or <CNTL>C to return to the main command level.

2.14.5 Specified Filename

If Q(uit) is pressed from an initialized file (file brought into editor with the Init subcommand) the following menu appears:

```
----Editing filename
Abort Execute eXit Init Update Write
```

In this mode, press X(eXit) to update an existing file and return to the development system software. Press U(pdate) to rewrite a file and remain in the QUIT sub-menu.

The rest of the QUIT subcommands work as previously described.

2.15 REPLACE (R)

The REPLACE command enables you to replace the target string with a new string. The REPLACE command also enables you to replace a target string with a NULL string.

Press R; the editor prompts:

```
Replace "old string"
```

Type in the string to be replaced (if other than the "*old string*") and press the <ESC> key.

NOTE

When the editor is first entered, *old string* and *new string* are initially NULL.

The editor prompts:

```
Replace "old string" with "new string"
```

Type in the new string and press the <ESC> key. The next occurrence of the old string is replaced with the new string. Note that if you do not specify a new string before you press the <ESC> key, the replacement string from the previous REPLACE command is used as the replacement string.

Designate a specific number of strings to be replaced by preceding the command with a decimal integer. For example:

```
[count]Replace "old string" <ESC> with "new string" <ESC>
```

Repeat the REPLACE command throughout the text by preceding the command with a forward slash (/).

To delete a target string from text, type a space for *new string*, press the <RUBOUT> key to erase the space, and then press the <ESC> key to execute the deletion. For example:

```
R "old string"<ESC> with "<space><RUBOUT>" <ESC>
```

2.16 ?REPLACE (?)

The ?REPLACE command, which you invoke by pressing ?, works like the REPLACE command except that you are prompted on each find:

```
ok to replace? (y or [n])
```

Press Y to replace; any other key not to replace. Once you specify not to replace, the editor continues to search and prompt until the end of the file. This command is especially useful when doing global replacements.

2.17 SET (S)

The SET command enables you to set the spacing for tabs, the viewrow (the row on which the cursor is positioned after a VIEW command; see section 2.19), and the left column so that lines over 80 characters long can be displayed.

The SET command also enables you to set or reset switches that control several options in the screen editor's environment. A switch is an option that has only two states: yes or no.

Press S; the editor displays the following subcommand options:

```
Autocr BAK file Case Indent Leftcol Notab Showfind --more--
Tabs Viewrow                                       --more--
```

2.17.1 Autocr

The Autocr subcommand prompts:

```
insert cr,lf automatically? ([y] or n)
```

Brackets [ ] indicate the current state of each option. The state is changed by specifying the alternative.

- If Y, a carriage return/line feed is inserted in the last clolumn of a line whenever an attempt is made to insert a character in that column. Trailing blanks and tabs are deleted and the end of line/line feed is inserted as a space. Words are not broken.
- If N, automatic carriage return/line feed is disabled.

2.17.2 BAK File

The BAK file subcommand determines whether or not the editor creates a back-up file. Press B; the editor prompts:

```
create .BAK files? (y or [n])
```

- If Y, the file you are editing is renamed *file*.BAK before the QUIT eXit or QUIT UPDATE replaces it.
- If N, a back-up file is *NOT* automatically created.

NOTE

This option is off by default. To guard against accidental loss of files, set BAK to Y.

2.17.3 Case

The Case option determines if uppercase or lowercase letters are significant when using the FIND or REPLACE command. Press C; the editor prompts:

```
ignore case of Find target? ([y] or n)
```

- If Y, you can type the target string in either uppercase or lowercase, or a combination of both and locate a string.
- If N, you must enter characters in upper and lower case to locate a string.

For example:

If you type "tHe" and the state is Y, the screen editor will find tHe, the, THE, etc.

If you type "tHE" and the state is N, the screen editor will find tHE only.

2.17.4 Indent

The Indent option determines if additional lines of text will be indented to correspond to the preceding line. Press I; the editor prompts:

```
automatically indent during insertion? ([y] or n)
```

- If Y, inserted carriage returns are followed automatically by the same combination of blanks and tabs as in the preceding line.
- If N, this option is turned off.

2.17.5 Leftcol

The Leftcol option enables you to view lines of text which are over 80 characters long. The command accepts any number from 0 to 79 (columns start at zero). The number indicates the column at which the left margin starts.

Press L; the editor prompts:

```
first column:
```

Enter a number between 0 and 79.

For example, if a line is 90 characters long, set the left column to 10 and the screen will display the line from column 10 to column 89.

An exclamation point (!) is printed in the leftmost column if there are no characters to be displayed.

2.17.6 Notab

The Notab option enables you to insert blanks instead of tabs whenever you press the TAB key while in INSERT mode.

Press N; the editor prompts:

```
insert blanks for tabs? (y or [n])
```

- If Y, blanks are inserted in the place of tabs when in INSERT mode.
- If N, this option is turned off.

2.17.7 Showfind

The Showfind option determines if lines of text found or replaced will be listed.

Press S; the editor prompts:

```
list lines on multiple finds?  (y or [n])
```

- If Y, when you execute a FIND or REPLACE command, all of the lines of text in which the target string is found or replaced are listed on the screen.
- If N, the FIND and REPLACE commands execute without displaying.

2.17.8 Tabs

The Tabs option enables you to determine the spacing for tabs.

Press T; the editor prompts:

```
Tabs:
```

The message line lists the current tab settings. If you only want to inspect the tab settings, type <CNTL>C to return to the main command level.

If you type in a value for tab in the range 4 to 158, tabs are set as indicated.

For example:

4 sets tabs at 4, 8, 12, 16, 20...

The default tab setting is 4.

2.17.9 Viewrow

The Viewrow option selects the line on which the cursor is positioned when you execute the VIEW command (see section 2.19).

Press V; the editor prompts:

```
row for View:
```

Type the number of the line on which you want the cursor positioned by the VIEW command. To center the cursor on the middle lines, set Viewrow to 10 or 11. Legal viewrow numbers are 1-21.

2.18 TAG (T)

The TAG command enables you to specify a location within a file. Position the cursor where you want the tag before invoking the TAG command. The cursor's current position determines the tag location. The editor returns automatically to main command level.

You can set four tags: A, B, C, and D.

Press T; the editor prompts:

```
A tag B tag C tag D tag
```

Set tag(s) A through D with the corresponding letter. An invisible tag marks the text for JUMP commands.

2.19 VIEW (V)

The VIEW command rewrites the screen leaving the cursor on the line specified with the SET (Viewrow) command. If you did not set the viewrow with the SET command, the cursor is placed on line 5 by default.

2.20 XCHANGE Mode (X)

The XCHANGE mode enables you to write over existing text, character for character. To exit XCHANGE mode, press the <ESC> key.

Press X; the editor displays the following on the menu line:

```
[exchange]
```

Move the cursor to any location in text and begin typing; characters are replaced on a one-for-one basis (except end of line/line feed, which is never replaced; instead, the line is extended).

The control delete commands, <CNTL>A, <CNTL>F, <CNTL>X, and <CNTL>Z work the same as in XCHANGE mode.

Pressing the <RUBOUT> key after an exchange replaces the character to the left of the cursor with the original character. The following are exceptions:

- If the line has been extended, <RUBOUT> deletes the character immediately to the left of the cursor.
- If the cursor is in the position where you started exchanging, <RUBOUT> moves one character to the left, but does not delete the character.

The <ESC> key returns the editor to the main command level.

Pressing <CNTL>C before <ESC> restores original text (aborts the exchange) and returns the editor to the main comand level.

Once you have have exchanged text and pressed the <ESC> key, the <RETURN> key, or any of the cursor movement arrow keys (<←>,<→>,<↑>,<↓> or <HOME>), changes cannot be undone. The <RETURN> key and the cursor movement command keys restart XCHANGE at a new location.

The maximum number of characters of original text that can be restored by pressing the <RUBOUT> key is 100. Any attempt to replace over 100 characters, without somehow restarting XCHANGE mode, will result in the following error message:

```
Xchange limit is 100
```

3

MACRO COMMANDS

3.1 Introduction

This chapter describes how to create, execute, and save macro definitions.

Edit macros are sequences of editor commands and keystrokes that have been collected and given a name.

Macros are typically used for long command sequences that are executed often. Instead of entering a series of commands, call the predefined macro to execute the commands automatically. The macro facility speeds your work and reduces typing errors associated with long command sequences. There is only 1 KB of memory available per macro definition. If a macro exceeds this limit, the message "no more room for macros" appears on the message line and the macro definition is aborted.

A set of macros can be grouped in a macro file. Macro files can consist of SET commands and macro definitions. When you bring a macro file into the editor, all the macros within the file are made available for execution in your current editing session.

SET commands, when included in a macro file, must be terminated by a semicolon (;) or carriage return. (SET commands are described in Chapter 2). When the macro is executed, the same error messages are issued as when the SET command is used.

3.2 Macro Files

When you create a macro, letters and digits are entered as is. Function keys are stored in a special format; for example, the <↑> key is displayed in its macro form, \CU.

The following is a representation of control characters and control codes:

Name	Represents
\BR	ESC
\CU	UP
\CD	DOWN
\CR	RIGHT
\CL	LEFT
\CH	HOME
\XA	DELETE RIGHT
\XF	CHAR DELETE
\XX	DELETE LEFT

Name	Represents
\XZ	CLEAR LINE
\NL	CARRIAGE RETURN
\RB	RUBOUT
\0h	Hex value of a character
\EM	End of macro definition

The backslash must appear twice (\\) if it is being used within the macro and not as a code lead.

Macros are defined with the following format:

M *macro name* \BR *macro characters* \EM { <RETURN> ; }

Where:

M	declares that a macro definition follows.
macro name	is any name given to the macro being defined.
\BR	is macro code for the ESC key.
macro characters	are any valid edit commands, letters, digits, or function keys.
\EM	signals the end of the macro.
<RETURN> or ;	signals the end of the macro definition.

3.3 Description of Macro Commands

There are two commands that pertain to macros:

MACRO	Creates macros, retrieves them from memory, and inserts them in text.
EXECUTE	Executes a specified macro.

3.3.1 Macro Command

The MACRO command enables you to define macros, retrieve them from memory, and list the names of all currently defined macros on the message line.

A macro definition is a series of commands written in macro form. Macros can be defined in two ways:

- Interactively — using the MACRO Create command.
- Directly — by writing macros to a macro file using the MACRO Insert command (this command inserts text in macro form automatically).

To save interactively defined macros, you must write them to a separate macro file in macro form. The MACRO Get command is used to get a macro file, thereby making the macros within it available for execution.

Press M; the editor prompts:

```
Create  Get  Insert  List  Save
```

If you select the Create subcommand, the message "----Macro" appears on the message line and remains there until you terminate the macro definition.

The following sections describe the macro subcommands.

3.3.1.1 Create

With the Create subcommand you can create macros interactively with a sequence of edit keystrokes. The macro is executed and created concurrently.

Press C; the editor prompts:

```
Macro name:
```

Enter a name for your macro followed by <ESC>. Macro names can be either single characters (whose names do not conflict with editor commands) or character strings up to 60 characters. The macro name may contain any characters. Single character macros are executed by typing the character; character string macros are executed with the EXECUTE command.

The entire set of editor commands is available for MACRO Create. After you type in the macro name, the editor returns to main command level, however, all subsequent keystrokes are trapped by the editor. These keystrokes make up the macro definition. Terminate the macro defintion using one of the following characters:

<CNTL>C — Terminates macro mode without defining the macro, the macro is deleted.

M(acro) — Terminates and saves the macro definition.

If macros exceed the amount of memory allocated for macros (1 KB), the message "no more room for macros" appears on the message line and the definition is aborted.

The following example shows how to interactively create a macro called "," (comma) that finds the next occurrence of the last target string:

```
MC                    (Invoke MACRO Create command)
,<ESC>                (Macro name is "," <ESC> terminates name definition)
F<ESC>                (Find next occurrence)
M                     (Terminate and save macro definition)
```

The following is an example to interactively create a macro called "." (dot) that moves the cursor eight places to the right:

```
MC                          (Invoke MACRO Create command)
.<ESC>                      (Macro name is "."; <ESC> terminates name definition)
8<->                        (Move cursor right 8 places)
M                           (Terminate and save macro definition)
```

3.3.1.2 Get

The GET subcommand retrieves a macro file and makes the macros within the file available for execution. The macros are not executed until the E(xecute) command is issued.

Press G; the editor prompts:

```
Macro file:
```

If you do not specify a filename and press the <RETURN> key, the current text is treated like a macro file.

If you do type in a filename, the named file is read as a macro file.

3.3.1.3 Insert

The Insert subcommand causes all subsequent input (including <RUBOUT>) to be inserted in the text in macro form. For example if you press ↑ in MACRO Insert, the character sequence "\CU" is inserted in the text. The macro definition can then be saved in a macro file. Insert is used to change and correct macro files.

The Insert subcommand is terminated by entering <CNTL>C.

Press I; the editor prompts:

```
Control C to stop
```

Enter the macro text. Terminate the definition by entering <CNTL>C.

The following macro example defines <CNTL>L as jump to the start of the line. Note that what you type does not execute, but is inserted in macro form.

```
MI                          (Invoke MACRO and Insert commands)
M                           (M indicates start of a macro definition)
<CNTL>L<ESC>                (Macro name is <CNTL>L; <ESC> terminates name)
<->^<->^<HOME>              (Move cursor to start of line)
/EM                         (\EM indicates end of macro)
<RETURN>                    (Move cursor to start of new line in edit buffer)
<CNTL>C                     (Terminate MACRO Insert command)
```

The following text is inserted into the edit buffer:

```
M\00C\BR\CR\CL\CH\EM
```

In MACRO Insert, all keys are entered as is. Thus, commands such as <HEX> or <ESC> do not perform a function but are inserted as their macro codes (refer to Section 3.2 for the list of macro codes).

<RETURN> is not converted to \NL (macro code for carriage return) since it is used to break macro definitions into readable lines. Therefore, you must type \NL if <RETURN> is required in the macro definition.

To make the previous macro available for execution, type:

```
MG                    (Invoke MACRO Get command)
<RETURN>              (Executes MACRO Get command, making macros
                      available for execution)
```

Typing <CNTL>L at main command level causes the cursor to be positioned at the start of the line on which the cursor is located. Note: This macro will not execute if the cursor is positioned at the start of a line or positioned over the EOF marker.

3.3.1.4 List

The List subcommand lists, on the message line, the names of all currently defined macros. If the message line is not large enough to list all the defined macros, the message "hit space" appears on the line. Press the space bar to continue the list. Any other key returns the editor to the main command level.

3.3.1.5 Save

The Save subcommand translates interactively defined macros to macro form so that they can be saved in macro files for future use. If you want to look at a macro definition, use the MACRO Save to translate and display the macro. You can then review the macro, save it to a macro file, or delete it.

Press S; the editor prompts:

```
Macro name:
```

Type in the macro name followed by <ESC> or RETURN. If the macro exists, it is placed in the text at the current cursor position in macro form.

If the macro does not exist, the message "no such macro" appears on the message line and the editor returns to main command level.

You may use the following procedure to save a new interactively created macro for future use:

1. Press O to enter the OTHER buffer.
2. Retrieve your macro file using the QUIT Init command (if you want to add the new macro to an existing macro file).

3. Specify the macro filename when prompted.
4. Insert the macro in macro form using the MACRO Save command.
5. Update the modified macro file using the QUIT Update command.

3.3.2 Execute

The EXECUTE command calls a macro by name and executes it. Normally, this command is used to execute macros whose names are longer than one character or that you want to execute a number of times using the COUNT or FOREVER option.

In macro execution, all input is taken from the macro except for answers to the following prompts:

- ?Replace: "ok to replace? (y or [n])"
- QUIT Init or QUIT Abort: "all changes lost? (y or [n])"
- BLOCK Delete: "cannot save in memory, save on disk? (y or [n])"
- "hit space to continue"
- QUIT Write or BLOCK Put: "overwrite existing file? (y or [n])"

For the preceding list of prompts, the response is taken from the console.

Press [count] E; the editor prompts:

```
Macro name:
```

Type the name of the macro followed by <ESC> or <RETURN>. If the macro exists, it is executed.

If it does not exist, the message "no such macro" appears on the message line.

3.4 Macro Termination

The macro terminates execution when:

- It has been executed the specified number of times. You can specify a COUNT ($1 \leq n \leq 65535$) or FOREVER (/) option before entering the EXECUTE command.
- An attempt is made to move forward (<→>, <↓>, <HOME>, or <RETURN>) at the end of the file.
- An attempt is made to move backwards (<←>, <↑>, or <HOME>) at the start of the file.
- A FIND, -find, REPLACE, or ?Replace command fails to find the target string.

The last three terminations return the editor to the main command level. Normal macro termination leaves the editor in the current mode. For example, if macros are nested (i.e., if a macro calls a macro), only the current macro ends.

When a command in a macro is terminated, the following occurs: Macro execution is terminated, and control is returned to the caller. If the caller is at main command level, the editor waits for the next command. If the caller is a macro, execution resumes with the caller's next command.

NOTE

Be aware when coding a macro that it is possible to create an infinite loop at execution. To exit from such a macro or to terminate any macro, enter <CNTL>C. <CNTL>C terminates all nested macros and returns the editor to the main command level.

3.5 Deleting Macros

To delete a macro from the set of available macros, use the MACRO Create command. Type the following:

```
MC                          (Invoke the MACRO Create command)
macro-name <ESC>            (specify macro name to be deleted)
<CNTL>C                     (Delete macro definition)
```

Where:

`macro-name` is the name of the macro to be deleted.

This procedure does not delete a macro from a macro file. To delete a macro from a macro file, edit the macro file like any other file.

3.6 Macro Examples

The following examples describe how to save keystrokes by creating and saving macros. The examples use both the MACRO Create and MACRO Insert subcommands. The examples assume that the edit (or Other) buffer is empty.

1. **MACRO Insert Example.** Invoke the MACRO command by typing M; the editor prompts:

   ```
   Create  GET  Insert  List  Save
   ```

   Press I; the editor displays the message:

   ```
   Control C to stop
   ```

Type in the following example, which uses the character] as the macro name; a macro that sets the left column to column 79 each time it is executed. This enables you to view lines of text longer than 80 characters. The second macro, called [, sets the left column back to column 0 each time it is executed.

```
M]\BRSL79\NL\EM
M[\BRSL0\NL\EM
```

These macros are not executed when you define them. To make them available for execution type:

```
MG                          (Invoke MACRO Get command)
<RETURN>                    (Treat text as macro file, make available for execution.)
```

2. **MACRO Create Example.** The previous macros can be defined interactively using the MACRO Create command. For example, the first macro can be created by entering the following commands:

```
MC                          (Invoke MACRO Create command)
]<ESC>                      (Macro name is ]; <ESC> terminates name definition)
SL                          (Invoke SET Leftcol command)
79                          (Set left column to position 79)
M                           (Terminate and save macro definition)
```

After you define the macro, typing] at main command level sets the left margin to column 79.

3. **Create and save a macro.** The following sequence of commands saves, in a new file named EDIT.MAC, an interactively defined macro named asterisk (*) that enables you to scroll backwards ten lines.

Create the macro:

```
MC                          (Invoke MACRO Create command)
*<ESC>                      (Macro name is "*"; <ESC> terminates name definition)
10<UP>                      (Scroll backwards 10 lines)
M                           (Terminate and save the macro definition)
```

Save the macro to a file:

```
O                           (Enter OTHER and edit buffer)
QI                          (Invoke QUIT and Init commands)
EDIT.MAC<RETURN>            (Filename is EDIT.MAC)
MS                          (Invoke MACRO and Save commands)
*<RETURN>                   (Specify macro to be saved)
QU                          (Save macro file)
```

3.7 Summary of Macro Commands

Remember the following when creating macros:

1. Use the MACRO Insert command to create macros (used when a macro file already exists or when you are creating macro files). Refer to the list of macro definitions listed in Section 3.2 for the format in which commands can be entered.
2. Use the MACRO Create command to create macros interactively. This command is useful when you are editing a text file and want to define a macro without leaving the editor. The macro definition is not displayed on the screen, but once entered, the macro is available for use by typing the macro name.
3. For ease of use, limit the macro name to one character. Then you can execute the macro by pressing only the character key.
4. Use an empty edit buffer (or empty OTHER buffer) before converting macros defined interactively (Create) into macro form. If a macro file already exists, bring the file into the edit buffer ((G)et *filename*) before appending new macros.
5. Experiment with the macro commands. Initially, they may seem complicated to define, but once you have done one or two, you will appreciate their value.

A

ERROR MESSAGES

This appendix lists the error messages the screen editor reports when a command problem is encountered.

Editing Command Errors

Message	Explanation
bad tabs	Attempt to set tabs incorrectly; e.g., 4,2. (The smallest tab increment is 4.) Editor returns to main command level.
bad Leftcol	Attempt to set incorrect left column; e.g., x or 81. (The range of settings is 0-80.) Editor returns to main command level.
bad Viewrow	Attempt to set incorrect viewrow; e.g., 45. (The range of settings is 1-21.) Editor returns to main command level.
cannot delete more than 32	Attempt to use count greater than 32 with <CNTL>F command. Editor returns to main command level.
illegal command	Attempt to enter a keystroke that is not acceptable. (Refer to menu.) Editor ignores command.
invalid hex number	Attempt to enter an invalid hex value. (You must enter a single digit or an even number of digits.) Editor returns to main command level.
no such macro	Macro is not currently in memory. Editor returns to main command level.
not found: "*target string*"	Target string not found (for FIND or REPLACE). Editor returns to main command level.

Message	Explanation
some text lost	Attempt to increase contents of secondary buffer past 7 KB. Output file changed to :BB: to avoid inadvertent loss of file with a QUIT Update command. Editor returns to main command level.
text does not fit	Attempt to edit a file larger than 7 KB in secondary buffer. Editor returns to main command level.
filename <error message supplied by host operating system>	An error occurs during a QUIT Exit, QUIT Update, GET, or BLOCK Put command. Editor returns to main command level.
Xchange limit is 100	Attempt to replace over 100 characters without restarting XCHANGE mode. Editor remains in XCHANGE mode. To restart XCHANGE mode, enter <ESC> X.

Macro File Errors

If any error is found in a macro file, one of the following messages is printed (where *nnn* is the line of the macro that contains the error) and macro file processing continues.

Message	Explanation
Error in *nnn* no more room for macros	Attempt to create a macro when the macro buffer is full. (Maximum macro size is 1 KB.) Macro definition is terminated.
Error in *nnn* bad hex value	Macro definition contains an incorrect hex value; e.g., 3G.
Error in *nnn* bad "\" code	Backslash is not followed by a valid value. (See Chapter 3 for valid values.)