

Quick Contents

Chapter 1.	Introduction and Tutorial
Chapter 2.	The Editor Basics
Chapter 3.	Editing Commands
Chapter 4.	AEDIT Invocation
Chapter 5.	Macro Commands
Chapter 6.	AEDIT Variables
Chapter 7.	Calc Command
Chapter 8.	Advanced AEDIT Usage
Chapter 9.	Configuration Commands
Appendix A.	AEDIT Command Summary
Appendix B.	AEDIT Error Messages
Appendix C.	Summary of AEDIT Variables
Appendix D.	Configuring AEDIT for Other Terminals
Appendix E.	ASCII Codes
Index	

Notational Conventions

This manual uses the following conventions:

- Computer input and output appear in this font.
- **Command names appear in this font.**



Note

Notes indicate important information.

Contents

1 Introduction and Tutorial

AEDIT Tutorial	2
Activating the Editor	2
Entering, Changing, and Deleting Text	3
Copying Text	5
Using the Other Command	5
Exiting the Editor	6

2 The Editor Basics

Keyboard	8
AEDIT Display Format	9
Prompt Line	10
Menu Prompt	10
Line-edited Prompt	12
Message Line	12
Beep Warning	13
Lines and Line Terminators	13
Printing and Nonprinting Characters	13
Tags	13
Repeat Function (Count)	14
Buffers	14

3 Editing Commands

Delete Commands and Function Keys	15
Rubout	15
Delete Character or Delch	15
Delete Left or Dell	15
Delete Right or Delr	17
Delete Line or Delli	17
Undo Command	18
Insert Mode	19
Xchange Mode	20
Find Command	21

-Find Command.....	22
Replace Command	23
?Replace Command.....	24
Tag Command.....	25
Jump Command.....	26
Block Command.....	27
Block Buffer.....	27
Delete Command.....	29
Get Command	30
View Command.....	31
Other Command	32
Again Command.....	33
Set Command	34
Hex Command.....	43
Quit Command	45
Paragraph Command	48
Window Command.....	49
Kill_wnd Command	50
!system Command.....	51
Calc Command.....	52
Execute Command	53
Macro Command.....	54

4 AEDIT Invocation

Invocation.....	55
Invocation Line Examples	56
Invocation Controls	57
Forwardonly	59
Viewonly.....	60
Recover	61
Macro	62
Macrosizes.....	63
Batch	64
Examples.....	65
Default Macro File	66
Work File.....	66
iRMX-specific Information	66

5 Macro Commands

Macro Command.....	68
Deleting Macros	72

Macros and AEDIT Variables	72
Macro Modes.....	73
Execute Command.....	74
Single-character Macros	75
Macro Files.....	76
Macro Execution After a Failure	78
Screen Display During Macro Execution	80
Text	80
Message.....	80
Prompt.....	81
Window	81
Macro Examples.....	82

6 AEDIT Variables

Global Variables.....	86
Global Numeric Variables.....	86
Global String Variables	87
Local Variables.....	89
Global Variables in Macros	91

7 Calc Command

Overview	93
Numeric Constants	95
String Constants.....	95
Operators	96
Shift/ Rotate Operators.....	97
Expression Evaluation	98
Examples	99
Errors.....	100

8 Advanced AEDIT Usage

The <i>Useful.Mac</i> File	102
Tips for Writing Macros	104

9 Configuration Commands

Introduction	109
Configuration Command Notes.....	110
Configuration Values.....	115
Delay Codes.....	115

Determining the Configuration Values.....	115
<hr/>	
A	AEDIT Command Summary
Function Keys.....	119
AEDIT Editing Commands	121
<hr/>	
B	AEDIT Error Messages
Invocation Errors	125
Editing Command Errors.....	126
CALC Command Errors.....	128
Maro File Errors	129
<hr/>	
C	Summary of AEDIT Variables.....
	131
<hr/>	
D	Configuring AEDIT for Other Terminals
Tested Configurations.....	135
DEC VT52	137
DEC VT100 and VT102.....	138
Hazeltine 1510E (with escape lead-in).....	139
Hazeltine 1510T (with tilde lead-in)	140
Lear Siegler ADM3A	140
PC.....	141
Qume QVT102.....	141
Televideo 910 PLUS	142
Televideo 925 and 950	142
Wyse 50.....	143
X-Terminal.....	143
Zentec Zepher and Cobra	144
<hr/>	
E	ASCII Codes
	145
<hr/>	
Index	149

Tables

Table 4-1. AEDIT Invocation Controls.....	57
Table 7-1. Operators' Precedence and Associativity.....	96
Table 9-1. Configuration Commands.....	111
Table 9-2. Configuration Default Values.....	116
Table A-1. Function Keys.....	119
Table A-2. AEDIT Editing Commands.....	121
Table C-1. AEDIT Variables.....	131
Table D-1. Switch Settings.....	135
Table E-1. ASCII Code List.....	145
Table E-2. ASCII Code Definition.....	147

Figures

Figure 2-1. AEDIT Display.....	9
Figure 2-2. Menu Prompt Lines.....	11

AEDIT is an interactive, screen-oriented text editor with menu-style command prompts.

AEDIT enables you to:

- Display and scroll text on the screen
- Move to any position in the text file or to any point on the screen
- Rewrite text by typing new characters over old ones
- Make insertions and deletions easily

To simplify text editing, AEDIT also provides features allowing you to:

- Find any string of characters
- Substitute one string of characters for another string
- Move or copy sections of text within a file or between files
- Create macros to execute several commands at once, thereby simplifying repetitive editing tasks
- Perform arithmetic functions
- Edit two files simultaneously
- View lines over 80 characters long
- Perform basic word processing operations

AEDIT Tutorial

This session is a short tutorial that illustrates the most basic AEDIT commands. These functions are covered:

- Activating the editor
- Entering text
- Changing text
- Deleting text
- Copying text
- Using the **other** command
- Exiting the editor

The purpose of this tutorial is to get you started, not to fully document AEDIT commands. Only a few of the most basic AEDIT commands are presented in this tutorial.

Activating the Editor

Activate AEDIT by typing:

```
AEDIT <CR>
```

The editor displays this prompt at the bottom of the screen:

```
-??- system-id AEDIT Vx.y Copyright yyyy Intel Corp.  
Again Block Calc Delete Execute Find -find --more--
```

The question marks (-??-) in front of *system-id* indicate that AEDIT is waiting for your input. When AEDIT is busy, the question marks are replaced by two exclamation points (-!!-). *System-id* is a string identifying the operating system, *x.y* is the AEDIT version number, and *yyyy* is the copyright year(s). The vertical bar (|) (initially in the upper left corner of the screen) marks the end of the file (EOF). As you type text into the file, the vertical bar moves and continues to mark the end of the file. The cursor initially covers the EOF marker.

When first invoked, AEDIT is at the main command level waiting for your input. The menu prompt line displays a selection of main commands or modes (**xchange** and **insert** are considered modes). AEDIT does not return automatically to the main command level after executing some commands (e.g., **block**). To return to the main command level or to exit the **insert** or **xchange** modes, press <Esc>.

To specify a menu selection for a command or mode, press the initial letter of the selection (for instance, B for **block**).

The word `--more--` on the prompt line indicates that there are more commands or modes. Press <Tab> to display the next line of prompts. Pressing <Tab> at the last line of prompts returns you to the first line of prompts.

Entering, Changing, and Deleting Text

Before typing text into the file, you must press **I** to enter **insert** mode. The word `[insert]` is displayed at the bottom of the screen, indicating that you are in **insert** mode. Type a word but misspell it. To correct the typing error, press the key configured to **rubout**. Each time you press the key configured to **rubout**, the cursor backs up one column and erases the character. When the erroneous character is erased, type the correct character.

The line you just typed may be deleted character-by-character with the key configured to **rubout**, or in its entirety with the key configured to **delli**, delete line (usually configured to `<Ctrl-Z>`). Delete the line. The file is now empty, and the EOF marker is back in the upper left-hand corner of the screen. The cursor, however, remains in the same position on the screen until the next command is given.

Now type this sentence, exactly as shown. Several words are deliberately misspelled.

```
High-level languages (Pacal in particular) more <CR>
closely modal the human thought process than <CR>
low-level languages such as assembly language.<CR>
```

The first word in the sentence, *levell*, is misspelled. To correct this error, use the cursor control keys to position the cursor on the erroneous *l*. Move the cursor by the cursor control keys (arrows) in the direction indicated by the arrow. Press `<Up>` twice to move the cursor to the first line. Then press `<Left>` followed by `<Home>` to move the cursor to the first position in the line. The `<Home>` key is used in conjunction with the cursor control keys for fast cursor movement. Press `<Right>` nine times to position the cursor on the first *l*. Then press the key configured to **delch**, delete character (usually `<Ctrl-F>`) to delete the extra *l*.

The *s* in *Pascal* has been omitted. To correct this error, position the cursor on the *c* in *Pacal* and type *s*. Text automatically moves to the right as the *s* is inserted.

Press `<Esc>` to leave **insert** mode and return to the main command level.

The word *model* is misspelled *modal*. To correct the error, type **x** to enter **xchange** mode. The word `[exchange]` is displayed at the bottom of the screen, indicating that you are in **xchange** mode. Position the cursor on the *a* and type *e*.

Press `<Esc>` to leave **xchange** mode and return to the main command level.

You have learned how to insert text, exchange text, and delete individual characters. Now type this sentence exactly as shown. First, press **I**. Then, position the cursor below the lines you just typed. Move the cursor to the end of the line using the right arrow key followed by `<Home>`, then press `<Down>` twice and `<CR>` once. The cursor is now positioned at the end of the file and at the beginning of an empty line. Type these lines exactly as shown:

Thus, high-level languages are easier and faster to <CR>
write than low-level languages, since one less less <CR>
translation step is required from concept to code. <CR>

Press <Esc> to leave **insert** mode and return to the main command level.

The word *less* is typed twice. To correct this error, position the cursor on the *l* of the second *less*. Because it appears at the end of the line, it may be deleted with **delr**, delete right. Press the key configured to **delr** (usually <Ctrl-A>). The **delr** command deletes all text to the right of the cursor.

Suppose you wish to delete the phrase *from concept to code* from the text, leaving the period at the end of the sentence. To do this, block (i.e., delimit) this section from the rest of the text using the **block** command followed by the **delete** subcommand.

First, position the cursor over the first character of the section. In this case you want the period to close the sentence, so position the cursor on the space before the *f* in *from*. Then press B for **block**. The @ sign covers the space. Then position the cursor one character past the end of the section you wish to block, in this case on the period immediately after the *e* in *code*. When you pressed B for **block**, the menu displayed several alternative subcommands: **buffer**, **delete**, **find**, **-find**, **jump**, and **put**.

To delete the phrase, press D for **delete**.

The phrase is deleted from the text and the space is closed automatically. The result is:

```
High-level languages (Pascal in particular) more  
closely model the human thought process than  
low-level languages such as assembly language.  
Thus, high-level languages are easier and faster to  
write than low-level languages, since one less  
translation step is required.
```

For faster cursor movement, use the (-) **find** or **jump** command. In the example above, to move the cursor to the word *human* (assuming that the current cursor position is at the end of the file), press the hyphen. You will see this prompt for a "target string":

```
-Find {Sh} ""
```

Type *human* (it will be between the quotation marks) and press <Esc>. The cursor moves to the *h* in *human*. To move forward in the file with the **find** command, press **F**, then type the target string, *human*, followed by <Esc>. Press <Esc> to terminate the **-find** command.

To jump to the beginning or the end of the file, press **J** for **jump** followed by **S** for **start** or **E** for **end**.

Copying Text

Use the **block** command to copy existing text. If you want to copy a section of text to another part of your file, delimit the text using the **block** command and press **B** at the other end to specify the **buffer** subcommand. The text is held temporarily in a buffer. Then, position the cursor where you want the text to appear and press **G**, the **get** command. This command prompts for an input file. Pressing <CR> retrieves the contents of the buffer (where the text had been held temporarily) and places it at the current cursor position.

To move a section of text to another part of your file and delete it from its present position, delimit the text at one end using the **block** command and press **D** at the other end to specify the **delete** subcommand. The text is held temporarily in a buffer. Then, position the cursor where you want the text to appear and press **G**, the **get** command. The command prompts for an input file. Press <CR> to place the buffer contents at the current cursor position. Copying or deleting a section of text is controlled by either the **buffer** or **delete** subcommand under the **block** command.

To copy a section of text to another file, delimit the text at one end using the **block** command, and press **P** at the other end to specify the **put** subcommand. The menu prompts for an output file. Type in the filename and press <Esc> or <CR>. If the specified file already exists, the message *overwrite existing file? (y or [n])* is displayed. The file is copied only if you respond with **Y**. If the specified file does not exist, it is created, and the text is copied to the specified file.

Using the Other Command

AEDIT has two distinct and equivalent files: the main file and the OTHER file. This enables you to edit two files simultaneously. To enter the OTHER file, use the **other** command. Press **O** to enter the OTHER file. Press **O** a second time to return to the main file.

See also: **other** command, Chapter 3.

Exiting the Editor

To exit from the editor, press Q for **quit**.

This prompt appears at the bottom of the screen:

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

There are three alternative subcommands: A aborts the session and w saves the file. If you do not want to save the contents of this practice session, press A to abort the session.

See also: **init** command, Chapter 3

This prompt is displayed:

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

Pressing Y returns control to the operating system without saving the file.

If you want to save this file, press w and enter a name under which the file will be saved, e.g., *myfile*.. You can now exit using the **quit abort** sequence.



The Editor Basics 2

These editor basics are described in this chapter:

- Keyboard
- AEDIT display format
- Message line
- Text area
- Beep warning
- Lines and line terminators
- Printing and nonprinting characters
- Tags
- Repeat function (count)
- Buffers

Keyboard

In AEDIT, certain keys are configured to perform functions. These function keys are enclosed in angle brackets throughout this manual. Some of these functions are also configurable.

See also: Configuration commands, Chapter 9

Arrows The four keys labeled with directional arrows are the cursor control keys <Left>, <Right>, <Up>, and <Down>.

Caps lock The Caps Lock key provides uppercase alphabetic characters.

Control <Ctrl> The Control () key changes the function of some keys on the keyboard. For example, <Ctrl-C> serves as a *soft command abort* and is a configurable key. <Ctrl-C> is recognized as soon as it is typed, even if a command is in progress.

<Esc> The <Esc> (escape) key exits modes, terminates commands, and returns the editor to main command level.

<Home> The <Home> key allows faster cursor movement. Press an arrow key followed by <Home> to page backward or forward through a file, or to move rapidly to the beginning or end of a line. <Home> is also used to enter the reedit mode for line-edit prompts.

<Return> The <Return> key moves the cursor to the beginning of the next line in **insert** and **xchange** modes and at the main command level. It also terminates the line-edit prompt except for the search commands (-) **find** and (?) **replace**.

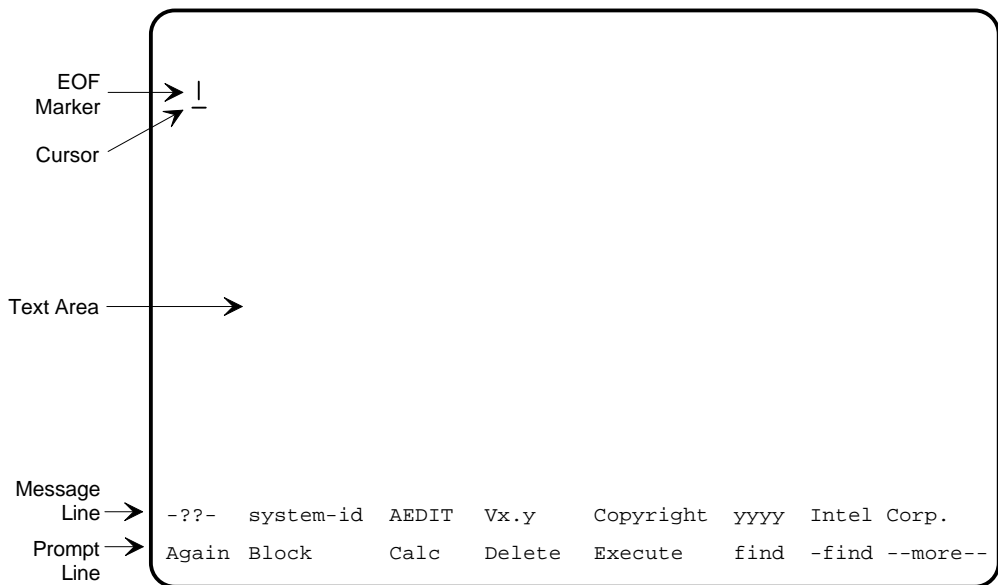
<Tab> The <Tab> key rotates the menu prompt line to display the next line of commands. In **insert** or **xchange** modes, <Tab> inserts the Tab character (or optionally, replaces it with an equivalent number of blank spaces).

AEDIT Display Format

AEDIT requires a CRT terminal (or a CRT section) with at least a 5-line, 80-column display screen (columns are numbered from 0 to 79). The screen is divided into three sections (listed from the bottom up):

- Prompt line
- Message line
- Text area

Figure 2-1 shows the screen after AEDIT is called but before any text has been typed.



W-2860

Figure 2-1. AEDIT Display

Prompt Line

The prompt line is the bottom line of the display. The first position of the prompt line is blank. The prompt line contains information on the options of commands or subcommands. The two types of prompts are menu prompts and line-edited prompts.

Menu Prompt

When you start AEDIT, the editor is at the main command level and the menu prompt is displayed. Menu prompts are a partial list of up to eight words indicating available commands. Pressing <Tab> displays the next line of prompts. Figure 2-2 shows the four prompt lines available at the main command level.

To select the desired command, type the first character of the prompt word. Uppercase letters are used in Figure 2-1, but you can type the letter in either uppercase or lowercase. The prompt for a command does not have to be visible to invoke it.

```
-??-  
Again   Block   Calc     Delete   Execute  Find    -find  --more--  
  
-??-  
Get     Hex     Insert   Jump     Kill_wnd Macro   Other  --more--  
  
-??-  
Paragraph  Quit   Replace  ?replace  set    Tag    View  --more-  
  
-??-  
Window Xchange                               --more--
```

Figure 2-2. Menu Prompt Lines

Line-edited Prompt

Line-edited prompts ask for information (such as a filename) that requires more than a single-character response. The response can be up to 60 characters. It is terminated and the information is sent by pressing <Esc>.

The prompt line always contains the parameters of the last command entered. To edit this previously entered information, press <Home> to enter reedit mode. Press <Esc> again when you are finished editing the parameters.

To enter a character using its ASCII value in the line-edit prompt, type `hex` (usually configured as <Ctrl-R>) followed by two hexadecimal digits. For example, `hex41` enters an `A`. This option enables you to enter control characters (such as `ESCAPE(1BH)`) into the text.

Entering <Ctrl-C> returns control to the main command level, leaving the original edited string unchanged.

Message Line

The message line is directly above the prompt line. Status messages indicate the command mode.

- ??- The feature is on and AEDIT expects input.
- !!- The feature is on and AEDIT is executing a command.
- - - - The feature has been turned off with an AEDIT configuration command; or that the feature is on, but the message line is for the nonactive window.

Following the busy/waiting indicator, one or more of these status words may appear:

- Macro Indicates that a macro is being defined.
- Other Indicates that the **OTHER** file is being edited.
- View Indicates that the **viewonly** control is in effect.
- Forward Indicates that the **forwardonly** control is in effect.

This part of the message line does not change unless the **other**, **macro**, or **editing** mode of the file is changed. Other messages displayed on the message line are status messages, count (repeat function), and the line-edit prompt <Home> to re-edit.

AEDIT does not write past the last column of the message line. If a message does not fit, `!` is printed as the last character.

Beep Warning

The editor beeps when you try to do something illegal, for example:

- Attempting to execute an illegal command
- Typing an invalid character during **insert** or **xchange** mode
- Typing more than 60 characters in a line-edited prompt
- Entering a repeat count greater than the maximum value

The editor also beeps when displaying error messages.

Lines and Line Terminators

A line of text consists of a sequence of characters terminated by a carriage return and line feed pair. This pair, called the line terminator, is entered in the file when you press <CR>, and it is displayed on the screen as a blank at the end of the line.

If a line is over 80 characters long, an exclamation point (!) is displayed in the last column on the screen. The portion of the line that does not fit on the screen is not displayed. To view the portion that is not displayed, use the **set leftcol** command.

See also: **set leftcol** command, Chapter 3

A line may contain any number of characters. AEDIT breaks lines longer than 255 characters into 255-character segments. A plus sign (+) is displayed at the end of each segment.

Printing and Nonprinting Characters

In general, all characters except those with ASCII values under 20H and characters with hexadecimal values equal to or above 7FH are displayed on the screen. All characters that are not displayed on the screen print as a question mark (?). The line terminator and tab print as blanks. If the **highbit** feature (described in Chapter 3) is set, characters with hexadecimal values over 7FH are displayed as-is.

See also: **highbit** feature, Chapter 3

Tags

Tags identify locations in a file. You can specify four locations, A through D, with the **tag** command and use them as destinations for the **jump** command. Tags are invisible and are not saved when you exit the file.

Repeat Function (Count)

`count` is displayed on the message line and indicates the number of times to repeat a command. Some commands ignore `count` or, like **delch** (delete character) limit `count`. Enter `count` before typing a command letter. It is then displayed at the left side of the message line. Use **Rubout** to delete the value being entered for `count`. The cursor position after a command has been executed `count` times is its location when `count` is exhausted or no more occurrences are found. When the message line contains a `count`, the `count` is blanked when the next prompt is issued. The repeat `count` is an optional decimal repetition factor in the range 0 to 65535 ($[2^{**} 16]-1$). Any attempt to type a larger value for `count` causes AEDIT to beep. A forward slash (/) is accepted as a `count` and means repeat forever. The default `count` is one.

Buffers

AEDIT has three buffers: the main buffer, the OTHER buffer, and the **block** buffer. All three buffers are allocated space in the user's free RAM.

The main buffer is the text area at startup. It always contains a portion of the main file.

The OTHER buffer is accessed with the **other** command and always contains a portion of the OTHER file (if one exists).

The buffer that is accessed and active is referred to as the current buffer; the one that is not being edited is called the secondary buffer. For example, if you are editing a file in the OTHER buffer, it would be referred to as the current buffer, and the main buffer would be referred to as the secondary buffer.

If either the main or OTHER buffer is too small for the text file, AEDIT extends the buffer with additional free RAM if it is available. When all free RAM is exhausted, AEDIT writes to temporary files, usually on disk or diskette. AEDIT's performance improves with the amount of free RAM available.

The **block** buffer is the storage area for text that you move, copy, or delete, using the **block/delete** commands. The **block** buffer allows you to move text between the main and the OTHER file. The **block** buffer has a fixed size of 2K bytes. If more than 2K bytes are required, AEDIT uses a temporary file.



Editing Commands 3

This chapter describes all AEDIT commands.

Delete Commands and Function Keys

Each of these **delete** commands are configurable to a key: **rubout**, **delch**, **dell**, **delr**, **delli**. Each performs a specific delete function.

Rubout

The **rubout** command deletes the preceding character, including a carriage return if present.

There is no recovery from this deletion.

Count: **rubout** ignores `count`.

See also: **count** command, Chapter 2

Delete Character or Delch

The **delete character** or **delch** command is configurable, usually as `<Ctrl-F>`. It deletes the character that the cursor is on, including a following carriage return if present.

There is no recovery from this deletion.

Count: This command limits `count` to 32 to prevent accidental destruction of the file. If `count` is greater than 32, the message cannot delete more than 32 is displayed on the message line.

Delete Left or Dell

The **delete left** or **dell** command is configurable, usually as `<Ctrl-X>`. It deletes all characters to the left of the cursor on the line on which the cursor is positioned.

The deletion can be recovered with the **undo** command.

Count: the **delete left** command ignores `count`.

Related Commands: **undo**

Delete Right or Delr

The **delete right** command is configurable, usually as <Ctrl-A>. It deletes all characters to the right of the cursor on the line, excluding the carriage return.

The deletion can be recovered with the **undo** command.

Count: the **delete right** command ignores `count`.

Related Commands: **undo**

Delete Line or Delli

The **delete line** or **delli** command is configurable, usually as <Ctrl-Z>. It deletes the entire line on which the cursor is positioned. All lines below the deleted line move up one row. The cursor is left in the same position on the new line.

The deletion can be recovered with the **undo** command.

Count: the **delete line** command ignores `count`.

Related Commands: **undo**

Undo Command

The **undo** command restores characters deleted by the last **delete left**, **delete right**, or **delete line** command at the current cursor position. If the previous command was **delete line**, the cursor moves to the beginning of the current line before the restoration. Consecutive **undo** commands repeat the restoration of the same string.

The **undo** command is configurable. On many terminals, <Ctrl-U> is the typical default. However, AEDIT relies on the Terminal Support Code (TSC), and TSC interprets a <Ctrl-U> as a command to empty the type-ahead buffer. Thus AEDIT never receives a <Ctrl-U> command.

We suggest using <Ctrl-Y> as the Aedit **undo** command. To configure this, edit the *:config:termcap* file. At the end of the configuration for your terminal type, add this configuration code:

```
AFXU = 19;
```

See also: Configuration commands, Chapter 9

Count: the **undo** command ignores `count`.

Related Commands: `dell`, `delr`, `delli`

Insert Mode

Insert mode enables you to enter text. To enter **insert** mode, press `I`. To exit **insert** mode and return to the main command level, press `<Esc>`.

Description

Press `I`; AEDIT prompts:

```
[insert]
```

The prompt `[insert]` is displayed whenever AEDIT is in **insert** mode. Move the cursor to any location in the text and begin typing; the characters are inserted into the text.

`<Esc>` causes the editor to leave **insert** mode and return to the main command level.

`<CR>` inserts a carriage return and moves the cursor to the beginning of the next line.

`<Ctrl-C>` deletes all text inserted since the beginning of **insert** mode, or since **insert** mode was restarted by one of cursor movement commands or delete commands but does not restore characters deleted with **rubout**, **delch**, **delli**, **dell**, or **delt**. After restoration, `<Ctrl-C>` returns the editor to the main command level.

In **insert** mode, macro execution usually restarts the insert process. The only exceptions are non-modeless macros that contain non-restarting commands only.

See also: Macro modes, Chapter 5

Insert mode is modified if it is preceded by a forward slash (`/`). All text past the cursor in the current line is moved down one line. The text is restored before any delete or move subcommand (except **rubout**) or when insertion is complete.

Count: repeat `count` is not a valid option in **insert** mode.

Related Commands: `set indent`, `set autonl`, `hex`, `mexec`, `fetn`, `fets`

Xchange Mode

Xchange mode enables you to overwrite existing text. To enter **xchange**, press **x**. To exit **xchange** mode and return to the main command level, press **<Esc>**.

Description

Press **x**; AEDIT prompts:

[exchange]

The prompt [exchange] is displayed whenever AEDIT is in **xchange** mode. Move the cursor to any location in the text and begin typing; characters are replaced on a one-for-one basis. The carriage return is not replaced; instead, the line is extended.

<Esc> causes the editor to leave **xchange** mode and return to the main command level.

Rubout works as normal except if the cursor is at the original replacement location, **rubout** then moves one character to the left but does not delete the character.

<Ctrl-C> restores original text (text before it was exchanged), however, once you have exchanged text and pressed **<Esc>** or restarted **xchange** with any of the cursor movement commands, changes cannot be revoked by pressing **<Ctrl-C>**.

Count: repeat count is not a valid option in **xchange** mode.

Error: `xchange limit is 100` is displayed if you attempt to exchange over 100 characters without restarting **xchange** mode. **Xchange** mode has a limit of 100 characters.

Find Command

The **find** command searches forward from the current cursor position to the end of the file for a string of characters.

Description

Press F; AEDIT prompts:

```
---- <HOME> to re-edit
Find {mode} "target_string"
```

The last `target_string` (if any) is displayed within quotes. Mode refers to the **set** options currently in effect that may influence the **find** command.

Pressing <CR> when specifying a target string inserts a carriage return into the target string and adds the carriage return symbol, <n1>, to the prompt line. You must press <Esc> to complete the string specification and execute the **find** command.

The cursor is placed immediately after the next occurrence of the target string. If the string is not found, the message `not found: "target_string"` is displayed in the message line and the **find** command is marked as failed.

The message `found: (number)` is displayed when the command is complete. Number refers to the number of found strings.

These attributes affect how the **find** command works. Select them through the `mode` value:

- **Case** - mode value Cs
- **Showfind** - mode value Sh
- **K_token** - mode value Tk

Case, **showfind**, and **k_token** refer to features set through the **set** command.

See also: **set** command, described later in this chapter

Count: the **find** command accepts any `count` where `count` indicates the number of times to search for a target string. The search stops after the last occurrence of the target string is found or `count` is exhausted.

Error: `not found: "target_string"` is displayed if no match is found, and the editor returns to the main command level.

Related Commands: **set case**, **set k_token**, **set showfind**

-Find Command

The **-find** command is identical to the **find** command with these exceptions:

- **-Find** searches backward from the current cursor position to the beginning of the file.
- The **showfind** option is ignored.
- The cursor is positioned on the first character of the matched string.

Description

Press the hyphen (-); AEDIT prompts:

```
---- <HOME> to re-edit  
-find {mode} "target_string"
```

The last `target_string` (if any) is displayed within the quotation marks.

Related Commands: `set case`, `set k_token`

Replace Command

The **replace** command is similar to the **find** command except that it enables you to replace the old target string with a new string. The **replace** command also enables you to delete a target string.

Description

Press R; AEDIT prompts:

```
---- <HOME> to re-edit
Replace {mode} "target_string"
```

The last `target string` (if any) is displayed within the quotation marks.

The prompt line contains two line-edited arguments. The first argument is the string to search for. After entering this string press <Esc> to enter the replacement string. This prompt will appear:

```
---- <HOME> to re-edit
Replace {mode}"target_string" with "replacement_string"
```

Pressing <Esc> finishes editing and starts the replacement process.

These attributes affect how the **replace** command works. Select them through the mode value:

- **Case** - mode value Cs
- **Showfind** - mode value Sh
- **K_token** - mode value Tk

Case, **showfind**, and **k_token** refer to features set through the **set** command.

See also: **set** command, described later in this chapter

To abort a **replace** command, press <Ctrl-C>.

The **(?)replace** and **(-)find** commands share the same target string and each changes the other's default target.

Count: the **replace** command accepts any `count` where `count` indicates the number of times to replace a target string. Replacement stops when there are no more target strings.

Error: `not found: "target_string"` is displayed if no match is found, and the editor returns to the main command level.

Related Commands: **set case**, **set k_token**, **set showfind**

?Replace Command

The **?replace** command is the conditional **replace** command.

Description

Press ?; AEDIT prompts:

```
---- <HOME> to re-edit  
?Replace {mode} "target_string"
```

The **?replace** works exactly the same as the **replace** command except that this prompt is displayed on each find:

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

If y (or Y) is typed, the replacement is made. Any other key is considered a negative response.

Tag Command

Use the **tag** command to specify four locations in a file and, with the **jump** command, move the cursor to one of these locations. The **tag** command relates to the current cursor position. Tags are invisible and are not saved when you exit the file. After the tag is set, the editor automatically returns to the main command level.

Description

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

The cursor's current position determines the tag location.

Press T; AEDIT prompts:

```
      A_tag   B_tag   C_tag   D_tag
```

Set the tag by pressing A, B, C, or D.

If the section containing the tag is deleted, AEDIT moves the tag to the first position after the deleted section.

Each input file (main or OTHER) has its own set of tags.

Count: the **tag** command ignores count.

Error: `invalid command` is displayed if a key other than A, B, C, or D is pressed.

Related Commands: **jump**

Jump Command

The **jump** command moves the cursor to a specified location in text. The editor automatically returns to the main command level.

Description

Press J; AEDIT prompts:

```
A_tag B_tag C_tag D_tag Start End Line Position
```

Count: the **jump** command ignores count.

A_tag, B_tag, C_tag, D_tag

Execute the **tag** subcommand by pressing A, B, C, or D. The cursor jumps to the specified tag, previously set with the **tag** command.

Error: no such tag is displayed if the specified tag does not exist.

Related Commands: **tag**

S-Start

The **start** subcommand, executed by pressing S, moves the cursor to the start of the file.

E-End

The **end** subcommand, executed by pressing E, moves the cursor to the end of the file.

L-Line

The **line** subcommand, executed by pressing L, prompts for a line number to jump to.

Error: illegal value is displayed if a value larger than the maximum value or any other illegal value is entered.

P-Position

The **position** subcommand, executed by pressing P, prompts for a column position to jump to.

Error: illegal value is displayed if a value larger than the maximum value or any other illegal value is entered.

Block Command

Invoke the **block** command by pressing B (or D; typing either B or D initially is equivalent). The **block** command enables you to select a section of text to delete, move, or copy. The **buffer** subcommand copies the section to the **block** buffer. The **delete** subcommand deletes the section and places it in the **block** buffer. The **put** subcommand copies the section to an external file.

Use the **get** command to retrieve text saved in the **block** buffer (or in an external file). The **get** command copies the contents of the **block** buffer (or external file) at the current cursor position in your file.

Block Buffer

The **block** buffer has a fixed maximum size of 2K bytes. If text copied to the **block** buffer is over 2K bytes, the remainder is written to a temporary work file. The contents of the buffer remain unchanged until you execute another **block** or **delete** command, when a new section of text overwrites the old contents in the buffer.

Description

To mark a section of text, first invoke the block command by pressing B. This prompt appears:

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

Move the cursor to the first character of the section and press B to mark the beginning of the block; the @ sign marks the start. Then move the cursor to the end of the section and press B to mark the end of the block. The section is copied into the **block** buffer and the @ markers are removed. To delete a block of text, use D to mark the start and end points of the text to be deleted.

To copy the block from the buffer, move the cursor to the target location. Press G and the press <CR> at the prompt for the input file. The block will be copied to the new location.

Count: the **block** command ignores `count`.

Related Commands: `get`

B-Buffer

To execute the **buffer** subcommand, press **B**. It copies text to the **block** buffer. The **@** signs are removed; the delimited section of text is copied to the **block** buffer. The delimited section of text is not affected.

D-Delete

To execute the **delete** subcommand, press **D**. It deletes the delimited section from the text and moves it to the **block** buffer. If the deleted text does not fit in the portion of the **block** buffer that is in memory, the menu prompts:

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

If **n** is specified, the delimited section of text is deleted, but the **block** buffer is not updated. If you press any other key, the delimited section is written to a temporary file. Press **<Ctrl-C>** to abort the command.

F-Find

To execute the **find** subcommand, press **F**. It works the same as it does at the main command level.

(-) -Find

To execute the **-find** subcommand, press the hyphen (**-**). It works the same as it does at the main command level.

J-Jump

To execute the **jump** subcommand, press **J**. It works the same as it does at the main command level.

P-Put

Use the **put** subcommand to copy a section of text to a named output file.

Press **P**; AEDIT prompts:

```
---- <HOME> to re-edit  
Output file: filename
```

The filename of the previous command (if any) is displayed to the right of the colon. You can copy the marked section of text to this file or a different filename may be specified. The marked section is not affected.

The specified file can also be written to an output device supported by your system.

Delete Command

Invoke the **delete** command by pressing `D`. The **delete** command enables deletion of a section of text by typing `D` at both endpoints.

See also: **block** command, described in this chapter

Get Command

The **get** command retrieves the contents of the **block** buffer or an external file and inserts it at the current cursor position in your file.

Description

Move the cursor to the point in your file where you want the contents of the buffer (or external file) to be placed.

Press **G**; AEDIT prompts:

```
---- <HOME> to re-edit  
Input file:  filename
```

The filename previously specified for the **get** command (if any) is displayed to the right of the colon.

To insert the contents of the **block** buffer at the current cursor location, press **<CR>**. To insert the contents of an external file, type the name of the file, then press **<Esc>** or **<CR>**.

The editor returns to the main command level with the cursor on the first inserted character.

Count: this command accepts any number less than 64K. The named file is copied to the current location *count* times. The repeat *count /* is not valid with the **get** command. If **/G** is typed, AEDIT returns to the main command level without issuing an error message.

Related Commands: **block**

View Command

To execute the **view** command, press `v`. This command rewrites (move) the text on the screen so that the row containing the cursor (the **viewrow**) is positioned on the row that you have defined. **View** is also useful to refresh the screen image. The **viewrow** is set with the **set viewrow** command; the default is $R/5$, where R is the number of rows in the screen.

View also issues an abbreviated sign-on message, which includes the busy/ waiting indicator, system-id, and AEDIT version number.

Count: the **view** command ignores `count`.

Related Commands: **set viewrow**, **window**

Other Command

AEDIT has two distinct buffers. The text area at startup is called the main buffer; the other is called the OTHER buffer. The **other** buffer enables you to edit a second file in the same way as the main buffer. To execute the **other** command, press `O`. Use this command to switch from editing text in one buffer, the current buffer, to editing text in the other or secondary buffer.

Pressing `O` a second time returns the editor to the main buffer.

The main and OTHER buffer text may be displayed simultaneously by splitting the screen into two windows using the **window** command. Typing `O` displays the OTHER file in one window. In this case, switching from one window to the other results in switching from one text buffer to the other.

Description

Press `O`; the message `Other Editing input file` or, if no input file has been specified, `Other no input file` is displayed at the start of the message line whenever the secondary text is displayed:

```
---- Other Editing input file
Again Block Calc Delete Execute Find -find --more--
```

Each buffer has a separate set of tags. A **jump** command is valid only within its own buffer; it cannot jump to the other buffer. Also, each buffer has its own value for **set leftcol**. The **block** buffer is common to allow moving text between buffers.

Press `O` to exit the **other** buffer and return to the main buffer.

Count: `count` has no meaning for the **other** command.

Again Command

To execute the **again** command, press A. It causes the last command, or in some cases the last subcommand, to be repeated.

In these commands, **again** repeats the entire command, including subcommand arguments:

- **paragraph**
- **(-)find, (?)replace**

In this command, **again** repeats the entire command including subcommands, but without its arguments:

- **hex** command

Count: `count` is the `count` for the repeated command. The value of `count` given for the last command is ignored.

Set Command

The **set** command enables you to set/reset several features that determine how AEDIT will operate, e.g., if case (upper case, lower case) should be considered in the target string of a search command.

Most **set** subcommands relate to switches. A switch is an option that has only two states: yes or no. When a subcommand of this type is activated, a yes/no question is displayed on the prompt line. The value currently in effect is enclosed in square brackets. Each feature has a default value; this value is in effect until it is reset using the **set** command.

Description

Press S (press <Tab> to view the remaining prompt lines); AEDIT prompts:

```
Autonl  Bak_file  Case  Display  E_delimit  Go  Highbit  --more--
Indent  K_token   Leftcol  Margin   Notab   Radix  Showfind  --more--
Tabs    Viewrow                                --more--
```

To specify an option, press the initial (upper case) letter of that option.

Count: the **set** command ignores count.

A-Autonl

This option automatically creates a new line at the right margin, in **insert** mode. AEDIT prompts:

```
insert <nl> automatically? (y or [n])
```

- If **y**, a carriage return is inserted in the right margin whenever an attempt is made to insert a character in that position. If the character to be inserted is not a white space (i.e., not a space, tab, carriage return, or line feed), the carriage return is inserted before the token, if possible. Trailing blanks and tabs are deleted, and the carriage return is inserted between words. The right margin is set using **set margin** (described later).
- If **n** (the default), the option is turned off.

Related Commands: **insert**, **set margin**

Set Command

B-Bak_file

Under this option, AEDIT saves a backup file that contains the last version of your file. AEDIT prompts:

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

- If **y** (the default), the file you are editing is renamed *file.bak* when **quit exit** or **quit update** is executed, before the edited text is written.
- If **n**, this option is turned off.

⇒ **Note**

Keep this option turned on. If your file is accidentally lost or damaged and **bak_file** is **yes**, the previous version of the file would be saved in the backup file.

C-Case

Under this option, AEDIT uses case as a criteria when searching for a target string in a search command. AEDIT prompts:

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

- If **y**, you can type the target string in upper case, lower case, or a combination of both, and the case is significant.
- If **n**, (the default), you can type the target string in upper case, lower case, or a combination of both, but the case is ignored.

Related Commands: (-)find, (?)replace

D-Display

Use this option to display the text changes resulting from macro execution. **Set e_delimit** is used by the **find/replace** commands under token mode.

AEDIT prompts:

```
display macro execution? (y or [n])
```

- If **y**, during macro execution all cursor movements and text changes are displayed on the screen.
- If **n** (the default), this option is turned off. Thus, when macro execution starts, cursor movements or changes in the text outside the current screen are not displayed on the screen.

E-E_delimit

You can display and change the current delimiter set using this option. **Set e_delimit** is used by the **find/replace** commands under token mode.

AEDIT prompts:

```
---- <HOME> to re-edit
delimiter set: current delimiter set
```

All characters currently specified as delimiters are displayed to the right of the colon on the prompt line.

Delimiters have these properties:

- A delimiter is always one character.
- Characters with hexadecimal values from 00H-20H, and 7FH or more are predefined delimiters. They are not displayed, and they cannot be excluded from the delimiter set.
- ASCII characters with the values 21H-7EH are displayed (if specified).
- Delimiters are displayed with no separating characters.

Delimiters specified by **set e_delimit** are used to define a token for the **(-)find/(?)replace** operation. A token is any nonempty string surrounded by delimiters.

When you specify a set of delimiters, you may include the same delimiter more than once. For example, you may separate input delimiters with blanks.

The default **e_delimit** string is:

```
! " # % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ ` { | } ~
```

Related Commands: **(-)find, (?)replace, set k_token**

Set Command

G-Go

This option relates to macro execution continuation after a **(-)find/(?)replace** command failed. AEDIT prompts:

```
continue macro execution after a failure? (y or [n])
```

This option is meaningful only in macro execution; it has no meaning at the main command level. The default for the **set go** option, when a macro is started, is **no** for all macros regardless of the current setting of the option. To use this option, specify **set go yes** in your macro. It may be reset within the same macro. The **set go** option affects only the current macro and not the enclosing or enclosed macros.

- If **y** is in effect for the current macro and a **(-)find/(?)replace** command fails, execution of the current macro continues, i.e., the next command is activated.
- If **n** (the default), is in effect for the current macro, and a **(-)find/(?)replace** command fails, execution of the current macro is terminated, and control is returned to the caller, either a macro or the main command level.

During **macro create**, the **set go** command is inserted into the macro definition, but the macro currently defined is executed as if **set go** is **yes**.

Related Commands: **macro create, execute**

H-Highbit

Under this option, AEDIT displays characters with hexadecimal values over 7FH. AEDIT prompts:

```
display parity-on characters as is? (y or [n])
```

- If **y**, all text characters with hexadecimal values over 7FH are written to the screen as-is.
- If **n** (the default), all text characters with hexadecimal values over 7FH are displayed as **?**.

I-Indent

Use this option when entering code in a structured language such as PL/M or Pascal. AEDIT prompts:

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

- If `y` is chosen, when `<CR>` is pressed in **insert** mode, the next line is automatically indented to as the preceding line. When `<CR>` is pressed at the main command level, the cursor moves to the first nonblank, nontab character in the next line.
- If `n` (the default) is chosen, this option is turned off.

This option is not active in **xchange** mode.

Related Commands: `<CR>` (`<Return>`), **insert**

K-K_token

⇒ Note

K-tokens are not the same as iRMX tokens.

This option enables you to find a string only if it is enclosed by delimiters and is not part of a larger string. AEDIT prompts:

```
find only token strings? (y or [n])
```

Token characters are all the characters that are not delimiters. A token is defined as a nonempty string surrounded by delimiters. Delimiters in this context are the characters specified in **set e_delimit**.

- If `y`, a string is found by the **(-)find** or **(?)replace** command only if the string fits the token definition.

A string in the text that is found by **(-)find** or **(?)replace** commands when **set k_token no** is in effect will also be found when **set k_token yes** is in effect only if that string is a nonempty string surrounded by delimiters. In general, delimiters include the beginning and end of a file, the cursor position, and carriage return.

- If `n` (the default) is pressed, a string in the text is found regardless of the characters that surround it.

Related Commands: **(-)find**, **(?)replace**, **set e_delimit**

Set Command

L-Leftcol

Use this option to view lines over 80 characters long on the screen.

AEDIT prompts:

```
left column: current_left_column
```

The current left column is displayed to the right of the colon. This command accepts any number from 0 to 175 (position count starts at 0). The number input indicates the number of characters at the start of a line that should not be displayed.

For example, if a line is 90 characters long, you can set **leftcol** to 20 and the screen will display the line from position 21 to the end of the line.

An exclamation point (!) is printed in column 0 when characters to the left are not displayed.

Leftcol can also be set by typing the plus sign (+) or the minus sign (-) followed by a valid decimal number. This sets the left column at the current value plus or minus the number given.

AEDIT may have two different values for **leftcol** simultaneously, one for the main file and one for the OTHER file. The default **leftcol** for both files is zero.

For example, if the left column is currently set at position 15 and you type `S(et) L(eftcol) -10`, the new left column is position 5.

Error: `bad Leftcol` is displayed if you attempt to set a value out of range.

M-Margin

This option sets values for indenting left and right margins for reformatting a paragraph. AEDIT prompts:

```
---- <HOME> to re-edit  
indent, left, right: current indent, left and right margins
```

The current values for indent, left, and right are displayed to the right of the colon separated by commas. The first number sets the indentation, the second the left margin, and the third the right margin. Indent may be set at any value from 0-253; left, from 0-253; and right, from 1-254. The value of the right margin must be greater than the indentation and the left margin. When entering the values, separate them by one or more blanks or a comma. The values of all three numbers are absolute and offset from position 0. The default values are 4, 0, and 76.

Press `<Esc>` or `<CR>` to execute the command and return to the main command level.

To set indent to 0, left to 5, and right to 70, type:

```
S(ET) M(argin) 0, 5, 70 <CR>
```

To reset the left margin to 2, type:

```
S(ET) M(argin), 2 <CR>
```

Related Commands: `paragraph`, `set autonl`

N-Notab

This option instructs the editor to replace inserted tabs with the appropriate number of blanks. AEDIT prompts:

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

- If `y`, blanks are inserted instead of tabs whenever you press the `<Tab>` key in the **insert** or **xchange** mode.
- If `n` (the default), this option is turned off.

This option does not affect tabs that are entered using the `<hex>` prefix (for instance, `<HEX>09`).

Related Commands: `set tab`

R-Radix

This option determines the radix (base) in which an AEDIT numeric variable will be inserted in the text. AEDIT prompts:

```
---- current Radix: current radix  
Alpha      Binary      Decimal      Hex      Octal
```

This option affects values inserted or exchanged by the **fetch** operation. The **radix** default is decimal.

See also: **radix** operation, Chapter 6

Related Commands: `insert`, `xchange`

S-Showfind

Use this option to display all lines containing the target string in a **find/replace** command. AEDIT prompts:

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

- If **y**, when you execute a **find** or **replace** command and **count** is greater than 1, the screen is cleared, and each text line that contains the target string is displayed on the screen.
- If **n** (the default), the **find** and **replace** commands execute as usual, but the screen is not cleared, and text lines that contain the string are not displayed on the screen.

Related Commands: **find**, **replace**

T-Tabs

This option sets tabs. AEDIT prompts:

```
---- <HOME> to re-edit  
Tabs: current tab setting
```

The prompt line lists the current tab settings. If you want to inspect the tab settings, type **<Ctrl-C>** to return to the main command level.

To enter tabs, type a list of decimal numbers separated by at least one blank or a comma. Specify the numbers in increasing order, from 1-253. Changing the tab settings does not change the file contents, but it may affect its display on the screen.

The default tab settings are every fourth position, i.e., 4, 8, 12, and so on.

The difference between the last two numbers specified for tabs is repeated, up to 253.

For example:

```
4          sets tabs at 4, 8, 12, 16,...  
5, 6, 10  sets tabs at 5, 6, 10, 14, 18,...
```

⇒ **Note**

Columns start at 0, not 1. Therefore, FORTRAN tabs should be 6,10, not 7,11.

Error: bad tabs is displayed if you attempt to set an illegal tab.

Related Commands: **insert**, **xchange**

V-Viewrow

This option selects a view row for rewriting the screen. AEDIT prompts:

```
row for View:  current viewrow
```

The current **viewrow** setting is displayed to the right of the colon. Type the number of the row on which you wish the cursor to be positioned by the **view** command.

This value must be between 0 and the text size -1. If, for example, your screen size is 25 rows, then text size is 23 (25 minus the message line and the prompt line).

Therefore, the legal values are 0-22.

The default **viewrow** setting is the number of rows in the screen, divided by 5, (which means 5 on most terminals).

If the screen is split using the **window** command, the **viewrow** is determined separately for each window.

Error: bad Viewrow is displayed if viewrow is illegal.

Related Commands: **view**, **window**

Hex Command

The **hex** command inserts the ASCII equivalents of hexadecimal values in the text. This command also displays the hexadecimal values of text contents in the message line.

Description

Press H; AEDIT prompts:

```
Input      Output
```

To specify a subcommand, press the initial letter of that subcommand.

I-Input

Press I; AEDIT prompts:

```
---- <HOME> to re-edit  
Hex value:
```

The last values entered for **hex** input are displayed to the right of the colon. Legal input values consist of one or more strings separated by one or more blanks. A legal string has these characteristics:

- Every character is a valid hexadecimal digit (0-9, A-F)
- Contains 1 or an even number of characters

Values entered are regarded as hexadecimal; therefore, the suffix H is an illegal character. The values may be separated by one or more blanks.

These are examples of legal input values:

```
9  
5A5B60  
3456 65      78F0 8      98C8A7
```

If the input is legal, the equivalent characters are inserted in the text at the cursor position.

Error: `invalid hex value` is displayed if an illegal value is entered, and the editor returns to the main command level.

O-Output

Press `o`.

The hexadecimal value of the character immediately to the right of the actual cursor position is displayed on the message line. The *count* that preceded the **hex** command gives the number of bytes in hexadecimal format. Up to 10 bytes of hexadecimal values can be displayed in the message line. If more bytes need displaying, the message `hit space to continue` is displayed.

Press the space bar to display the next 10 bytes. Any other key returns AEDIT to the main command level.

Examples

To insert the form-feed character (hexadecimal value `0C`) to the current location, type:

```
H(EX) I(nput) C <CR>
```

To insert the digits 1, 2, and 3 to the text, type:

```
H(EX) I(nput) 31 32 33 <CR>
```

Position the cursor over the one and type `3 H(EX) O(utput)` to display the characters. The characters are displayed on the message line.

Quit Command

The **quit** command performs several functions depending on its subcommand. It ends the editing session, it initializes processing a new input file, or it updates your edited file. **Quit** has different prompts depending on whether or not a filename has already been specified for the file you are editing.

Filename Specified

Press Q; AEDIT prompts:

```
---- Editing input file [to output file]
Abort           Exit           Init           Update          Write
```

To specify a subcommand, press the initial letter of that subcommand.

A-Abort

When **abort** is activated, if any changes have been made to the current file, AEDIT prompts `all changes lost? (y or [n])` to avoid inadvertent loss of text. A `y` continues the abort process; the same procedure is then applied to the secondary file. Control returns to the operating system only after both questions have been answered with `y`. All changes (if any) that were made to the input file(s) are lost. If either question is answered with a response other than `y`, AEDIT returns to the main command level.

Note that:

- You are questioned concerning only input files that have been changed.
- The first question (if any) relates to the current file and the second one (if any) relates to the secondary file.
- If you answer `y` to the first question and `n` to the second while AEDIT is still active, the next time you type `quit abort`, you will not be asked further about the file about which you have already answered `y` unless new changes have been made.

E-Exit

When **exit** is activated, AEDIT rewrites the current file. Then, if the OTHER file has also been changed, AEDIT automatically performs the **other** command and asks `all changes lost? (y or [n])`. A `y` returns AEDIT to the operating system without rewriting the OTHER file. Any response other than `y` returns the editor to the main command level.

I-Init

Use this option to start editing a new file without returning to the operating system. If any changes have been made to the current file, the menu prompts `all changes lost? (y or [n])`. If `y` is answered (or if no changes have been made since the last update), AEDIT prompts:

```
---- <HOME> to re-edit
enter [file [TO file | VO | FO ]]:
```

Enter the input file (or `<CR>`) followed optionally either by an output file name or by the **VO** or **FO** controls. **VO** is the abbreviation of the control **viewonly**; **FO** is the abbreviation of **forwardonly**. `File` is the file you want to edit, `TO file` indicates the output file. The `TO` option can be used only when the input file exists.

See also: **viewonly**, **forwardonly** commands, Chapter 4

U-Update

When **update** is activated, the updated version of your file is written without returning to the operating system.

After the file has been written, the message `file has been written` is displayed.

After completing the **quit update** command, the editor is at the **quit** prompt level, not at the main command level. Press `<Esc>` to continue editing, or enter another **quit** subcommand, such as **init**.

Quit Command

W-Write

This option writes your file to the specified filename. This prompt is displayed:

```
---- <HOME> to re-edit
Output file:
```

Enter the output filename. If the specified file exists, the editor beeps and this question is displayed;

```
overwrite existing file? (y or [n])
```

- If **y**, the entire text file is written to the named file, overwriting the existing file. AEDIT returns to the **quit** prompt level.
- Any other response returns the editor to the **quit write** prompt level.

After the file has been written, the message `file has been written` is displayed.

The **quit** prompt is always reissued after an **update** or **write** subcommand. Press `<Esc>` or `<Ctrl-C>` to return to the main command level. Two points to remember are:

- **Quit abort** and **quit exit** relate to the entire AEDIT session, i.e., to both the current file and the secondary file. **Quit init**, **quit update**, and **quit write** relate only to the file you are currently editing.
- An output file indicated by the `TO` clause can be specified for a file in either the invocation line or at the **quit init** command. Only the subcommands **update** and **exit** relate to this output file; **write** does not.

Filename Not Specified

If you are editing a new file and have not yet specified a filename, the **exit** and **update** subcommands are not available as both subcommands require a filename. The AEDIT prompt is altered as:

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

The subcommands function the same as discussed above.

Paragraph Command

To invoke the **paragraph** command, press P. The **paragraph** command reformats a paragraph using the values set for indent and left and right margin in the **set margin** command. The default is to reformat one paragraph.

⇒ Note

The **paragraph** command execution starts by identifying the beginning of the current paragraph. AEDIT searches backward for an empty line; then, AEDIT finds the end of the paragraph by searching forward for an empty line. This implies that if your file has no empty lines, **paragraph** will process the entire file as one paragraph, wherever the cursor is currently positioned in the file.

Description

Move the cursor to any position in the paragraph to be reformatted.

Press P; AEDIT prompts:

Fill Justify

To specify an option, press the initial letter of that option.

Count: `count` defines the number of consecutive paragraphs to reformat.

Related Commands: **set margin**

F-Fill

Filling means that the white space sequences are reduced to one blank, and every blank after a sentence terminator (such as a period or question mark) is extended to two blanks. Words are moved to fill the line between the right and left margins or from line to line if necessary. The first line is indented according to the value of indent. Words are moved to the left as much as possible. Words are not split and lines are not right-justified.

J-Justify

In justification, the first step is as described in the filling process. The second step is performed separately for each line: words are shifted to the right (if necessary), and the space between words expanded so that the last word of every line ends at the right margin and the spaces between words are approximately even. The last line of the paragraph is not right-justified.

Window Command

To invoke the **window** command, press `w`. It horizontally splits the text area of the screen into two partitions. Each partition or window contains the text, message, and prompt sections. The **window** command enables you to view two different parts of the same file or two different files, using the main file and the OTHER file.

The screen is split above the cursor row. If the cursor is placed so that one window's size is less than five rows, the screen is not split, and this message is displayed:

```
window too small
```

After the screen is split, pressing `w` causes the cursor to jump between the two windows.

If the same file is displayed in both windows, a change in one window is not reflected in the other window until you press `w` and jump to the other window.

See also: **kill_wnd** command

Related Commands: **kill_wnd**, **set viewrow**, **view**

Kill_wnd Command

Invoke the **kill_wnd** command by pressing **κ**. It returns the screen to one window. The current (active) window is the dominant one.

Related Commands: **window**

!system Command

The **!system** command enables you to execute a system command from within AEDIT. Activate it by typing an exclamation point (!) at the main command level. The prompt line then displays the previous system command (if any). You may re-edit the previous command or enter a new string. Terminate the input string or command by entering either <Esc> or <CR>. Before the string is executed, the command clears the text area of the screen.

Calc Command

To invoke the **calc** command, press C. It provides you with computation capabilities.

Description

Press C; AEDIT prompts:

```
---- <HOME> to re-edit  
Calc:
```

The last statement entered under the **calc** command is displayed to the right of the colon.

See also: **calc** command, Chapter 7

Execute Command

To invoke the **execute** command, press **E**. **Execute** is used to execute macros.

Description

Press **E**; AEDIT prompts:

```
---- <HOME> to re-edit  
Macro name:
```

The last macro name entered for this command is displayed to the right of the colon.

When the macro name is entered, the specified macro is executed.

See also: **execute** command, Chapter 5

Related Commands: **set display**, **set go**

Macro Command

To invoke the **macro** command, press M. It is used for manipulating macros.

Description

Press M; the menu prompts:

 Create Get Insert List Save

To specify a subcommand, press the initial letter of that subcommand.

When a macro file is specified in the invocation line (explicitly or implicitly), it is read and processed immediately after the AEDIT invocation. This has the same effect as using a **macro get** as the first command after invocation.

See also: Macro files, Chapter 5

Related Commands: **set go**



This chapter describes the AEDIT invocation and invocation controls, and operation specific to iRMX OS. The iRMX OS may be used with various terminals.

Invocation

This is the syntax that invokes AEDIT:

```
[directory]AEDIT [input_file [TO output_file|file_processing_mode]]  
[recover]  
[,other_input_file[TO other_output_file|file_processing_mode]]  
[execution_mode]
```

Where:

<code>input_file</code>	The file you want to edit. If a file is not specified, a new file is created, and it is named when you call the quit command.
<code>output_file</code>	The name of the destination file for the file you are editing. It is written when you call quit update or quit exit . If you specify viewonly or forwardonly for the input file, you cannot specify an output file.
<code>other_input_file</code> <code>other_output_file</code>	Filenames for the OTHER input and output files.
<code>file_processing_mode</code>	[viewonly noviewonly] [forwardonly noforwardonly]
<code>recover</code>	[recover norecover]
<code>execution_mode</code>	[macro](macro_file) nomacro] [macrosize(macro_buffer_size)] [batch nobatch]

Use a comma to separate the main filename from the OTHER filename.

Invocation Line Examples

1. Invoke AEDIT by itself to create a new file:

```
aedit<CR>
```

2. This example shows invoking AEDIT with an existing file:

```
aedit filename<CR>
```

3. This example shows invoking AEDIT with a main input file, an OTHER input file, and a macro file:

```
aedit a1.txt, 2.txt macro(txt.mac)<CR>
```

4. This example shows invoking AEDIT with a main input file, an OTHER input file that is **viewonly**, and specifying a macro file and macro size:

```
aedit main.txt, second.txt vo macro(prog.mac)  
macrosize(1024)<CR>
```

Error: Cannot Open Input File is displayed if an output file is specified and the input file does not exist.

Invocation Controls

AEDIT controls can be divided into three groups: file processing mode controls, recover control, and execution mode controls. Table 4-1 lists the AEDIT invocation controls. The remainder of this chapter explains each control in detail.

Table 4-1. AEDIT Invocation Controls

Control Name	Abbreviation	Default	Meaning
File Processing Mode Controls			
forwardonly noforwardonly	FO NOFO	NOFO	Enables faster editing of large files, but the files are truncated.
viewonly noviewonly	VO NOVO	NOVO	Enables fast viewing of large files; no changes allowed.
Recover Control			
recover norecover	RC NORC	NORC	Enables file reconstruction
Execution Mode Controls			
macro nomacro	MR NOMR	MR(:HOME:aedit.mac)	Specifies macro file.
macrosize	MS	MS(3072)	Defines macro buffer size
batch nobatch	BA NOBA	NOBA	Activates AEDIT in noninteractive mode; used if AEDIT is activated from a command file.

The processing mode for an input file uses the **viewonly** and **forwardonly** controls.

- Specify these controls for either the main input file, for the OTHER input file, or both. **Viewonly** and **forwardonly** cannot be specified together for the same file. However, if you specify either one in the negative form, any combination is legal, e.g., **noviewonly forwardonly**.
- Give a filename if either of these controls is specified in the positive form in the invocation line.
- When specifying an output file, these controls may only be used in their negative form, e.g., **noviewonly**.
- Use only the **viewonly** and **forwardonly** controls under **quit init**.

The processing mode for the main input file uses the **recover** control. This control may be specified only in the invocation line and only for the main input file.

The execution mode uses the **macro**, **macrosize**, and **batch** controls. These controls may be specified once per AEDIT invocation. They cannot be specified for a particular file, and they cannot be specified under **quit init**.

A control may be specified only once, except for **viewonly** and **forwardonly**, which may be specified once for the main input file and once for the OTHER input file.

Forwardonly

Syntax

FORWARDONLY | NOFORWARDONLY

Abbreviation

[NO]FO

Default

NOFORWARDONLY

Control Type

Processing mode for an input file

Description

Forwardonly enables faster editing of large files because it instructs AEDIT to allocate a fixed amount of memory for the file. If the file is larger than the amount of memory allocated, some text may be lost. This loss applies only to the current editing of the file; the original file is not affected.

Forwardonly can be specified for either the main input file or the OTHER input file. **Forwardonly** can also be specified under the **quit init** command. **Forwardonly** cannot be specified simultaneously with **viewonly**. When **forwardonly** is in effect for the input file, an output file may not be specified.

While the **forwardonly** control is in effect, the message line displays the word `Forward`.

Error: `some text lost` is displayed if text is lost during the current edit. If this error is displayed, you are unable to execute **quit update** or **quit exit**; however, you may execute **quit write**.

Viewonly

Syntax

VIEWONLY | NOVIEWONLY

Abbreviation

[NO]VO

Default

NOVIEWONLY

Control Type

Processing mode for an input file

Description

You can view a large file, such as a large listing file that you do not want to change, much faster using the **viewonly** control. It is also an advantage to use **viewonly** if you want to be certain that no changes are made unintentionally.

You can specify **viewonly** for: the main input file, the OTHER input file, or under the **quit init** command. It cannot be specified simultaneously with **forwardonly**. When **viewonly** is in effect for the input file, an output file may not be specified.

If using the **viewonly** control, the input file may not be changed. These keys are not valid with **viewonly**: **rubout, delch, dell, delr, delli**.

These commands are still displayed on the prompt line; however, they are not valid with **viewonly**: **(?)replace, block delete, get, hex input, insert, macro insert, macro save, paragraph, quit exit, quit update, quit write, and xchange**.

All other commands are legal. You may save a **viewonly** file or a portion of it using the **block put** command.

While the **viewonly** control is in effect, the message line displays the word **view**.

Error: `illegal command` is displayed if a command is given that is invalid with **viewonly**.

Recover

Syntax

RECOVER | NORECOVER

Abbreviation

[NO]RC

Default

NORECOVER

Control Type

Processing mode for the main input file on invocation

Description

The **recover** option can be used to help you reconstruct edited files if a fatal system error occurs during AEDIT operation, or if an unintentional termination of an AEDIT session occurs using **quit abort**.

If a crash occurs, reinvoke AEDIT with the **recover** control. The **recover** control can be specified only for the main input file and only in the invocation line.

When **recover** is specified, AEDIT takes the entire memory contents as the input file. If the memory contains previously edited file(s), your file must be reconstructed.

Recover is, however, only a means for the reconstruction. You must identify, gather, and connect the relevant text portions in memory.

Recover may be used only if the memory allocated to AEDIT in the current activation is the same as that used in the previous activation. This implies that **recover** is probably useless on virtual-memory-based systems or in a multitask environment.

The reconstruction process is difficult or impossible if the edited file is so large that it is spilled to extra memory or to temporary files. The memory content in such a case does not reflect the entire file contents.

If an input file is specified, the input file is not read when **recover** is in effect, but it serves as an output file for the **quit update** command.

Macro

Syntax

```
MACRO [(macro_file)]|NOMACRO
```

Abbreviation

```
[NO]MR
```

Default

```
MACRO (AEDIT_filename.MAC)
```

Control Type

Execution mode

Description

Use the **macro** control to specify a macro file for the current AEDIT invocation. The **nomacro** option prevents AEDIT from reading a macro file. Not specifying this control or just specifying **macro** is equivalent to the default. When using the **macro** control with a filename, the filename can have any extension.

The default macro file is *aedit.mac* in your *:home:* directory.

Macrosize

Syntax

```
MACROSIZE (macro_buffer_size)
```

Abbreviation

MS

Default

```
MACROSIZE( 3072 )
```

Control Type

Execution mode

Description

Use this control to allocate more macro buffer space if, for example, a huge batch operation is implemented using macros. Also, more macro space may be required if many macros or long macros are used.

Macrosize enables you to specify the macro buffer size for the current AEDIT invocation. `macro_buffer_size` is a decimal number specifying the number of bytes to be allocated. The minimum `macro_buffer_size` is 1024 (400H) bytes; the maximum allowed is 32767 (8000H) bytes. The default size is 3072 bytes. The maximum size actually allowed also depends on the amount of RAM available; therefore, it may be less than the maximum listed.

The buffer that is allocated for macros is not available for text; therefore, allocating a large macro buffer is not recommended.

Batch

Syntax

BATCH | NOBATCH

Abbreviation

[NO]BA

Default

NOBATCH

Control Type

Execution mode

Description

Use the **batch** control to activate AEDIT in a noninteractive mode, usually from a command file. When **batch** is in effect, AEDIT suppresses all output except the MESSAGE line.

Although AEDIT may receive input from the console in **batch** mode (implying a semi-batch mode where input is from the keyboard), this is not recommended. For example, in this mode yes/no questions (such as `ok to replace?`) are suppressed.

AEDIT commands specified after the invocation line can be given in the **init** macro if it is present in the default or specified macro file. Executing batch commands using the **init** macro is preferable because it works equivalently on different operating systems and different terminals.

Activation with Batch Control

If AEDIT is invoked from a command file, all input is from the command file. The sequence of commands and characters should be exactly the same as if you were executing AEDIT interactively. Input is echoed to the system console.

Examples

1. If you want to change *dog* to *cat* throughout your file, you can create this command file:

```
...
AEDIT EXAMPL.SRC BATCH
/Rdog<Esc>cat<Esc>QE
...
```

Where:

EXAMPL.SRC is the input file.

/R means replace all occurrences.

QE is the **quit exit** command.

Insert <Esc> in the command file using the **hex input** command or using the function key for **hex** under **insert**.

2. If all operations are defined in a macro, the command file requires only two AEDIT commands. In this example, given a macro with the required operation sequence called **pass1** and a macro file called *pass1.mac*, the command file is as follows:

```
...
AEDIT EXAMPLE.SRC BATCH
MGPASS1.MAC<Esc>EPASS1<Esc>
...
```

Where:

EXAMPLE.SRC is the input file.*

MG is the **macro get** command.

Pass1.MAC is the macro file.

E is the **execute** command.

PASS1 is the file containing the operation sequence.

See also: Macro files, Chapter 5

Default Macro File

The name of the default macro file is *aedit.mac*; it is assumed to be in the home directory. If it is not, you must explicitly specify the **macro(filename)** control.

Work File

The predefined file *:work:* must be properly assigned when AEDIT is invoked. This assignment should be done automatically when iRMX is booted.

See also: Logical Names Created by the Operating System, *Command Reference*

iRMX-specific Information

When using AEDIT on iRMX Operating Systems, do not perform these operations:

- Do not use the **attachfile** facility to redefine the default directory.
- Do not press <Ctrl-C> more than once when a **!system** command is being executed; pressing it twice in succession may abort AEDIT.

If you are using an integrated environment where upper and lower case characters are significant in filenames, note that AEDIT always creates uppercase filenames. You must manually convert the filenames to lowercase if your environment requires it.



AEDIT macros are sequences of AEDIT commands (sequences of keystrokes) that have been collected and given a name.

The AEDIT macro set uses these subcommands for processing macros. These subcommands are listed under the **macro** command.

create	create a macro interactively
get	processes a macro file
insert	create a macro directly
list	displays all available macros

Additional information concerning deleting macros, executing macros, and processing macro files is listed later in this chapter.

Macro Command

Invoke the **macro** command by pressing **M**, which allows you to manipulate macros.

A macro definition is a series of commands written in macro form. Define macros interactively through the **macro create** command, or directly through the **macro insert** command. To save interactively defined macros, you must write them to a separate macro file in macro form. Use the **macro get** command get the macro file.

Description

Press **M**; AEDIT prompts:

```
      Create      Get      Insert      List      Save
```

To specify a subcommand, press the initial letter of that subcommand.

C-Create Subcommand

The **create** subcommand creates a macro interactively by accumulating a sequence of keystrokes. The macro is executed and created concurrently.

Press **C**; AEDIT prompts:

```
---- <Home> to re-edit  
Macro name:
```

The name of the last macro specified for **macro create**, **macro save** or **execute** (if any) is displayed to the right of the colon. Type in the macro name followed by **<Esc>** or **<CR>**.

A macro name can consist of either a single character or a character string of up to 60 characters. The macro name may contain any characters, for instance **+**, **6**, **a**.

After you type the macro name, the word **Macro** is displayed on the message line, and remains there until the macro definition is complete. AEDIT returns to main command level, and the entire set of AEDIT commands is now available for **macro create**.

All subsequent keystrokes are executed in the regular manner, but they are also trapped by the editor. These keystrokes make up the macro definition. This includes special keys like **<Esc>** and the key for **hex**.

Terminate the macro by typing one of these characters:

<Ctrl-C> Terminates **macro** mode without defining the macro; the macro is deleted.

M (main level only) Successful termination of macro definition.

By defining a single-character macro, you can configure keys to execute the macro in a single stroke, thus making them powerful new function keys.

See also: **execute** command, described later in this chapter

Examples

This is an example that interactively creates the macro dot (.) that finds the next occurrence of the last target string:

```
M(ACRO) C(reate)
Macro name: .<Esc>
F<Esc>M
```

This is an example that interactively creates a macro to configure <Ctrl-L> to mean jump to start of line.

```
M(ACRO) C(reate)
Macro name: <Ctrl-L><Esc>
<Right><Left><Home>M
```

Error: no more room for macros is displayed on the message line if macros exceed the amount of memory allocated to macros. The definition is terminated, and the current incomplete macro definition is deleted.

G-Get Subcommand

The **get** subcommand reads and processes a macro file, with the result that:

- The macro definitions in the file are available for execution.
- The configuration commands in the file are executed.
- The **set** commands in the file are executed.

The new macro definitions are added to the current set of available macros. If a macro in the new set has the same name as a macro already available, the new macro overrides the previous one. Configuration commands and **set** commands are executed.

See also: Macro files

Press G; AEDIT prompts:

```
---- <Home> to re-edit  
Macro file:
```

The name of the last macro file read (if any) is displayed to the right of the colon. Edit the macro filename, if required, then press <CR>.

You may insert an empty string as a filename (e.g., by typing `M(acro)G(et)<CR>`); AEDIT gets the present text buffer as a macro file.

Errors: Errors may be issued during a **macro get**. The error is displayed, the area causing the error is skipped, and processing continues.

Related Commands: The invocation controls **macro**, **macrosize**

I-Insert Subcommand

The **insert** subcommand causes all subsequent input, including function keys (such as <Esc>) to be inserted in the text in macro form and not executed. Use it to change and correct macro files. For example, pressing <Up> in **macro insert** inserts the character sequence `\CU` in to the text. The macro definition may then be saved in a macro file. This command is terminated by pressing <Ctrl-C>.

Press I; AEDIT prompts:

```
Control C to stop
```

In **macro insert**, all keys are entered as-is (e.g., **f(ind)** is entered as `F`). Thus, keys such as <Esc> do not perform a function but are inserted as their macro codes. These are exceptions:

- When <CR> is typed it is not converted to `\NL` because the line terminator is used to break macro definitions into more readable lines. Therefore, you must type `\NL` if a line terminator is required in the definition.
- If the backslash is not a lead-in character, you must enter it twice (`\\`). However, the backslash is not doubled when it is typed, which enables you to type `\MM` to terminate the macro or `\NL` for line terminator.

Type <Ctrl-C> to terminate **macro insert** mode.

Example

This macro defines <Ctrl-L> to mean jump to start of line. Remember that what you type does not execute but is inserted in macro form.

```
AEDIT filename
M(ACRO)I(nsert)M<Ctrl-L><Esc><Right><Left><Home>
\MM(end macro)
<Ctrl-C>
Q(UIT u(pdate) or Q(UIT) E(xit)
```

This text is inserted into filename:

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

MM terminates the macro definition.

See also: Macro modes, described later in this chapter

L-List Subcommand

The **list** subcommand displays on the message line the names of all currently available macros. If there are more macros available, press the space bar to continue; any other character returns the editor to the main command level.

S-Save Subcommand

The **save** subcommand translates an available macro to macro form and inserts the definition at the current position in the text. The macro may subsequently be modified or saved in a macro file. If you want to look at a macro definition, use **macro save** to translate and display the macro, review it, and delete it (if desired).

Press S; AEDIT prompts:

```
---- <Home> to re-edit
Macro name:
```

The name of the last macro specified (for **macro create**, **macro save** or **execute**) is displayed to the right of the colon. Type a macro name followed by <Esc> or <CR>. If the macro exists, it is inserted in the text at the current cursor location in macro form.

You may use this procedure to save a new, interactively created macro for future use:

1. Press o to enter the OTHER buffer.
2. Use the **quit init** command to start editing your macro file.
3. Insert the macro in macro form using the **macro save** command.
4. Update the modified macro file using the **quit update** or **quit exit** command.

Deleting Macros

To delete a macro from the set of available macros, use the **macro create** command. Type:

```
M(ACRO)C(reate) macro_name <Esc><Ctrl-C>
```

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 from a macro file, you should edit the macro file like any other file, and use the delete commands.

Macros and AEDIT Variables

When AEDIT variables are referenced during **macro create**, the actual variable value is fetched for the current activation of the macro, and a reference to this variable is inserted into the macro definition. AEDIT variables provide a primitive way to simulate passing a parameter to a macro.

Macro Modes

A macro may be either modeless (terminated with `\MM` after it is converted to macro form) or non-modeless (terminated with `\EM` after it is converted to macro form). All macros created with **macro create** are modeless. You can create a non-modeless macro by using **macro insert** or by editing a saved modeless macro.

You can use any modeless or non-modeless macro at the main command level or in either **insert** or **xchange** modes.

A modeless macro is independent of whether it is called from main command level, **insert** or **xchange** mode. This enables you to use the same macro at the main command level and in **insert** or **xchange** mode. When you execute a modeless macro, it executes as if it is at the main command level.

When the macro finishes execution, it restores the mode (**insert** or **xchange**) that was in effect when it was activated.

A non-modeless macro is executed at the AEDIT prompt level that was in effect when the macro was activated. When the macro finishes execution, it does not restore the mode that was in effect when it was activated. Instead, AEDIT remains in the mode determined by the macro. Non-modeless macros provide compatibility and upgrading with respect to AEDIT V1.0.

Modeless macro execution always gives the same result regardless of the mode from which it was executed. Non-modeless macro execution results usually depend on the context mode from which they were called. Use modeless macros whenever possible.

For example, compare these macros:

```
MA\BRi*\BR\MM
MB\BRi*\BR\EM
```

The first macro is modeless. When it is executed, the character `*` is inserted whether or not it was called from main command level, **insert**, or **xchange** mode; when the macro finishes, the initial mode is retained.

The second macro is non-modeless. If it is called from the main command level, the results are the same as for the first; i.e., the character `i` is inserted and the editor remains at main command level. But, if it is called from **insert** mode, for example, the characters `i*` are inserted into the text, and the `<Esc>` command (`\BR`) causes the editor to leave **insert** mode and return to the main command level.

Execute Command

The **execute** command requests a macro name and executes the specified macro.

In macro execution all input is taken from the macro except for answers to these questions/requests:

- **?Replace:** ok to replace?
- **Quit init** or **quit abort:** all changes lost?
- **Block delete:** cannot save in memory, save anyway?
- hit space to continue
- **quit write** or **block put:** overwrite existing file?

In the prompts listed above, the response to the prompt is taken from the console.

Description

Press [*count*] E; AEDIT prompts:

```
---- <Home> to re-edit  
Macro name:
```

The name of the last macro specified (for **macro create**, **macro save**, or **execute**) is displayed to the right of the colon. Type a macro name followed by <Esc> or <CR>. If the macro exists, it is executed.

If the busy/waiting indicator is active, the prompt line displays -!!- when a macro is being executed. This is important in single-character macro execution where there may not be any other indication that a macro is still executing.

The macro terminates when it has been executed the specified number of times or has failed. Macro execution termination is described in more detail later in this chapter.

Macro activation may be nested up to eight levels.

Type <Ctrl-C> to force termination of macro execution.

Errors:

- no such macro is displayed if the macro specified does not exist.
- macro nesting too deep is displayed if you attempt to nest macros to more than eight levels.

Count: the **execute** command accepts any count.

Related Commands: **set display**, **set go**

Single-character Macros

You can write macros with single-character names. You can activate a single-character macro by simply typing its name if this character has no other function in the context from which it is being activated. Single-characters that can be used as macro names are referred to as "free" characters, e.g., `L`, `U`, `Z`, `+`.

Single-character macros can be activated in these ways:

- Using the **execute** command at the main command level (as in the case for all macros).
- By pressing the macro character preceded by the key for **mexec**.
- By pressing the key itself, if the key is "free", as follows:
 - In **insert** and **xchange** modes: all nonconfigured control characters. These control characters also cannot be used as macros in this way: `<Ctrl-M>` (`<CR>`), `<Ctrl-I>` (`<Tab>`), and `<Ctrl-J>` (line feed).
 - In the main command levels: the same as for **insert** and **xchange** modes with the addition of all printable characters that are not used as AEDIT commands.

The command **mexec** is configurable to a key (usually `<Ctrl-E>`). To activate the single-character macro, press `<Ctrl-E>` followed by the macro name. For example, if you are using the macros from *useful.mac* in **insert** mode, typing `<Ctrl-E>^` converts the word that the cursor is on to uppercase letters.

A digit may not be used as a single-character macro at the main command level because it is always interpreted as a *count*. A function key may not be used as a macro name because the key's function overrides the macro definition.

This is an example to demonstrate using a single-character macro in **insert** or **xchange** mode. You can define a macro called `<Ctrl-P>` as the word *procedure* and save this macro in a file called *plm.mac*. If you are editing a PL/M source file, you can call *plm.mac* with **macro get**; then, each time you type `<Ctrl-P>` the word *procedure* will be inserted into the file. This saves having to type out the word each time you want to insert it.

Errors:

- `illegal command` is displayed when you type a character that is not a command abbreviation, a decimal digit, or a macro name, at the main command level.
- `no such macro` is displayed if the character following the **execute** command or the **mexec** key is not a macro name; i.e., no macro with that name exists.

Macro Files

A macro file may consist of:

- Configuration commands
- **Set** commands
- Macro definitions
- Macro comments

Set commands and configuration commands are the only commands used in a macro file. However, by using the **init** macro you can specify any command. Using the **set** commands enables you to specify the mode of operation. For example, you may include the command **set k_token yes** if you want the AEDIT search mode initialized to search for tokens only. In the macro file, this command appears as `SKY`.

A macro definition is a series of commands written in macro form. It has this format:

```
M macro_name \BR characters_in_macro \MM
```

Where:

<code>M</code>	declares that a macro definition follows.
<code>macro_name</code>	is any name given to the macro being defined.
<code>\BR</code>	stands for <Esc>.
<code>characters_in_macro</code>	is the macro contents.
<code>\MM</code>	signals the end of a modeless macro.
<code>\EM</code>	signals the end of a non-modeless macro.

These representations of control characters and control codes are used in the macro definitions:

Name	Represents
\BR	<Esc>
\CL	<Left>
\CR	<Right>
\CU	<Up>
\CD	<Down>
\CH	<Home>
\NL	<CR>
\RB	rubout
\TB	<Tab>
\XF	delch , delete character
\XX	dell , delete left
\XA	delr , delete right
\XZ	delli , delete line
\XU	<Undo>
\XH	hex , hex prefix character
\XE	mexec , macro execute
\XN	ftn , fetch numeric
\XS	fets , fetch string
\Oh	hexadecimal value of a character
\MM	end of modeless macro definition
\EM	end of non-modeless macro definition

The backslash (\) must appear twice if it is not used as a code lead.

A macro definition and configuration commands should be ended with either a semicolon (;) or line terminator <n1>. The <n1> (as opposed to \NL and \OA) is ignored in a macro file, even within macro definitions. This enables you to split the macro definition into lines so that it is easier to read.

Mark the beginning of a comment in a macro file with a backslash/asterisk character pair (*) and end-mark with an asterisk/backslash pair (*\).

Macro Execution After a Failure

A command execution is marked as failed if:

- An attempt is made to move forward (<Right>, <Down>, <Home>, or <CR>) at the end of the file.
- An attempt is made to move backward (<Left>, <Up>, or <Home>) at the start of the file.

⇒ **Note**

A cursor movement command prefixed by +/ is never marked as failed.

- A command prefixed by a finite count is marked as failed if any of its execution is thus marked.
- A **(-)find** or **(?)replace** command fails to find the target string.
- A **(-)find** or **(?)replace** command prefixed by / is marked as failed only if it fails on its first execution.

⇒ **Note**

If the **set go** option is on, the **(-)find** and **(?)replace** commands are never marked failed.

When a command in a macro is marked as failed, macro execution is terminated, and control is returned to the caller. If the caller is main command level, AEDIT simply waits for the next command. If the caller itself is a macro, execution continues with the caller's next command.

Examples

```
Macro A:      /E(XECUTE)B  
              E(XECUTE)C  
              E(XECUTE)D
```

```
Macro B:      J(UMP)S(tart)  
              /R(EPLACE)"<nl><nl>" with "<nl>"
```

```
Macro C:      S(ET) G(o) Y(es)
              J(UMP)S(tart)
              /R(EPLACE) "DCL" with "DECLARE"
              J(UMP) S(tart)
              /R(EPLACE) "IS" with "LITERALLY"
```

A value for **set go** in macro A is meaningless, because it does not contain a **(-)find** or **(?)replace** command. When `/EXECUTE B` is terminated either normally or because macro B failed, macro C is executed. Likewise, when macro C is terminated, macro D is executed.

Set go must be set to `no` (the default) for macro B. The **replace** command is successful as long as the file contains at least one `<nl><nl>` sequence. When the file contains no `<nl><nl>` sequences, macro B fails, execution of macro B is terminated, and macro C is executed. If **set go** is `yes` for macro B, it will never fail, and execution of macro B would enter an infinite loop.

Set go must be set to `yes` for macro C. This option ensures that the second **replace** command is performed regardless of the results of the first **replace** command. `S(et) G(o) Y(es)` could be placed after the **jump** command, and the effect would be the same.

If you are not careful in coding your macro, it might enter an infinite loop when it executes. To exit from such a macro or to terminate any macro, press `<Ctrl-C>`.

Screen Display During Macro Execution

To speed up macro execution, the amount of data written to the screen while a macro is executing is reduced to a minimum. Only selected text or information is sent to the screen. The information in this section is given for reference.

Text

If **set display** `no` (the default) is in effect, changes in the text and cursor movements are displayed on the screen as long as the cursor does not leave the current display screen, or until a **view** command (either explicit or implicit) is issued. When the cursor leaves the screen or a **view** command is issued, screen display is frozen until the macro execution terminates. Then, an implicit **view** command is performed using the current logical cursor location.

If **set display** `yes` is in effect, all changes and cursor movements are displayed on the screen, even if the cursor leaves the screen or a **view** command is given.

Regardless of the **set display** value, the text is updated if you give the **?replace** command or the **find/replace** command and **set showfind** `yes` is in effect.

Message

The message line is updated only in these cases:

- Error message display. The error message display lasts for at least a second. The **find/replace** command message `not found` is not considered an error.
- **Calc** command messages. For an argument that is an expression rather than an assignment statement (e.g., `N3+1` versus `N2=N3+1` or `S9` versus `S9="abc"`).
- **Hex output** and **macro list** command messages.
- **Quit** or **other** command filename messages.

When macro execution is terminated, the last message remains on the screen. However, the status (e.g., **other**, **viewonly**) is updated if needed.

The `found: (number)` message is a special case; it is displayed only if it is the last message of the macro execution at main command level.

Prompt

The prompt line is changed only when a macro requests an answer to one of the questions listed in the earlier section on the **execute** command in this chapter.

After macro execution terminates, the prompt line reflects the current mode of the editor. This mode is either main command level, **insert** mode or **xchange** mode.

Window

When a new window is constructed, the text is updated immediately, which is an exception to the previous description. The reason for this exception is to create a place for a future message that may refer to the upper window. The **kill-wnd** command operates in the usual manner. That is, when the cursor leaves the screen, the screen will be updated only when macro execution terminates.

Macro Examples

1. This example resets the left column. The single-character macro right square bracket (]) sets the left column one position to the right each time it is executed, and the single-character macro left square bracket ([) sets the left column one position to the left each time it is executed:

```
M ] \BRs1+1\NL \MM
M [ \BRs1-1\NL \MM
```

These macros can be defined interactively using the **macro create** command. For example, the first macro can be created by entering:

```
M(ACRO)C(reate)
Macro name: ]<Esc>
S(ET) L(eftcol)+1<CR>
M(to terminate macro definition)
```

After you define these macros, typing a right square bracket (]) at main command level, or the key for **mexec** and] in **insert** or **xchange** mode, sets **leftcol** one position to the right of its current setting; typing a left square bracket ([) at main command level, or the key for **mexec** and [in **insert** or **xchange** mode, sets **leftcol** one position to the left of its current setting.

2. These single-character macros, named dot (.) and comma (,) find the next occurrence of the target string, or find the previous occurrence of the target string, respectively.

```
M. \BRf\BR\MM
M, \BR-\BR\MM
```

You can define these macros interactively using the **macro create** command. You could create, for example, the first macro as:

```
M(ACRO)C(reate)
Macro name: .<Esc>
F(IND)<Esc>
M (to terminate the macro definition)
```

After you define these macros, typing a dot (.) at main command level or the **mexec** key followed by a dot in **insert** or **xchange** modes finds the next occurrence of the target string; typing a comma (,) at main command or the **mexec** key followed by a comma in **insert** or **xchange** modes, finds the previous occurrence of the target string.

- This example sets visual breakpoints in programs. For example, you can use comment lines filled with hyphens to separate procedures in a language. This macro creates a break line:

```
M@\BRI( *-----* ) \NL\BR\MM
```

You can create this macro interactively with these commands:

```
M(ACRO) C(reate)
Macro name: @ <CR>
I( *-----* )<CR><Esc>
M
```

After you define this macro, typing @ at main command level, or the **mexec** key and @ in **insert** or **xchange** mode, inserts the break line at the current cursor position in text.

- This sequence of commands saves, in a new file named *exmpl.mac*, an interactively defined macro named asterisk (*) that allows you to scroll backwards ten lines.

M(acro) C(reate)	Creates macro * interactively
Macro name: * <CR> <Esc>10<UP>M	Command sequence
O(ther)	Invokes the macro save command, which prompts for the macro name.
Q(uit) I(nit) EXMPL.MAC<CR>	
M(acro) S(ave)	
Macro name: * <CR>	Executes the macro save command, translates the macro * to macro form.
Q(uit) U(pdate)	Updates the macro file (<i>exmpl.mac</i>), which now includes the macro *.

□ □ □

You can access a set of AEDIT variables with the following characteristics: string variables versus numeric variables, read only variables versus read-write variables, local variables versus global variables.

- Read-only variables reflect internal AEDIT values that you can retrieve but not modify. Read-write variables can be modified freely. Read-write variable assignment can be done only in the **calc** command.
- Local variables can be accessed only in the **calc** command. Global variables can be accessed in other contexts as well.

Global Variables

The two types of global variables are numeric and string. The global numeric variables are all read-write. The global string variables can be read-only or read-write.

Global Numeric Variables

The ten global numeric read-write variables (N variables), are $N0-N9$, which are 32-bit numbers. Values are assigned to the N variables only in the **calc** command. To fetch an N variable, type **<fctn>** i , where **<fctn>** is the **fetch numeric** key (usually configured to **<CTRL-N>**), and i is any digit from 0-9. When AEDIT is invoked, the N variables are initialized to zero.

The N variables may be used in the following ways:

- In any line-edit prompt such as *target-string* or *replacement-string*, or for the **get** command. The contents are inserted and displayed as signed decimal numbers; leading zeros are suppressed.
- As a **count** (or part of a **count**) for a command. **Count** cannot be negative; therefore, the absolute value of the N variable is used. In this case, the value of the N variable should be in the valid range, 0-65535. The contents are displayed as an unsigned decimal number; leading zeros are suppressed.
- In **insert** and **xchange** modes. For example, if you type **<fctn>**1, the contents of $N1$ are inserted in the text according to the value of **set radix**. For example, if variable $N1$ contains the hexadecimal value 45H, the following text would be inserted:

If SET Radix is:	Text inserted:
alpha	E
binary	1000101
decimal	69
hex	45
octal	105

The value is inserted without a suffix and leading zeros are suppressed.

- Under the **calc** command. The variable may be retrieved as in any line-edit prompt. Also, you can use the variable name as-is, for example, $N1$ instead of **<fctn>**1. In this case the name, rather than the value, is displayed, for example, $N1$ instead of 45. The entire processing is done by the **calc** command, not by the line-editing mechanism. A variable may be modified only if it appears in **calc** with its name on the left-hand side of an assignment statement.

When you press the key for **<fctn>**, the message **<FETN>** appears on the message line. This message disappears on the next keystroke.

Global String Variables

The two groups of global string-variables (*S* variables) are:

- Read-write string variables
- Read-only string variables

Read-Write String Variables

The ten read-write variables are *S0-S9*. Value assignment is done only in the **calc** command. When AEDIT is invoked, these variables are initialized to a null string.

Read-Only String Variables

The following is an alphabetic list of the read-only variables. No assignment of values is allowed.

<i>SB</i>	Up to 60 characters of the block buffer. By using <i>SB</i> , a portion of the text file may be used later as, for example, an argument in a find command.
<i>SE</i>	The name of the current edited file (as opposed to the secondary file).
<i>SG</i>	The name of the last file specified in the get command.
<i>SI</i>	The name of the main input file.
<i>SM</i>	The name of the last file specified for the macro get command.
<i>SO</i>	The name of the OTHER input file.
<i>SP</i>	The name of the last file specified for the block put command.
<i>SR</i>	The replacement string of (?)replace .
<i>ST</i>	The target string of (-)find and (?)replace .
<i>SW</i>	The name of the last file specified in the quit write command.

Using String Variables

To fetch the value of a string variable, type `<fets>x`, where `<fets>` is the key configured to **fetch string** (usually `<Ctrl-V>`), and where `x` is a digit (0-9) or a letter appearing as the second letter in a name in the above list (e.g., B, E, G). For example, `<fets>7` fetches the value of `S7`, and `<fets>G` fetches the value of `SG`. An `S` variable contains a string of 0-60 characters.

The `S` variables may be used in the following circumstances:

- In any line-edit prompt such as *target-string*, *replacement-string*, **get filename**. The contents are inserted and displayed as an ASCII string.
- In **insert** and **xchange** modes. For example, if you type `<fets>1`, the contents of `S1` are inserted in the text.

Note that a character that is inserted in this way loses any special meaning it may otherwise have. For example, `0DH` is inserted as-is and not as a carriage return, or `01BH` (`<Esc>`) does not cause mode termination.

- Under the **calc** command. The `S` variable may be retrieved as with any line-edit prompt. Also, you can use the `S` variable name as is, for example, `SB` instead of `<fets>B`. In this case, the name rather than the value is displayed, for example, `SM` rather than *aedit.mac*. All processing is done by the **calc** command and not by the line-editing mechanism. An `S` variable may be modified only if it appears in **calc** with its name on the left side of an assignment statement.

An `S` variable is always considered as if all its characters are literalized. This means they are interpreted as regular characters even if in other cases they may have a special meaning, such as `<hex>`. Thus an `S` variable is never searched to determine if it fetches another `S` variable.

When you press the key for `<fets>`, the message `<FETS>` appears on the message line. This message disappears on the next keystroke.

Local Variables

All local variables are read-only numeric variables. Therefore, they cannot be assigned, and they can be used only in the **calc** command.

The following is an alphabetic list of positional values:

<i>BOF</i>	Logical value: true if the cursor is at the beginning of file.
<i>COL</i>	The current logical cursor position in the line. This value is not affected by the setting of leftcol .
<i>CURCH</i>	ASCII value of the current character.
<i>CURWD</i>	ASCII value of the two bytes at the current cursor location.
<i>EOF</i>	Logical value: true if the cursor is at the end of file.
<i>INOTHR</i>	Logical value: true if you are in the <i>OTHER</i> buffer.
<i>ISDEL</i>	Logical value: true if the character at the current position is in the user-defined delimiter set.
<i>ISWHT</i>	Logical value: true if the character at the current position is whitespace (space, tab, LF or CR).
<i>LOWCH</i>	If the current character is an uppercase character (41H to 5AH), <i>LOWCH</i> is the ASCII value of the lowercase character. Otherwise, <i>LOWCH</i> is the same as <i>CURCH</i> .
<i>NXTCH</i>	ASCII value of the next character.
<i>NXTTAB</i>	The column number of the next tab position (to the right of the cursor) as defined by set tab . If there are no tabs to the right of the cursor in the current line, the value of <i>NXTTAB</i> is zero.
<i>NXTWD</i>	ASCII value of the second and third bytes following the current cursor location.
<i>ROW</i>	Current cursor row (actual row, not the logical line in the text).
<i>UPCH</i>	If the current character is a lowercase character (61H to 7AH), <i>UPCH</i> is the ASCII value of the uppercase character. Otherwise, <i>UPCH</i> is the same as <i>CURCH</i> .

The following values are offset from the beginning of the currently processed input file. If the file is edited using the **forwardonly** control, the offset is from the current beginning of text. This position may vary as the cursor is moved forward.

<i>CURPOS</i>	Offset of current location in file. <i>CURPOS</i> is zero for the first character of the file.
<i>TAGA</i>	Offset of tag A .
<i>TAGB</i>	Offset of tag B .
<i>TAGC</i>	Offset of tag C .
<i>TAGD</i>	Offset of tag D .

The following value relates to the *S* variables:

<i>SLx</i>	The length of the global <i>S</i> variable, <i>SLx</i> , where <i>x</i> is 0-9 or the second letter of a read-only string variable.
------------	---

The following is an alphabetic list of counters that contain the actual number of command repetitions from the last time the command was specified with count greater than one. The *CNT* prefix signifies *COUNT*.

<i>CNTEXE</i>	The number of times the macro that is currently executing has executed in the current activation. The first execution is number one. If none, the value is zero.
<i>CNTFND</i>	Relates to (-)find .
<i>CNTMAC</i>	The number of times that the last macro (which has finished executing) was executed.
<i>CNTREP</i>	Relates to (?)replace .

The following values relate to the margin and indentation settings used by **paragraph** and **set autonl** commands:

<i>IMARGN</i>	The value of current indent margin setting.
<i>LMARGN</i>	The value or current left margin setting.
<i>RMARGIN</i>	The value of current right margin setting.

The following values are returned using the system real-time clock services:

<i>DATE</i>	Date in decimal format <i>MMDDYY</i>
<i>TIME</i>	Time decimal format <i>HHMMSS</i>

The following are other values:

LSTFND Logical value: true if the target of the last find or replace string was found. (Note that an infinite **find (/f)** sets this variable to TRUE if at least one **find** succeeded.)

NSTLVL Nesting level of the currently executing macro, console input is level 0.

Global Variables in Macros

When the value of any AEDIT variable is fetched during **macro create**, the actual value is used for the current activation. A reference to this variable is inserted into the macro definition. Therefore, when you activate the macro at a later time, it will use whatever the value of the variable is at that time. It will not use the value of the variable at the time you created the macro.

For example, assume the following is a macro:

```
I (NSERT) <FETS>7<ESC>
```

If *S7*="abc" at the time the macro is defined, *abc* is inserted the first time the macro is run. If the definition of *S7* is changed to *xyz* and the macro is activated again, *xyz* is inserted.



The **calc** command provides you with computation capabilities. Using these capabilities coupled with AEDIT variables, you can perform functions such as centering a phrase on a line, finding the size of an input file, or changing a letter from lower case to upper case or vice versa.

Overview

To execute the **calc** command, press C; AEDIT prompts -

```
---- <Home> to re-edit
Calc:
```

The last statement entered under **calc** is displayed to the right of the colon.

Input to the **calc** command is either a numeric statement or a string statement:

```
numeric_statement:==[N_variable =]...numeric_expression
```

or

```
string_statement:==[S_variable =]...string_expression
```

Thus, a **calc** statement can be a numeric (or string) expression, optionally assigned to one or more N (or S) variables.

The **calc** command evaluates the numeric or string expression and displays its value in the message line. If the assignment form is used (e.g., $N1 = N2+1$), the left-side variable is assigned that value. As specified by the ellipsis you can include multiple assignments in a single statement. Furthermore, you can embed a numeric statement in a numeric expression if it is enclosed in parentheses.

A numeric expression can only be assigned to an N variable. Similarly, a string expression can only be assigned to an S variable.

Here are some simple examples:

```
S1 = "A simple string" — string statement
assignment"
SB — an unassigned string expression
N1 = N2 = CURPOS + 1 — multiple-assignment numeric statement
```

The various components of a **calc** statement are:

- `N_variable` or `S_variable`

Indicates one of the 10 `N(S)` variables. It can have one of two forms:

- `N` (or `S`) followed by a decimal digit, statically identifying the variable, e.g., `N1`, `N8`, `S5`.
- `N` (or `S`) followed by a parenthesized numeric expression that yields a value from 0-9. This sets the execution-time specification of a variable and is best suited for macro definitions.

Example: `N(N9+1)`

- Numeric expression is comparable to expressions in other high-level languages like C, Pascal, and PL/M.

The operands of a numeric expression can be either:

- Numeric constants, e.g., 4, 100, 5FH
- Local or global numeric variables, e.g., `CURPOS`, `N4`, `N(N2+ 1)`
- Parenthesized numeric statement, e.g., `(N1 = CURPOS + 1)`

The operators are the usual set of arithmetic, logical, relational and shift operators. For example `(NO SHL 4) + 1`, shifts the variable `NO` four places to the left, then adds 1 to it.

- String expressions can be either:
 - String constants, e.g., "This is a constant"
 - Global string variables, e.g., `S2`, `SM`, `SB`, `S(N2+N3)`

Numeric Constants

The **calc** command numeric constants are integer numbers. They can be binary, octal, decimal, or hexadecimal. The **calc** command recognizes these constants by the suffix **B**, **O** (or **Q**), **D**, or **H**, respectively. Numbers without a suffix are considered decimal. A numeric constant may be in the range $-(2^{**31})$ to $+(2^{**31} - 1)$

String Constants

A string constant may be 0-60 characters long with same delimiter at both ends. The same delimiter means that there is no predefined string delimiter; rather, the character immediately to the left of the string constant is identified as a delimiter. Then the second occurrence of that character signifies the right end of the string. To prevent ambiguity, the following characters may not serve as string delimiters: letters, digits, blank, and tab.

A natural delimiter choice is a quotation mark. However, if the string constant includes a quotation mark, then a different character, one that does not appear in the string, should be used as the delimiter.

The case of the letters within the string is preserved.

Operators

Table 7-1 displays a functional grouping of operators. The groups are: logical operators, relational operators, shift/rotate operators, and arithmetic operators.

Table 7-1. Operators' Precedence and Associativity

Operator Class	Operator	Interpretation	Associativity
Parentheses	()	Controls evaluation order: expressions in parentheses are evaluated before the items in the parentheses	From inside to outside
Unary	+ - ~ ` ! #	Single positive operator, Single negative operator, 1's complement (~ or `) POS operator (!) NEG operator (#)	From right to left, e.g., !#3 is !(#3)
Power	**	Raising to the power of	From right to left, 3**4**5 is 3**(4**5)
Arithmetic (mul/div)	* / \	Multiplication, division, mod (remainder) div	From left to right, e.g., 11 * 12 * 13 is (11 * 12) * 13
Arithmetic (add/sub)	+ -	Addition, subtraction	From right to left e.g., 2-3+4 is (2-3)+4
Shift/rotate	SHL, SHR, SAL, SAR, ROL, ROR	Shift left, shift right, shift algebraic left, shift algebraic right, rotate left, rotate right	From left to right, e.g., 5 SAR 1 SAR 1 is (5 SAR 1)SAR 1
Relational	< <= == <> => >	Less than, less than or equal to, equal to, not equal to, greater than or equal to, greater than	From left to right, e.g., 3<4<2 is (3 < 4) < 2
Logical	& ^	AND, OR, XOR	From left to right, e.g., 5 & 3 ^ 7 is (5 & 3) ^ 7

Table 7-1 lists all the **calc** command operators with a brief description of the semantics of each operator. These are more detailed descriptions of the nontrivial operators:

- \sim and $'$ (1's complement) have the same meaning. The duplicate notation prevents possible difficulties on terminals where one of these characters may have a special meaning. 1's complement means for every 0 bit, a 1 bit is returned and vice versa.
- $!$ and $\#$:
The POS operator ($!$) is defined as follows:
If number ≥ 0 then return true (-1)
else return false (0).
The NEG operator ($\#$) is defined as follows:
If number ≥ 0 then return false (0)
else return true (-1).
- \backslash (modulo division) returns the remainder of an integer division, for example, $7 \backslash 4 = 3$; $16 \backslash 4 = 0$.
- \wedge (XOR) returns true only if one operand is true and the other is false; otherwise it returns false. This is done for each bit in the argument, for example, $5 \wedge 1 = 4$.

Shift/ Rotate Operators

In the shift/rotate operations, the left operand is handled as a pattern of 32 bits. It is moved to the right or to the left by the number of bits specified by the right operand.

In a shift, bits moved off one end of the pattern are lost, and 0 bits or 1 bits are moved into the pattern from the other end. In a rotate, bits moved off of one end move onto the other end.

SAL and SAR are algebraic shift operators. This means that the high order bit is the sign bit, and there is no shift of bits between the sign and the rest of the number. In a left shift (SAL), 0 bits move into the pattern from the right. In a right shift (SAR), either 0 bits (if pattern is positive) or 1 bits (if pattern is negative) move into the pattern from the left.

In every shift/rotate operation, the right operand (**count**) is always taken as modulo 256.

Expression Evaluation

Operators in the **calc** command have an implied order that determines how operands and operators are grouped and evaluated.

Table 7-1 lists the **calc** command operators from highest to lowest precedence (i.e., those that take effect first are listed first). Operators in the same line are of equal precedence.

The evaluation order is the same as that used in most programming languages. It is controlled first by parentheses, then by operator precedence, and finally by operator associativity. The **calc** command first evaluates operands and operators enclosed in paired parentheses as subexpressions, working from innermost to outermost parentheses pairs. The value of the subexpression is then used as an operand in the remainder of the expression.

The precedence and associativity rules are demonstrated by these examples.

These two statements are equivalent:

```
n1 + n2 * n3 ** - n4
n1 + (n2 * (n3 ** (-n4)))
```

These two statements are equivalent:

```
n1 SHR 3 ≥ n2 SHL n7 + 4 * #2
(n1 SHR 3) ≥ (n2 SHL (n7 + (4 * (#2))))
```

These two statements are equivalent:

```
n2 + 3 < > n4 - 5 & n4 - 2 == n5
((n2 + 3) < > (n4 - 5)) & ((n4 - 2) == n5)
```

Upper case and lower case letters are not distinguished from each other except in string constants. For example, *curpos* and *CURPOS* are interchangeable.

The value TRUE is represented by -1; i.e., 0FFFFFFFH, and the value FALSE is represented by 0. Whenever an operator produces a logical (Boolean) value, that value is either TRUE or FALSE.

Examples

The two examples show how to use the N and S variables.

Example 1

```
S3
```

The value of the S variable S3 is displayed at the message line; no assignment.

Example 2

```
n1 = n(n(n2 = 2* n3))
```

Assuming that N3 contains 3 and N6 contains 8, and all other N variables are 0. Then $2 * N3 = 6$; therefore, N2 gets 6, and N6 is the index for the "outer" N because N (6) is equivalent to N6, thus giving $N1 = N(N6)$. N6 contains 8; thus $N1 = N8$. Because N8 contains 0, N1 gets this value (0), and 0 is displayed on the message line.

Errors

The **calc** command returns these messages. When an error is detected, the corresponding error message is displayed on the MESSAGE line, followed by a portion of the command where the error was detected.

Divide by zero error

An attempt was made to divide by zero.

Expression too complex

The expression is too complex; simplify the expression.

Floating point not allowed

Real values (e.g., 5.2) are not allowed.

Illegal exponential operation

Usually occurs when a negative value is used as the right operand. The illegal exponential expression is displayed on the message line. Correct it and rerun **calc**.

Illegal expression

The illegal expression is displayed on the message line. Correct it and rerun **calc**.

Invalid base character

The base character is not valid, e.g.,18A.

Invalid numeric constant

The numeric constant is not valid, e.g.,1AD.

MOD by zero error

An attempt was made to take MOD with zero.

Numeric constant too large

A numeric integer constant must be in the range $-(2^{**31})$ to $+(2^{**31} - 1)$.

Unbalanced parenthesis

Either the right or left parenthesis is missing.

Unrecognized identifier

The illegal identifier is displayed on the message line. Correct it and rerun **calc**.



Macros give AEDIT great flexibility and power. You can write macros to incorporate the **calc** command and AEDIT variables. These macros can print dates, directories, use an on-line calculator, and convert letters from upper case to lower case and vice versa.

Several macros are included with AEDIT in the file *useful.mac*. You must have AEDIT version 2.0 or later to use this macro file. To activate these macros, type:

```
M(ACRO) G(et) USEFUL.MAC
```

However, you should append *useful.mac* to your default macro file (*aedit.mac*), or include the macros you find useful in your default macro file.

The macros in *useful.mac* occupy about 1900 bytes of the macro buffer. The default macro buffer size (3072 bytes) is usually sufficient for *useful.mac*. However, you may use the invocation control **macrosize** if you need to increase the macro buffer capacity.

The macros in *useful.mac* use the N variables N7, N8, and N9, and the S variables S0 and S9. Use the **calc** command to define N and S variables.

The *Useful.Mac* File

These macros are contained in *useful.mac*. They are listed by name with a short descriptive sentence.

Macro Name	Description
<Blank>	The space bar may be used in addition to the <Tab> key to scroll the menu.
.	Find the next occurrence of the target string from the last find/replace .
,	Find the previous occurrence of the target string from the last find/replace .
L	Convert the character the cursor is on to a lower case character.
U	Convert the character the cursor is on to an upper case character.
-	Convert the word starting at the cursor from upper case to lower case letters. Convert the word, starting at the cursor, from lower case to upper case letters.
<Ctrl-W>	Move the cursor right to the next word.
<Ctrl-K>	Move the cursor left to the previous word.
]	Set leftcol one position to the left.
[Set leftcol one position to the right.
}	Set leftcol three positions to the left.
{	Set leftcol three positions to the right.
+W	If a white space, skip to the right to the next nonwhite character.
+N	If a nonwhite character, skip to the right to the next white space.
+B	If a blank, skip to the right to the next nonblank.
+W	If a white space, skip to the left to the next nonwhite character.
+N	If a nonwhite character, skip to the left to the next white space.
+B	If a blank, skip to the left to the next nonblank.
DT	Insert the date in mmm dd, yyyy format (e.g., July 24, 1984)
DM	Insert the date in dd-mmm-yyyy format (e.g., 24-Jul-1984)
<Ctrl-B>	Insert the contents of the block buffer into the text at the cursor position.
PG	Page the text. Header is always written as Heading.

PP	Page the text. Header is stored in the S0 S variable.
	PG and PP are meant for paging text. These macros attempt to put headers on empty lines, if possible. They use tag C and tag D for internal computations.
CNTR	Center the text in the current line.
DETAB	Convert all tabs from the current position to the end of the file to blanks, using current tab setting.
ENTAB	Convert all blanks from the current position to the end of the file to tabs. This macro works very slowly.
SHL	Display the current line number.
SFL	Display the total number of lines in the file.
SFC	Display the total number of characters in the file.
SHP	Display the current position in the line.
C	On-line calculator. In insert or xchange modes you may enter an arithmetic expression. Press the key configured to mexec , then C, and the expression value is displayed at the cursor position.
SMP	Set the indentation and left and right margins according to the values for the paragraph in which the cursor is currently positioned.
NUM	Insert line number prefix on each line in a text file. The macro uses tag D for internal computations.
0	Set paragraph with indentation 0, left margin 0, right margin 70
2	Set paragraph with indentation 0, left margin 3, right margin 70
3	Set paragraph with indentation 3, left margin 3, right margin 70
4	Set paragraph with indentation 3, left margin 5, right margin 70
5	Set paragraph with indentation 5, left margin 5, right margin 70
6	Set paragraph with indentation 5, left margin 7, right margin 70
7	Set paragraph with indentation 7, left margin 7, right margin 70

Tips for Writing Macros

The techniques described in this section will help you to understand the macros in *useful.mac* and to write your own macros.

Sending Text to the Message Line

Sending a message to the message line is done using the **calc** command with an expression, rather than an assignment statement, as the argument. As stated in Chapter 7, when the argument is an expression, its value is output to the message line even if it is executed within a macro.

Suppose, for example, that `N9` contains the current line number, and you want to output this value, with an appropriate title, to the message line. This is done as follows:

```
C(alc) N9 = expr
C(alc) S9 = "current line: <FETN>9"
C(alc) S9
```

Or in macro form:

```
...CN9= expr \NLCS9="current line:
\XN9" \NLCS9\NL\MM
```

Simulate "IF *cond* THEN RETURN" (to the caller)

This construct is done by using the "fail" characteristic of the **find** command. Recall that searching an empty string always fails (assuming that **set go no** is in effect), but searching zero times always succeeds, regardless of the operand. The following method may thus be used:

```
C(ALC) N9= cond \ * cond is any logical expression * \
<fetcn>9 F(ind) ~<rout> <ESC>
\ * Find argument is an empty string * \
...
```

Or in macro form:

```
...CN9= cond \NL\XN9F~\RB\BR...
```

Note that *cond* must be a logical expression, so that it will have the value of 0 (false) or -1 (OFFFFFFFFFH, true). When it is used as a count, the absolute value, 0 (false) or 1 (true) is used.

Simulate "IF *cond* THEN *statement*"

In this case, **statement** is implemented as a separate macro (named **statement**). This macro will be executed 0 or 1 times depending on the value of *cond*:

```
C(alc) N9= cond
<FETN>9 E(XECUTE) statement
```

Or in macro form:

```
...CN9= cond\NL\XN9E statement\NL...
```

Note again that *cond* must be a logical value to ensure that N9, when used as a count, is either 0 or 1.

Simulate "IF *cond* THEN *statement1* ELSE *statement2*"

This is based on the example given above:

```
C(alc) N8=!(N9=cond) \*!" is the NEG operator * \
<FETN>9 E(XECUTE) statement1
<FETN>8 E(XECUTE) statement2
```

Or in macro form:

```
...CN8=!(N9=cond)\NL\XN9E statement1\XN8E statement2\NL...
```

Note that *statement1* may not change N8; if it does, *statement2* may be executed unintentionally.

Simulate "Advance_While *cond*"

This construct is usually needed for macros like skip to next blank, skip to next word, etc. This is done using the "IF *cond* THEN RETURN" method described above. *Cond* in this case must be a logical expression that involves the current cursor location (e.g., ISDEL, CURCH=20H). The method consists of two nested macros. The "low level" macro advances one character and fails when the condition is not met (named **advance_one**). The "main" macro executes the first one an infinite number of times and actually terminates when the condition is not met, and continues with the next instruction:

```
Advance_While:.../E(xecute)Advance_One...
Advance_One:C(alc) N9=!cond
<fetn>9 F(ind)~<rubout><ESC>
<RIGHT>
```

Or in macro form:

```
MAdvance_While\BR.../EAdvance_One\NL...  
MAdvance_One\BRCN9=!cond\NL\XN9F~\RB\BR\CR...
```

If, for example, you want to implement skip to the next blank/tab, then *cond* is CURCHC< >20H & CURCHC< >09H.

To implement **Backward_While** (*cond*), the same method is used, but the last command in **advance_one** should be <LEFT> (\ CL) instead of <RIGHT> (\ CR).

Examples

The examples included in this section are macros from *useful.mac*. They are explained in greater detail to show the usage of the above techniques.

1. The following set of macros converts single letters or words from upper case to lower case or vice versa. It executes nested macros (e.g., macro **L** calls macro **U2L**), uses the **calc** command (e.g., C(alc) n8=(n9=lowch) and read-only variables (e.g., *lowch*, *curch*, *upch*), calls the **set** command (S(et) R(adix) A) and the fetch function (e.g., <fetr>8).

```
ML\BReu21\NL\CR\XN8eLU11\BR\MM;      \* letter to lower case *\  
MU\BRe12u\NL\CR\XN8eLU11\BR\MM;      \* letter to upper case *\  
M_\BRE+W\NL/e112\NLe+W\BR\MM;        \* word to lower case *\  
MÛ\BRE+W\NL/eu12\NLe+W\BR\MM;        \* word to upper case *\  
MU2L\BRCN8=(N9=lowch)<>curch\NL\MM;  
ML2U\BRCN8=(N9=upch)<>curch\NL\MM;  
MLU11\BR\CLsrax\XN9\BR\MM;  
ML12\BRCN7=iswhte \NL\XN7f~\RB\BR1\MM  
MU12\BRCN7=iswhte \NL\XN7f~\RB\BRu\MM;
```

Note the use of the following techniques:

- The L Macro executes **LU11** using the IF *cond* THEN *statement*
- The L12 Macro simulates **advance_while** the current character is not a white space. In this case, it also converts the character to lower case.
- The “_” (underscore) Macro uses +B to skip to the next nonwhite space character.

- The <Ctrl-W> Macro moves the cursor one word to the right. A word (in this case) is defined as a sequence of characters enclosed by delimiters. Delimiters are defined as white spaces or the user defined delimiters (listed under **set E_delimiter**). The technique IF *cond* THEN RETURN is used here. In the nested macros, N9 defines whether or not the cursor is on a delimiter. If the value fetched by <fetn>9 is 0 (false), the **find** succeeds and the macro is repeated. If the value fetched by <fetn>9 is 1 (true), the **find** fails and execution returns to the calling macro.

```
M\017\BR0f~\RB\BR/e\0171\NL/e\0172\NL\MM;
\ * word right macro * \
M\0171\BRcn9=iswhte\NL\XN9f\BR\CR\MM;
M\0172\BRcn9=! isdel\NL\XN9f\BR\CR\MM;
```

The <Ctrl-K> Macro differs from the <Ctrl-W> Macro only in that it moves the cursor one word to the left.

```
M\00B\BR0f~\RB\BR\CL/e\0111\NL/e\0112\NL\CR\MM;
\ * word left macro * \
M\0112\BRcn9=iswhte\NL\XN9f\BR\CL\MM;
M\0111\BRcn9=! isdel\NL\XN9f\BR\CL\MM;
```

Note that the macros are optimized to some extent. A **find empty string** is issued first; therefore, a future **find** command uses <Esc> as an argument, not the sequence ~**rubout**<Esc>.

- The CNTR Macro centers the text on the line. This macro strips all blanks from the left side, then all blanks from the right side. It calculates the number of blanks to be added and adds the blanks from the left margin to the first character so that the text is centered on the line. Skipping blanks is done with the **+B** and the **-B** macros. The number of blanks to be added is calculated using the read-only variables *rmargn*, *lmargn*, and *col*. **+B** and **-B** use **advance_while** macro techniques.

```
MCNTR\BRjp0\NLe+b\NL\XXjp254\NLi\BR\CLe-b\NL\CR
\XA cn9=(rmargn+1+lmargn-col)/2\NLcn9=-n9*(n9>0)\NLjp0\NLi\BRb
\CLb\XF\XN9g\NL\NL\MM;

M+B\BR/e+B1\BR\MM;
M+b1\BRcn9!=(curch=20H)\NL\XN9f~\RB\BR\CR\MM;

M-B\BR/e-B1\BR\MM;
M-b1\BRcn9!=(curch=20H)\NL\XN9f~\RB\BR\CL\MM;
```



Introduction

AEDIT is designed to run with various terminals. In some cases, for example VT100, AEDIT is able to identify the host environment. In other cases, you should specify the characteristics of your particular terminal.

For the iRMX version of AEDIT, specify the terminal characteristics with configuration commands in the `:config:termcap` file. The parameters and control sequences are listed in Table 9-1.

A configuration command must be terminated with a semicolon (;) or a carriage return. For some terminals, such as the Series-III, Series-IV and ANSI/VT100 family of terminals, only the appropriate hardware identification configuration command, `AH=string`, is required. Appendix D lists configurations for several non-Intel terminals. If your terminal is not included in the list of tested terminals, refer to your terminal user manual for the terminal characteristics and add the appropriate commands to your `:config:termcap` file.

AEDIT requires that the terminal meet the following conditions:

- ASCII codes 20H - 7EH display some symbol that requires one column space. Carriage return (ODH) and linefeed (OAH) perform their usual functions.
- The following cursor functions have cursor key input codes and CRT cursor output codes: **left**, **right**, **up**, **down**, **home** and **carriage return**. Output codes such as **clear screen**, **clear rest of screen**, **clear line**, **clear rest of line**, and **direct cursor addressing** are desirable for faster screen plotting, but not required. The codes, shown in Table 9-1, can be changed with the configuration commands.
- The CRT accepts a blankout code that blanks out the former contents of the screen location to which it is output. The default, 20H, can be changed with the configuration commands.
- AEDIT automatically generates a linefeed each time a carriage return is entered. Your terminal should not transmit a linefeed with a carriage return. In some terminals, this feature can be switched on and off.

Table 9-1 lists the configuration commands and their meanings. These commands are divided into three groups:

- Terminal attributes and generals
- Input codes, which specify codes sent from the keyboard to the terminal (i.e., AEDIT)
- Output codes, which specify codes to be sent from the terminal (i.e., AEDIT) to the display.

A configuration command may be used to set a value to a specific feature or to indicate that the feature is not available. To indicate that the feature is not available, the command is specified without an associated value, for example, `AFER=;`

Configuration Command Notes

AS It is highly recommended to set this feature off (`AS=F`), if one of the following is true:

- Your terminal does not have the direct cursor addressing feature (e.g., `AH=S3`, the terminal is a SERIES-II terminal).
- If your terminal operates at a relatively low baud-rate, less than 9600 baud.
- If the **AT** configuration command is off (`AT=F`). In this case, setting `AS=T` causes the busy/waiting indicator to toggle for each input character.

In all the cases listed above, setting `AS=T` degrades performance due to slow output or huge amounts of output to the screen.

AT Setting `AF=F` directs AEDIT to not support type-ahead. When **AT** is off (`AT=F`), the **AFCC** function is not fully functional. The synonym works only for synchronous operations (e.g., to terminate **insert** mode), but not for asynchronous operations (e.g., exit from a loop within a macro).

AB The natural choice is **ESC** (1BH). However, this value cannot be used if other terminal input function begin with **ESC** (e.g., VT100).

AFCC **AFCC** is not fully functional if **AT** is off (`AT=F`). See notes on **AT** above.

Table 9-1. Configuration Commands

Terminal Attributes and Generals	
Command	Meaning
AH= <i>string</i>	<p>Hardware Identification: where <i>string</i> is one of the following values, implying a set of configuration commands</p> <p><null> Equivalent to specifying the minimal default set, which is specified with AH =;. The default set is not sufficient for interactive use and must be completed with other explicit configuration commands.</p> <p>S3 Series III systems. Equivalent to specifying all configuration commands with the Series III default values.</p> <p>S3E Equivalent to S3 with the following changes:</p> <ul style="list-style-type: none"> - New console output functions, including: <ul style="list-style-type: none"> Various clear text functions Direct cursor addressing Local scrolling - Fast block move command for data to the CRT <p>S3ET Equivalent to S3E without the fast-block-move command for data to the CRT. S3ET is used when the Series III E is used as a terminal.</p> <p>S4 Series IV systems. Equivalent to specifying all configuration commands with the Series IV default values.</p> <p>ANSI Equivalent to specifying all the configuration commands with the default values according to ANSI X3.64 (1977).</p> <p>VT100 Equivalent to specifying all the configuration commands with the default values for the DEC VT100 family of terminals.</p>
AV=n (5:66)	Sets the number of rows in the display.
AW=T or F	True if the terminal wraps when a character is printed in the last physical column.
AS=T or F	True to display the busy/waiting indicator.
AT=T or F	True if type-ahead is to be done by AEDIT.
AG=T or F	True if bell signal is to be issued along with warning messages (e.g., rewriting files on quit write).

Notes:

- n*(*n1:n2*) an integer in the inclusive range, *n1* to *n2*.
- h* a 1-byte hexadecimal number.
- hhh* a 1-byte to 4-byte hexadecimal number.
- hhh...* the same as *hhh*, but the length may be up to 40 bytes.
- T *T* or *t* indicates true.

F *F* or *f* indicates false.
string a 0- to 60-character string.

Table 9-1. Configuration Commands (continued)

Output Codes	
Command	Meaning
Cursor moves:	
<i>AFMB=hhhh</i>	Moves cursor to start of line
<i>AFML=hhhh</i>	Moves cursor left
<i>AFMR=hhhh</i>	Moves cursor right
<i>AFMU=hhhh</i>	Moves cursor up
<i>AFMD=hhhh</i>	Moves cursor down
<i>AFMH=hhhh</i>	Moves cursor home
Erase:	
<i>AFES=hhhh</i>	Erases entire screen
<i>AFER=hhhh</i>	Erases rest of screen
<i>AFEK=hhhh</i>	Erases entire line
<i>AFEL=hhhh</i>	Erases rest of line
Cursor addressing:	
<i>AFAC=hhhh</i>	Addresses cursor lead-in. When used, code will be followed by column number (0 to <i>max_col_value</i>) and row number (0 to <i>max_row_value</i>).
<i>AO=h</i>	Offset to add to both row and column number with address cursor commands.
<i>AX= T or F</i>	True if <i>x</i> (column) precedes <i>y</i> (row) in address cursor command.
Delete/insert:	
<i>AFIL=hhhh</i>	Insert line code. Used in line 0 for reverse scrolling.
<i>AFDL=hhhh</i>	Delete line code. Used to speed up display on the Hazeltine 1510 and similar terminals.
Reverse video:	
<i>AFRV=hhhh</i>	Start reverse video character(s). Used on the PROMPT line display.
<i>AFNV=hhhh</i>	Return to normal video characters. Used to restore the display after a reverse video line.
<i>AI= T or F</i>	True if the CRT has invisible attributes, or False if the attribute occupies a position on the screen.
<i>AC= T or F</i>	True if the CRT has attributes per character, or False if the attributes are per field.
Initialization/termination:	
<i>AFST=hhhh...</i>	Start-sequence. This sequence is output to the terminal when AEDIT encounters it. Can be used to initialize the terminal, e.g., to initialize a scrolling region.
<i>AFEN=hhhh...</i>	End-sequence. Output to the terminal when AEDIT exists. Can be used to unset values that have been set by AFST.

Table 9-1. Configuration Commands (continued)

Input Codes	
Command	Meaning
Escape keys:	
<i>AB=hhhh</i>	Sets <ESC>
<i>AFCC=h</i>	Sets <CTRL-C> synonym
Cursor move keys:	
<i>AFCL=hhhh</i>	Sets <LEFT>
<i>AFCR=hhhh</i>	Sets <RIGHT>
<i>AFCU=hhhh</i>	Sets <UP>
<i>AFCD= hhhh</i>	Sets <DOWN>
<i>AFCH=hhhh</i>	Sets <HOME>
Delete keys:	
<i>AR=hhhh</i>	Sets rubout
<i>AFXF=hhhh</i>	Sets delete character-delch
<i>AFXX=hhhh</i>	Sets delete left-dell
<i>AFXA=hhhh</i>	Sets delete right-delr
<i>AFXZ=hhhh</i>	Sets delete line-delli
<i>AFXU=hhhh</i>	Sets undo
Prefix keys:	
<i>AFXE=hhhh</i>	Sets macro exec key- mexec
<i>AFXH=hhhh</i>	Sets hex character- hex
<i>AFXN=hhhh</i>	Sets fetch numeric-fetn
<i>AFXS=hhhh</i>	Sets fetch string-fets
Others:	
<i>AFIG=h</i>	Sets character(s) to be ignored. This specification is needed on terminals (such as the Hazeltine 1510) that have multiple character key codes for up and down . <i>AFIG</i> should be set to the lead-in (tilde), and up and down should be set to the second letter of the cursor up or down key code. This avoids problems caused by the lack of a type-ahead buffer.

Notes:

- n(n1:n2)* an integer in the inclusive range, *n1* to *n2*.
- h* a 1-byte hexadecimal number.
- hhhh* a 1-byte to 4-byte hexadecimal number.
- hhhh...* the same as *hhhh*, but the length may be up to 40 bytes.
- T* *T* or *t* indicates true.
- F* *F* or *f* indicates false.
- string* a 0- to 60-character string.

Configuration Values

An **AH** command value implies a complete set of values for the entire configuration command set. Table 9-2 lists the default configurations for various terminals or systems.

Delay Codes

Some CRTs are too slow with respect to some AEDIT output functions. To enable a smooth operation of AEDIT with these CRT types, AEDIT should be informed how long it has to wait before it may issue a new output operation.

You can specify delay codes for the various output functions with configuration commands of the form **AD_{xx}=*n***, where **xx** is the function code in the corresponding **AF_{xx}** code, and *n* is a decimal number specifying the delay in milliseconds. For example, **ADDL=30** defines a delay of 30 milliseconds for the function **AFDL** (delete line). An **AD_{xx}** value may be specified for every **AF_{xx}** output function.

Determining the Configuration Values

Table 9-2 lists the default configuration values. AEDIT processes configuration commands in this order:

1. Sets the default value for each configuration command.
2. Checks to see if the system is one of the default types listed in Table 9-2. If so, AEDIT sets the configuration as defined in Table 9-2.
3. Processes configuration commands in the *:config:termcap* file. For every valid command under your terminal type, AEDIT overwrites the previous value with the new one.
4. Checks the configuration to make sure these values are defined:

AFMB	AFMD
AFMH	AFML
AFMU	AFMR

⇒ **Note**

If any of the values in step 4 are undefined, AEDIT terminates and displays the following message:

Table 9-2. Configuration Default Values

Command	AH=; Default	AH=S3 (S III)	AH=S3E (S III E)	AH=S4 (S IV)	AH=ANSI AH=VT100	Meaning
Input Codes						
AV=	24	25	25	25	24	Screen length
AW=	T	T	T	T	T	Terminal wraps?
AS=	F	F	T	F	F	Display busy/ waiting indicator
AT=	T	T	T	T	T	Type-ahead
AG=	T	T	T	T	T	Bell signal
Escape keys:						
AB=	1B	1B	1B	1B	1B4F53 ⁽²⁾	Sets <ESC>
AFCC=	03	03	03	03	03	Sets <CTRL-C>
Cursor move keys:						
AFCL=	-- ⁽¹⁾	1F	1F	89	1B5B44	Sets <LEFT>
AFCR=	--	14	14	8A	1B5B43	Sets <RIGHT>
AFCU=	--	1E	1E	87	1B5B41	Sets <UP>
AFCD=	--	1C	1C	88	1B5B42	Sets <DOWN>
AFCH=	--	1D	1D	81	1B4F50 ⁽³⁾	Sets <HOME>
Delete keys:						
AR=	7F	7F	7F	7F	7F	Sets rubout
AFXF=	06	06	06	80	06	Sets delch
AFXX=	18	18	18	18	18	Sets dell
AFXA=	01	01	01	01	01	Sets delr
AFXZ=	1A	1A	1A	82	1A	Sets delli
AFXU=	15	15	15	15	15	Sets undo
Others:						
AFIG=	--	--	--	--	--	Ignore character
Prefix keys:						
AFXE=	05	05	05	05	05	Sets mexec
AFXH=	12	12	12	12	12	Sets hex
APXN=	0E	0E	0E	0E	0E	Sets fetn
AFXS=	16	16	16	16	16	Sets fets

Notes:

- means the feature is either unavailable or meaningless.
- Because 1B is used as a prefix for input sequences on ANSI terminals, it may not be used as <ESC>. The choice of the FP4 key (for AH=VT100) or 04H (for AH=ANSI) is arbitrary and maybe changed.
- In the absence of a "natural" <HOME> key, the choice of the FP1 key (for AH=VT100) or 0CH (for AH=ANSI) is arbitrary and may be changed.

Table 9-2. Configuration Default Values (continued)

Command	AH=; Default	AH=S3 (S III)	AH=S3E (S III E)	AH=S4 (S IV)	AH=ANSI AH=VT100	Meaning
Output Codes						
Cursor moves:						
AFMB=	0D	0D	0D	0D	0D	Carriage return
AFML=	--	1F	1F	08	1B5B44	Cursor left
AFMR=	--	14	14	1B43	1B5B43	Cursor right
AFMU=	--	1E	1E	1B41	1B5B41	Cursor up
AFMD=	--	1C	1C	1B42	1B5B42	Cursor down
AFMH=	--	1D	1D	1B48	1B5B48	Cursor home
Erase:						
AFES=	--	1B45	1B45	1B45	1B5B324A	Entire screen
AFER=	--	1B4A	1B53	1B4A	1B5B4A	Rest of screen
AFEK=	--	1B4B	1B4B	0D1B4B	1B5B324B	Entire line
AFEL=	--	--	1B52	1B4B	1B5B4B	Rest of line
Cursor addressing:						
AFAC=	--	--	1B59	1B59	1Bx; y48 ⁽⁴⁾	Address lead in
AO=	--	--	20	20	--	Row/column offset
AX=	--	--	F	F	--	X (column) before Y (row)
Delete/insert:						
AFIL=	--	--	1B57603F	--	1B5B4C ⁽⁵⁾	Insert line
AFDL=	--	--	1B573F60	--	1B5B4D ⁽⁵⁾	Delete line
Reverse video:						
AFRV=	--	--	1B4C90	1B7C50	1B5B376D	Reverse video
AFNV=	--	--	1B4C80	1B4C40	1B5B6D	Normal video
AI=	--	--	F	F	T	Invisible attributes
AC=	--	--	F	F	T	Character attributes
Initialization/termination:						
AFST=	--	--	--	--	--	Start-sequence
AFEN=	--	--	--	--	--	End-sequence

Notes:

4. The ANSI escape sequence for cursor addressing is hard-coded in AEDIT. The table shows the format only. This format cannot be coded using AFAC.
5. Insert/Delete line functions, although available on the VT100, are disabled because of poor performance.



AEDIT Command Summary

A

This appendix lists the AEDIT commands and subcommands, with their formats and functions. Angle brackets (< >) indicate a key configured for a function, and <Ctrl> represents the Control key.

Function Keys

Execute these commands by pressing the specifically labeled key on the keyboard or by typing the indicated <Ctrl> commands. Some function keys are configurable. If so, the default is shown; if not, N/A is shown. These functions are also available, where applicable, in line editing.

Table A-1. Function Keys

Key	Default	Function
<Left>	N/A	Left arrow, moves the cursor left.
<Right>	N/A	Right arrow, moves the cursor right.
<Up>	N/A	Up arrow, moves the cursor up.
<Down>	N/A	Down arrow, moves the cursor down.
<Home>	N/A	Allows fast cursor movement, permits entry into reedit mode.
rubout	backspace (upper back- arrow)	Deletes the character to the left of the cursor at the main command level or insert mode. In xchange mode, rubout exchanges the new character to the left of the cursor with the original character.
delch	<Ctrl-F>	Deletes the character that the cursor is on.
delli	<Ctrl-Z>	Deletes the entire line on which the cursor is positioned.
dell	<Ctrl-X>	Deletes all characters to the left of the cursor.
delr	<Ctrl-A>	Deletes all characters to the right of the cursor.
undo	<Ctrl-U>	At the cursor position, restores the characters deleted by the last dell , delr , or delli command.

Table A-1. Function Keys (continued)

Key	Default	Function
<Esc>	<Esc>	Terminates line editing input and sends the entire string, or exits the mode and returns the editor to the main command level.
<Ctrl-C>	<Ctrl-C>	Aborts the command in progress and returns the editor to the main command level.
<CR>	N/A	<ol style="list-style-type: none">1. Moves the cursor to the start of the next line.2. In (-)find/(?)replace commands, adds a line terminator character to the target string, displaying <n1>.3. In line editing input, terminates the line at the cursor position and sends the string to the left of the cursor to be processed by the command.
<Tab>	N/A	Rotates the menu prompt line to display the next line of commands or, in insert or xchange mode, inserts tab or equivalent number of blanks.
hex	<Ctrl-R>	Inserts a character in the text as its ASCII value. Should be followed by two digits that are interpreted as the hexadecimal value.

AEDIT Editing Commands

This table lists the AEDIT editing commands, with their subcommands, formats and functions. The main commands are in alphabetical order.

Table A-2. AEDIT Editing Commands

Command	Format	Function
again	[count] A	Repeats the last command or subcommand.
block	B or D	Delimits a section of text that can then be deleted, moved, or copied. Has the following subcommands:
buffer	B	Copies text to the Block buffer.
delete	D	Deletes the delimited section and moves it unchanged to the Block buffer.
find	F	Same as in the main command level.
-find	-	Same as in the main command level.
jump	J	Same as in the main command level.
put	P	Copies the delimited section of text to a specified output file.
calc	C	Provides computing capabilities.
delete	B or D	Delimits a section of text that can then be deleted, moved, or copied. (Same as in block command.)
execute	[count] E	Executes the specified macro.
find	[count] F	Searches forward from the current cursor position for string. Moves the cursor if found.
-find	[count] -	Searches backward from the current cursor position for string. Moves the cursor if found.
get	[count] G	Retrieves the contents of the Block buffer or external file; places the contents at the current cursor position. Count must be a number less than 64K.
hex	[count] H	Hex command
input	I	Inserts the ASCII equivalent of hexadecimal values in text.
output	O	Displays hexadecimal values of ASCII characters in the message line.

Table A-2. AEDIT Editing Commands (continued)

Command	Format	Function
insert	[/] I	Begins insert mode; inserts text at the cursor position.
jump	J	Moves the cursor to a specified location in text.
A_tag	A	Moves the cursor to tag A.
b_tag	B	Moves the cursor to tag B.
c_tag	C	Moves the cursor to tag C.
d_tag	D	Moves the cursor to tag D.
start	S	Moves the cursor to the start of the file.
end	E	Moves the cursor to the end of the file.
line	L	Moves the cursor to the start of the designated line.
position	P	Moves the cursor to the designated position in the current line.
kill_wnd	K	Deletes the secondary window and extends the current window. The cursor remains at its current position.
macro	M	Allows you to manipulate macros with the following subcommands:
create	C	Creates macros interactively by accumulating a sequence of keystrokes.
get	G	Retrieves macros from an external file or from the current text buffer.
insert	I	Inserts subsequent input in text in macro form.
list	L	Lists the names of all currently defined macros on the message line.
save	S	Translates macros to macro form and inserts the definition at the current position in the text.
other	O	Switches between the main and OTHER files.
paragraph	[count] P	Reformats the paragraph using the values for indentation, left and right margins set with the set margin command.
fill	F	The paragraph is reformatted with no right-justification
justify	J	The paragraph is reformatted with right-justification.
quit	Q	Ends, updates, restarts, etc., the editing, depending on which subcommands are used.
abort	A	Returns to the operating system; all changes are lost.
exit	E	Returns to the operating system; the file is updated.
init	I	Restarts editing; initializes a new file without returning to the operating system.
update	U	Updates the file without returning to the operating system.
write	W	Writes the file to the output file specified without returning to the operating system.

Table A-2. AEDIT Editing Commands (continued)

Command	Format	Function
replace	[count] R	Searches forward for target string; replaces it with new string if found.
?replace	[count]?	Conditional replace command.
set	S	Sets several AEDIT features, with the following subcommands:
autonl	A	While in insert mode, inserts a new line in the text automatically when the line is full (default = no).
bak_file	B	Creates a backup file of the file being currently edited when quit update or quit exit is executed (default=yes).
case	C	Tells the editor to consider case of strings during (-)find and (?)replace commands (default=no).
display	D	Displays any movements in or changes to the text during macro execution (default=no).
e_delimit	E	Defines the token delimiters (default = ! " # % & ' () * + , - . / : ; < = > ? @ [\] ^ ` { } ~).
go	G	Continues macro execution even if a (-)find/(?)replace command in the macro fails (default=no).
highbit	H	Displays all text characters with hexadecimal values over 7FH as-is instead of ? (default=no).
indent	I	Indents inserted/exchanged text automatically (default = no).
k_token	K	A string in the text needs to be a token string to be found (default=no).
leftcol	L	Enables you to view lines over 80 characters long (default =0).
margin	M	Sets indentation, and left and right margins used by the paragraph and set autonl commands (default: indent=4, left=0, right =76).
notab	N	Inserts blanks in place of tabs in insert or xchange mode (default=no).
radix	R	Sets the radix by which a numeric variable is output by fetn in insert or xchange mode (default = decimal). Alpha A Binary B Decimal D Hex H Octal O
showfind	S	Lists target-string lines of multiple search commands (default = no).

Table A-2. AEDIT Editing Commands (continued)

Command	Format	Function
set subcommands (continued)		
tabs	T	Sets tab positions (default = 4).
viewrow	V	Sets the row to which text is moved relative to the screen on view command (default= row 5).
tag	T	Specifies locations in a file; used with the jump command. These are valid tags:
a_tag	A	
b_tag	B	
c_tag	C	
d_tag	D	
view	V	Rewrites (moves) text on the screen leaving the cursor in viewrow (default= row 5).
window	W	Splits the text area of the screen in two, enabling the user to look at two different parts of the same file or two different files.
xchange	X	Enters xchange mode; replaces characters on a one-for-one basis.
!system	!	Allows executing system commands from within AEDIT, executed by pressing the exclamation point (!).

□ □ □

AEDIT Error Messages

B

This appendix lists the error messages reported by AEDIT when a problem is encountered in the invocation line, an editing command, the **calc** command, or a macro file.

Invocation Errors

When an error occurs in the invocation line, AEDIT displays the sign-on message followed by the error message, and control returns to the operating system.

Invocation errors have the following form:

```
***ERROR:      illegal invocation, NEAR:      token
```

or

```
***ERROR:      message
```

Where:

token is an invocation command specification.

message is one of the following error messages.

Conflicting controls

An illegal combination of **viewonly** and **forwardonly** controls is used.

Insufficient configuration commands

The interactive session has been initialized; however, required configuration commands of the type **AFMx** or **AFBK** are undefined.

Insufficient memory

AEDIT does not have a large enough RAM partition.

Macro buffer too large

The macro buffer size specified is too large, and the buffer size remaining is insufficient for the text to be edited.

Macro buffer too small

The macro buffer size specified is less than the minimum allowed.

Editing Command Errors

`bad indent margin`

Attempt to set indent margin out of the legal range. Editor returns to main command level.

`bad left margin`

Attempt to set left margin out of the legal range. Editor returns to main command level.

`bad margins`

Attempt to set margins out of the legal range. Editor returns to main command level.

`bad right margin`

Attempt to set right margin out of the legal range. Editor returns to main command level.

`bad tabs`

Attempt to set bad tabs; e.g., 4,2 is illegal since the second value is less than the first. Editor returns to main command level.

`bad Leftcol`

Attempt to set bad left column, **leftcol** accepts any number from 0 to 176. Editor returns to main command level.

`bad Viewrow`

Attempt to set bad **viewrow**. This value must be between 0 and (text size -1). Editor returns to main command level.

`block buffer too large for SB`

Attempt to specify **fets** B or SB when the current **block** buffer is greater than 60 characters.

`cannot delete more than 32`

Attempt to use *count* greater than 32 with **delch** command. Editor returns to main command level.

`illegal command`

Attempt to enter illegal and/or unknown command. Editor ignores command.

`illegal invocation`

Attempt to invoke AEDIT with an illegal invocation line, or an illegal invocation under **quit init**.

`insufficient terminal capabilities`

Attempt to set windowing on a terminal that does not have **AFEK** or **AFEL** output functions. Editor returns to main command level.

`invalid hex value`

Attempt to input an invalid hexadecimal value. Editor returns to main command level.

invalid variable name
Attempt to input an invalid value for **fets** or **fetn**.

macro creation is forbidden while executing a macro
Attempt was made to define a macro while a macro was being executed.

macro nesting too deep
Nesting level for macros exceeded. A maximum of eight levels is allowed.

no more room for macros
Macro buffer is full.

no such macro
Attempt to execute macro that does not exist. Editor returns to main command level.

not found: "target string"
Target string not found. Editor returns to main command level.

some text lost
In **forwardonly**, the file is larger than the allocated memory.

text does not fit
Attempt to edit a file that is too large under the **forwardonly** control. Editor returns to main command level.

filename (error message supplied by operating system)
An error occurs during a **quit exit**, **quit init**, **quit update**, **get**, or **block put** command. Editor returns to main command level.

xchange limit is 100
Attempt to exchange over 100 characters without restarting **xchange** mode. Editor remains in **xchange** mode.

window too small
Attempt to split screen with one window size less than five lines.

CALC Command Errors

The following messages are issued only under the **calc** command. When an error is detected, the corresponding error message is displayed on the message line, followed by a portion of the command where the error was detected.

Divide by zero error

An attempt was made to divide by zero.

Expression too complex

The expression is too complex; simplify the expression.

Floating point not allowed

Real values (e.g., 5.2) are not allowed.

Illegal exponential operation

Usually occurs when a negative value is used as the right operand. The illegal exponential expression is displayed on the message line. Correct it and rerun **calc**.

Illegal expression

The illegal expression is displayed on the message line. Correct it and rerun **calc**.

Invalid base character

The base character is not valid; e.g., 18A.

Invalid numeric constant

The numeric constant is not valid; e.g., 1AD.

Mod by zero error

An attempt was made to take **mod** with zero.

Numeric constant too large

A numeric integer constant must be in the range $-(2^{31})$ to $+(2^{31}-1)$.

Unbalanced parentheses

Either the right or left parenthesis is missing.

Unrecognized identifier

The illegal identifier is displayed on the message line. Correct it and rerun **calc**.

Maro File Errors

If any error is found in a macro file, one of the following messages is printed. Macro file processing continues.

Macro errors have the following form:

```
Error in line nnn: <message>
```

Where:

nnn is the line number containing the error in the macro file.

<message> is one of the following error messages.

```
bad \ code
    Backslash (\) is not followed by a valid value.
bad AC type
    Illegal value configuration command.
bad AF type
    Illegal AF configuration command.
bad AH type
    Illegal value configuration command.
bad AI type
    Illegal value configuration command.
bad AS value
    Illegal value configuration command.
bad AT value
    Illegal value configuration command.
bad AV value
    Illegal value configuration command.
bad AW type
    Illegal value configuration command.
bad AX value
    Illegal value configuration command.
bad command
    Macro file contains a bad command_ unknown control code or character; i.e., not A,
    M, or S.
bad hex value
    Configuration command contains bad hex value, e.g., 3G.
missing '='
    A configuration command is missing an equal sign.
```

missing ';'

A configuration command is not terminated with a semicolon or line terminator.

no macro name

Macro definition does not include macro name.

no more room for macros

Attempt to create a macro when macro buffer is full. Macro definition is terminated.

□□□

Summary of AEDIT Variables

C

The following table summarizes the string, Boolean, and numeric variables that may be used in the various commands. Local variables can be used only within the **calc** command.

Table C-1. AEDIT Variables

Name	Type	Scope	Description
<i>bof</i>	Boolean	local	True if the cursor is at the beginning of the file.
<i>cntexe</i>	numeric	local	The number of times the macro that is currently executing has executed in the current activation. The first execution is number one. If none, the value is zero.
<i>cntfnd</i>	numeric	local	Relates to (-)find .
<i>cntmac</i>	numeric	local	The number of times that the last macro (which has finished executing) was executed.
<i>cntrep</i>	numeric	local	Relates to (?)replace .
<i>col</i>	numeric	local	The current logical cursor position in the line. (This value is not affected by the setting of leftcol .)
<i>curch</i>	numeric	local	ASCII value of the current character.
<i>curpos</i>	numeric	local	The offset of current location in file.
<i>curwd</i>	numeric	local	ASCII value of the two bytes at the current cursor location.
<i>date</i>	numeric	local	Date in decimal format MMDDYY.
<i>eof</i>	Boolean	local	True if the cursor is at the end of file.
<i>imargn</i>	numeric	local	The value of the current indent margin setting.
<i>inothr</i>	Boolean	local	True if you are in the OTHER buffer.
<i>isdel</i>	Boolean	local	True if the character at the current position is in the user-defined delimiter set.
<i>iswhite</i>	Boolean	local	True if the character at the current position is whitespace (space, tab, LF or CR).
<i>lmargn</i>	numeric	local	The value of the current left margin setting.

Table C-1. AEDIT Variables (continued)

Name	Type	Scope	Description
<i>ldwch</i>	numeric	local	If the current character is an uppercase character (41H to 5AH), <i>ldwch</i> is the ASCII value of the lowercase character. Otherwise <i>ldwch</i> is the same as <i>curch</i> .
<i>lstfnd</i>	Boolean	local	True if the last find or replace string was found.
<i>n 0-9</i>	numeric	global	Read-write <i>n</i> variables, assigned only in the calc command.
<i>nstlvl</i>	numeric	local	The nesting level of the currently executing macro; main command level is level zero.
<i>nxtch</i>	numeric	local	ASCII value of the next character.
<i>nxttab</i>	numeric	local	The column number of the next tab position (to the right of the cursor) as defined by set tab .
<i>nxtwd</i>	numeric	local	ASCII value of the second and third bytes following the current cursor location.
<i>rmargn</i>	numeric	local	The value of the current right margin setting.
<i>row</i>	numeric	local	The current cursor row (the actual row, not the logical line in the text).
<i>sb</i>	string	global	Up to 60 characters of the block buffer.
<i>se</i>	string	global	The name of the current edited file.
<i>sg</i>	string	global	The name of the last file specified in the get command.
<i>si</i>	string	global	The name of the main input file.
<i>s 0-9</i>	string	global	Read-write <i>s</i> variables, assigned only in the calc command.
<i>slx</i>	numeric	local	The length of the global <i>s</i> variable <i>slx</i> , where <i>x</i> is 0-9; or the second letter of a read-only string variable.
<i>sm</i>	string	global	The name of the last file specified for the macro get command.
<i>so</i>	string	global	The name of the OTHER input file.
<i>sp</i>	string	global	The name of the last file specified for the block put command.
<i>sr</i>	string	global	The replacement string of (?)replace .
<i>st</i>	string	global	The target string of (-)find and (?)replace .
<i>sw</i>	string	global	The name of the last file specified for the quit write command.

Table C-1. AEDIT Variables (continued)

Name	Type	Scope	Description
<i>tagA</i>	numeric	local	The offset of tag A.
<i>tagB</i>	numeric	local	The offset of tag B.
<i>tagC</i>	numeric	local	The offset of tag C.
<i>tagD</i>	numeric	local	The offset of tag D.
<i>time</i>	numeric	local	Time in decimal format HHMMSS.
<i>upch</i>	numeric	local	If the current character is a lowercase character (61H to 7AH), <i>upch</i> is the ASCII value of the uppercase character. Otherwise, <i>upch</i> is the same as <i>curch</i> .

□□□

Configuring AEDIT for Other Terminals

D

Tested Configurations

This appendix lists the configuration functions and values required to run AEDIT on several terminals. The `:config:termcap` file supplied with the iRMX OS contains configuration commands for these terminals. You can add AEDIT configuration commands for other terminals to that file.

Each terminal's configuration in the `:config:termcap` file has two parts, separated by a blank line. The first part is used by both AEDIT and the CLI (Command Line Interpreter). The second part is used only by AEDIT.

Most CRT terminals have switches to set certain screen or keyboard characteristics. These switches must be set as in Table D-1 for AEDIT to function correctly.

Table D-1. Switch Settings

Switch	Setting
Baud Rate	Must match system. Use the maximum baud rate possible
Full Duplex	On
RS232	On
Communication	Conversational
Self Echo	Off
Parity	Inhibit if available, otherwise space or 0 (zero)
Parity Sense	Even or odd (don't care)
Bits/Char	8
Stop Bits	1
Scrolling	On
EOM (End Of Message)	Off
Auto Line Feed or EOL Char	Off
or New Line	CR Only
or Return	CR Only
DTR	Off
25th Line	Match your AEDIT configuration
Chars/Lines	80-255 (AEDIT only uses 80 characters/line)
Lines/Page	5-66

Table D-1. Switch Settings (continued)

Unique to the Lear Seigler ADM3A	
Switch	Setting
Xon/Xoff	Off (Xon/Xoff protocol should be disabled, if available.)
Wraparound	On if AW=T, or off if AW=F (Wraparound must correspond to the AW configuration command.)
Space/Advance	Space (destructive space)
<CTRL-Z>Clear Screen	Enable
AEDIT Don't Care Settings	
Switch	Setting
Scroll Type	Jump or Smooth
Character Set	
Cursor Style	Underline, block, steady, blinking
Autorepeat	
Margin Bell	
Keyclick	
Screen Background	Normal, reverse
Uppercase Only or Upper-/Lowercase	

DEC VT52

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 7x9 dot matrix. The maximum transmission rate is 19.2K baud. Note that the Escape character has to be changed so that the default Escape code can be used; <Ctrl-K> is typed instead of <Esc>. This terminal does not have a <Home> key; <Ctrl-O> is typed instead of <Home>.

```
AB = 0B; AR = 7F;
AFCL = 1B44; AFCR = 1B43; AFCU = 1B41; AFCD = 1B42; AFCH = 08;
AFML = 1B44; AFMR = 1B43; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 1B4B; BELL = 07;

AV = 24; AW = F;
AFMU = 1B41; AFMD = 1B42; AFMH = 1B48;
AFES = ; AFER = 1B4A;
AFDL = ; AFIL = 1B49;
AFAC = 1B59; AO = 20; AX = F;
```



Note

<Ctrl-K> is used for <Esc>.
<Ctrl-O> is used for <Home>.

DEC VT100 and VT102

This terminal can be formatted with 132 characters per line or 80 characters per line. The characters are generated in a 7x9 dot matrix. The maximum transmission rate is 19.2K baud; do not use baud rates above 9600. You may choose between the DEC VT52 compatible and the ANSI standard (X3.41-1974, X3.64-1977) compatible terminal escape sequences for cursor control and screen erase functions. See the DEC VT52 description for the VT52 codes. Note that the Escape character has to be changed so that the default Escape code can be used; <PF4> is typed instead of Escape. This terminal does not have a <Home> key; <PF1> is typed instead of Home.

```
AB = 1B4F53; AR = 7F;
AFCL = 1B5B44; AFCR = 1B5B43; AFCU = 1B5B41; AFCD = 1B5B42; AFCH = 1B4F50;
AFML = 1B5B44; AFMR = 1B5B43; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = 1B5B324B; AFEL = 1B5B4B;
BELL = 07;

AV = 24;
AFMU = 1B5B41; AFMD = 1B5B42; AFMH = 1B5B48;
AFES = 1B5B324A; AFER = 1B5B4A;
AFDL = ; AFIL = ;
AFRV = 1B5B376D; AFNV = 1B5B306D;
AI = T; AC = T;
AH = VT100;
```



Note

<PF4> is set as Escape.

<PF1> is set as Home.

Wraparound must be turned off.

Automatic Xon/Xoff must be turned off.

The delete/insert line output function codes are not used because of poor performance, although they are available on the VT100 and VT102.

Hazeltine 1510E (with escape lead-in)

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 7X10 dot matrix. The maximum transmission rate is 19.2K baud.

You may choose between the Esc or the tilde (~) character as the control sequence lead-in. It is advisable to use the tilde; if you use the Esc character, you must change the Break character.

```
AB = 7E; AR = 7F;
AFCL = 08; AFCR = 10; AFCU = 1B0C; AFCD = 1B0B; AFCH = 1B12;
AFML = 08; AFMR = 10; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 1B0F; BELL = 07;

AV = 24;
AFMU = 1B0C; AFMD = 1B0B; AFMH = 1B12;
AFES = ; AFER = 1B18;
AFDL = 1B13; AFIL = 1B1A;
AFAC = 1B11; AO = 0; AX = T;
AFRV = 1B1F; AFNV = 1B19; AC = T; AI = T;
ADDL = 20; ADIL = 20;
```



Note

Tilde is used for <ESC>. <CTRL-V> is used for <HEX>.

Hazeltine 1510T (with tilde lead-in)

```
AB = 1B; AR = 7F;
AFCL = 08; AFCR = 10; AFCU = 7E0C; AFCD = 7E0B; AFCH = 7E12;
AFML = 08; AFMR = 10; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 7E0F; BELL = 07;

AV = 24;
AFMU = 7E0C; AFMD = 7E0B; AFMH = 7E12;
AFES = ; AFER = 7E18;
AFDL = 7E13; AFIL = 7E1A;
AFAC = 7E11; AO = 0; AX = T;
AFRV = 7E1F; AFNV = 7E19; AC = T; AI = T;
ADDL = 20; ADIL = 20;
```



Note

<Ctrl-V> is used for <Hex>.

Lear Siegler ADM3A

This terminal displays 24 lines of 80 characters per line. The characters are generated in a 5X7 dot matrix. The maximum transmission rate is 19.2K baud.

```
AB = 1B; AR = 7F;
AFCL = 08; AFCR = 0C; AFCU = 0B; AFCD = 0A; AFCH = 1E;
AFML = 08; AFMR = 0C; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = ; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 0A; AFMH = 1E;
AFES = 1A; AFER = ;
AFAC = 1B3D; AO = 20; AX = F;
ADES = 5; \ * 19200 baud * \
```



Note

<Ctrl-V> is used for <Hex>.

Switch

Space/Advance

<Ctrl-Z> Clear Screen

Setting

Space

Enable

PC

This is the standard PC console used for iRMX for Windows or iRMX for PCs. It displays 25 lines with 80 characters per line.

```
AB = 1B; AR = 7F;
AFCL = 1F; AFCD = 1C; AFCH = 1D;
AFCR = 19; AFCU = 1E; AFCD = 1C; AFCH = 1D;
AFMB = 0D; AFML = 1F; AFMR = 19;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = 02; AFEL = 03; BELL = 07;

AV = 25;
AFMU = 1E; AFMD = 1C; AFMH = 1D;
AFES = 0C; AFER = 01; AFEN = 05;
AFDL = 05; AFIL = 06; AFBK = 20;
AFAC = 04; AO = 20; AX = F;
AFRV = 0B5F; AFNV = 0B0A; AFST = 15010B0A; AFEN = 15021503;
AT = F; AC = T; AI = T;
```

Qume QVT102

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2K baud.

```
AB = 1B; AR = 7F;
AFCL = 08; AFCD = 0A; AFCH = 1E;
AFCR = 0C; AFCU = 0B; AFCD = 0A; AFCH = 1E;
AFML = 08; AFMR = 0C; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 1B54; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 0A; AFMH = 1E;
AFES = 1B2B; AFER = 1B59;
AFDL = 1B52; AFIL = 1B45;
AFAC = 1B3D; AO = 20; AX = F;
ADIL = 140; \ * 19200 BAUD * \
```



Note

<Ctrl-V> is used for <Hex>.

Televideo 910 PLUS

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2K baud.

```
AB = 1B; AR = 7F;
AFCL = 08; AFCR = 0C; AFCU = 0B; AFCD = 16; AFCH = 1E;
AFML = 08; AFMR = 0C; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 1B54; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 16; AFMH = 1E;
AFES = 1B2B; AFER = 1B59;
AFDL = 1B52; AFIL = 1B45;
AFAC = 1B3D; AO = 20; AX = F;
ADIL = 20; \ * 19200 BAUD * \
```



Note

<Ctrl-V> is used for <Hex>.

Televideo 925 and 950

This terminal displays 24 lines with 80 characters per line. The maximum transmission rate is 19.2K baud.

```
AB = 1B; AR = 7F;
AFCL = 08; AFCR = 0C; AFCU = 0B; AFCD = 16; AFCH = 1E;
AFMR = 0C; AFML = 08; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = ; AFEL = 1B54; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 0A; AFMH = 1E;
AFES = 1B2B; AFER = 1B59;
AFDL = 1B52; AFIL = 1B45;
AFAC = 1B3D; AO = 20; AX = F;
AFRV = 1B4734; AFNV = 1B4730; AC = F; AI = F;
ADIL = 60; \ * 19200 baud * \
```



Note

<Ctrl-V> is used for <Hex>.

Wyse 50

AB = 1B; AR = 7F;
AFCL = 08; AFCR = 0C; AFCU = 0B; AFCD = 0A; AFCH = 1E;
AFML = 08; AFMR = 0C; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = 0D1B54; AFEL = 1B74; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 0A; AFMH = 1E;
AFES = 1B2B; AFER = 1B59;
AFDL = 1B52; AFIL = 1B45;
AFAC = 1B3D; AO = 20; AX = F;
ADIL = 140;

X-Terminal

AFCL = 1B5B44; AFCR = 1B5B43; AFCU = 1B5B41; AFCD = 1B5B42;
AFML = 1B5B44; AFMR = 1B5B43; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = 1B5B324B; AFEL = 1B5B4B; BELL = 07;

AFMU = 1B5B41; AFMD = 1B5B42; AFMH = 1B5B48;
AFES = 1B5B324A; AFER = 1B5B4A;
AFDL = ; AFIL = ;
AFRV = 1B5B376D; AFNV = 1B5B306D;
AI = T; AC = T;
AH = VT100;
AB = 1B; AR = 08; AFCH = 1B5B48;
AV = 24;

Zentec Zepher and Cobra

This terminal displays 24 lines of 80 characters per line. The maximum transmission rate is 19.2K baud. To rub out a character on this terminal you must use the <Shift>-<Esc> key sequence.

```
AB = 1B; AR = 7F;
AFCL = 08; AFCD = 0A; AFCH = 1E;
AFML = 08; AFMR = 0C; AFMB = 0D;
AFXA = 01; AFXF = 06; AFXX = 18;
AFEK = 0D1B54; AFEL = 1B54; BELL = 07;

AV = 24;
AFMU = 0B; AFMD = 0A; AFMH = 1E;
AFES = 1B2B; AFER = 1B59;
AFDL = 1B52; AFIL = 1B45;
AFAC = 1B3D; AO = 20; AX = F;
AFRV = 1B4734; AFNV = 1B4730; AC = F; AI = F;
ADIL = 60; \ * 19200 baud * \
```



Note

The key (<Shift>-<Esc>) is used for <Rub-out>, <Ctrl-V> is used for <Hex>.



ASCII Codes

E

This appendix lists ASCII codes. Table E-1 is a list of codes and Table E-2 is a list of code functions.

Table E-1. ASCII Code List

Dec	Hex	Character	Dec	Hex	Character
0	00	NULL	26	1A	SUB
1	01	SOH	27	1B	ESC
2	02	STX	28	1C	FS
3	03	ETX	29	1D	GS
4	04	EOT	30	1E	RS
5	05	ENQ	31	1F	US
6	06	ACK	32	20	SP
7	07	BEL	33	21	!
8	08	BS	34	22	"
9	09	HT	35	23	#
10	0A	LF	36	24	\$
11	0B	VT	37	25	%
12	0C	FF	38	26	&
13	0D	CR	39	27	'
14	0E	SO	40	28	(
15	0F	SI	41	29)
16	10	DLE	42	2A	*
17	11	DC1	43	2B	+
18	12	DC2	44	2C	,
19	13	DC3	45	2D	-
20	14	DC4	46	2E	.
21	15	NAK	47	2F	/
22	16	SYN	48	30	0
23	17	ETB	49	31	1
24	18	CAN	50	32	2
25	19	EM	51	33	3

Table E-1. ASCII Code List (continued)

Dec	Hex	Character	Dec	Hex	Character
52	34	4	90	5A	Z
53	35	5	91	5B	[
54	36	6	92	5C	\
55	37	7	93	5D]
56	38	8	94	5E	^
57	39	9	95	5F	-
58	3A	:	96	60	`
59	3B	;	97	61	a
60	3C	<	98	62	b
61	3D	=	99	63	c
62	3E	>	100	64	d
63	3F	?	101	65	e
64	40	@	102	66	f
65	41	A	103	67	g
66	42	B	104	68	h
67	43	C	105	69	i
68	44	D	106	6A	j
69	45	E	107	6B	k
70	46	F	108	6C	l
71	47	G	109	6D	m
72	48	H	110	6E	n
73	49	I	111	6F	o
74	4A	J	112	70	p
75	4B	K	113	71	q
76	4C	L	114	72	r
77	4D	M	115	73	s
78	4E	N	116	74	t
79	4F	O	117	75	u
80	50	P	118	76	v
81	51	Q	119	77	w
82	52	R	120	78	x
83	53	S	121	79	y
84	54	T	122	7A	z
85	55	U	123	7B	{
86	56	V	124	7C	
87	57	W	125	7D	}
88	58	X	126	7E	~
89	59	Y	127	7F	DEL

Table E-2. ASCII Code Definition

Abbreviation	Meaning	Dec	Hex
NUL	NULL Character	0	0
SOH	Start of Heading	1	1
STX	Start of Text	2	2
ETX	End of Text	3	3
EOT	End of Transmission	4	4
ENQ	Enquiry	5	5
ACK	Acknowledge	6	6
BEL	Bell	7	7
BS	Backspace	8	8
HT	Horizontal Tabulation	9	9
LF	Linefeed	10	0A
VT	Vertical Tabulation	11	0B
FF	Form Feed	12	0C
CR	Carriage Return	13	0D
SO	Shift Out	14	0E
SI	Shift In	15	0F
DLE	Data Link Escape	16	10
DC1	Device Control 1	17	11
DC2	Device Control 2	18	12
DC3	Device Control 3	19	13
DC4	Device Control 4	20	14
NAK	Negative Acknowledge	21	15
SYN	Synchronous Idle	22	16
ETB	End of Transmission Block	23	17
CAN	Cancel	24	18
EM	End of Medium	25	19
SUB	Substitute	26	1A
ESC	Escape	27	1B
FS	File Separator	28	1C
GS	Group Separator	29	1D
RS	Record Separator	30	1E
US	Unit Separator	31	1F
SP	Space	32	20
DEL	Delete	127	7F

□□□

- ! lines and line terminators, 13
- !system command, 51
- / in Insert mode, 19
- / repeat function, 14
- ? printing and nonprinting characters, 13
- ?replace command, 24
- @ in block command, 4
- @ sign, 27

A

- again command, 33
- algebraic shift operator, 96
- ascii codes, 145

B

- backslash in macros, 70
- BATCH control (BA), 64
- beep warning, 13
- block buffer, 27
- block command, 27
 - block buffer, 27
 - buffer, 28
 - delete, 28
 - find, 28
 - find, 28
 - jump, 28
 - put, 28
- buffer, 14
 - block, 14
 - current, 14
 - main, 14
 - other, 14
 - secondary, 14

- busy/waiting indicator, 2

C

- calc command, 52
- calc command, 93
- commands
 - jump
 - tags, 13
 - other, 5, 32
 - configuration command, 109
 - configuration values, 115
 - copying text, 5
 - count (repeat function), 12
 - current buffer, 32
 - current file, 45
 - cursor, 2
 - cursor control keys <Left>, <Right>, <Up>, and <Down>, 8

D

- DEC VT100 terminal, 138
- DEC VT52 terminal, 137
- delay codes, 115
- delch command(<Ctrl-F>), 15
- delete command, 29
- delete commands, 15
- deleting
 - macros, 72
 - text, 3
- dell command (<Ctrl-X>), 15
- delli command (<Ctrl-Z>), 17
- delr command (<Ctrl-A>), 17
- display, 9

E

- editing commands, 121
- ending an editing session, 6
- end-of-file (EOF), 2
- error messages, 125
- errors
 - AEDIT, 115
- Esc key, 8
- examples
 - macro, 82
- Exchanging text, 20
- execute command, 53
- execute commands, 74

F

- filenames, case, 66
- find command, 21
- Forwardonly control (FO), 59
- function key, 8
- function keys, 15, 119

G

- get command, 30
- global variables, 86

H

- Hazeletine 1510E terminal, 139
- Hazeletine 1510T terminal, 140
- hex command, 43
 - input, 43
 - output, 44
- Home key, 8

I

- insert command, 19
- insert mode (I), 19
- inserting text, 3, 19
- invocation commands, 55
- invocation controls, 57
 - forwardonly (FO), 59
 - macro (MR), 62
 - macrosize (MS), 63

- recover (RC), 61
- viewonly (VO), 60

J

- jump command, 26
 - end, 26
 - line, 26
 - position, 26
 - start, 26
 - tags, 26

K

- keyboard, 8
- kill_wnd command, 50

L

- Lear Siegler ADM-3A terminal, 140
- line terminator, 13
- line-edited prompt, 12
- local variables, 89

M

- macro
 - conditional statements, 105
 - control, 62
 - control codes, 77
 - definitions, 68
 - deleting, 72
 - examples, 82
 - failure, 78
 - files, 76
 - messages, 80
 - prompt, 81
 - screen display, 80
 - single-character, 75
 - useful.mac file, 101
 - window, 81
 - writing, 104
- macro command, 54
 - create, 68
 - get, 69
 - insert, 70
 - list, 71

- save, 71
- macro command, 68
- macro modes, 73
 - modeless macro, 73
 - non-modeless macro, 73
- macrosize control (MS), 63
- main buffer, 14
- main command level, 4
- mde
 - insert, 19
- menu prompt, 8
- message line, 9
- Mode
 - Exchange, 20

N

- N variables, 86
- numeric constant, 95

O

- operators, 96
 - algebraic shift, 96
 - binary, 96
 - shift/rotate, 96
 - unary, 96
- Other buffer, 32
- other command, 32

P

- paragraph command, 48
 - fill, 48
 - justify, 48
- PC terminal, 141
- printing and nonprinting characters, 13
- prompt line, 10

Q

- quit command, 45
 - abort, 45
 - exit, 45
 - init, 46
 - update, 46
 - write, 47

- Qume QVT102 terminal, 141

R

- radix (set subcommand), 86
- read-only variables, 87
- read-write variables, 87
- recover control (RC), 61
- reedit mode, 12
- repeat function (count), 14
- replace command, 23
- rubout command, 15

S

- S variables, 87
- secondary
 - buffer, 32
 - file, 45
- set command, 34
 - autonl, 34
 - bak_file, 35
 - case, 35
 - display, 35
 - e_delimit, 36
 - go, 37
 - highbit, 37
 - indent, 38
 - k_token, 38
 - leftcol, 39
 - margin, 39
 - notab, 40
 - radix, 40
 - showfind, 41
 - tabs, 41
 - viewrow, 42
- Set commands in macro files, 76
- setdisplay command, 80
- shift/rotate operator, 96
- single-character macros, 75
- starting an Editing session, 2
- string constant, 95
- string-variables, 87

T

- Tab key, 8

- tag command, 25
- Televideo 910 Plus terminal, 142
- Televideo 925 and 950 terminal, 142
- termcap file, 135
- terminal
 - Televideo 910 Plus, 142
 - Televideo 925 and 950, 142
 - Wyse 50, 143
 - X-terminal, 143
 - Zentec Zepher and Cobra, 144
- terminal configuration, 135
- terminals
 - DEC VT100, 138
 - DEC VT52, 137
 - Hazeletine 1510E, 139, 140
 - Lear Siegler ADM-3A, 140
 - PC, 141
 - Qume QVT102, 141
- text area, 14
- token, 38
- tutorial, 2

U

- undo command, 18
- uppercase filenames, 66
- useful.mac macro, 101

V

- variables
 - AEDIT, 72, 85
 - global, 86
 - numeric, 86
 - N-variables, 86
 - read-only, 87
 - read-write, 87
 - string, 87
 - S-variables, 88
- variables in Aedit, 131
- view command, 31, 80
- viewonly control (VO), 60

W

- window command, 49
- Work files, 66
- Wyse 50 terminal, 143

X

- xchange command, 20
- xchange mode, 3
- X-terminal, 143

Z

- Zentec Zepher and Cobra terminal, 144