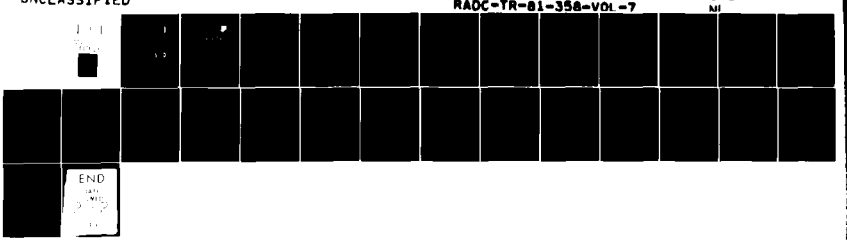
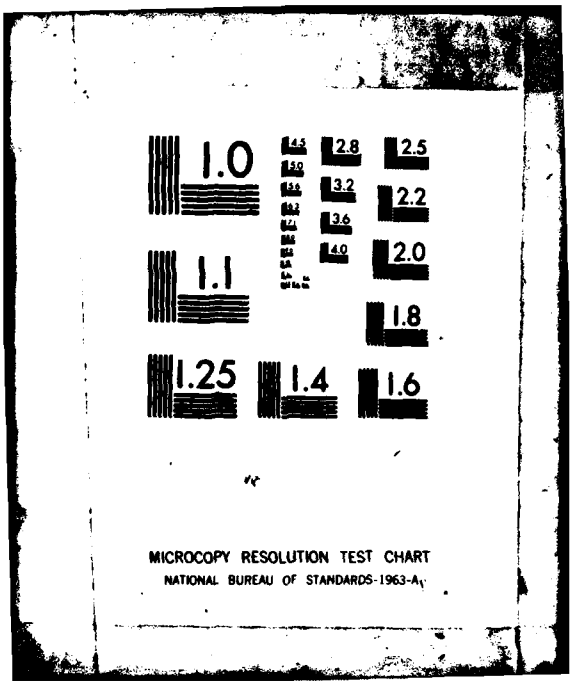
AD-A110 005 INTERMETRICS INC CAMBRIDGE MA F/G 9/2
ADA INTEGRATED ENVIRONMENT I COMPUTER PROGRAM DEVELOPMENT SPECI--ETC(U)
DEC 81 F30602-80-C-0291
UNCLASSIFIED RADC-TR-81-358-VOL-7 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PHOTOGRAPH THIS SHEET

AD A110005

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

Intermetrics, Inc
Cambridge MA

ADA Integrated Environment I Computer Program Development Specification. Interim Rept. 15 Sep. 80 - 15 Mar. 81 Dec. 81

DOCUMENT IDENTIFICATION

Contract F30602-80-C-0291 RADC-TR-81-358, Vol. VIII

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS	GRA&I
DTIC	TAB
UNANNOUNCED	
JUSTIFICATION	
BY	
DISTRIBUTION /	
AVAILABILITY CODES	
DIST	AVAIL AND/OR SPECIAL
A	

DTIC COPY INSPECTED

DISTRIBUTION STAMP

DTIC
ELECTE
S JAN 25 1982 D
D

DATE ACCESSIONED

82 01 12 012

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDA-2

DOCUMENT PROCESSING SHEET

DTIC FORM OCT 79 70A

RADC-TR-81-358, Vol VII (of seven)
Interim Report
December 1981

ADA INTEGRATED ENVIRONMENT I COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

Intermetrics, Inc.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This document was produced under Contract F30602-80-C-0291 for the Rome Air Development Center. Mr. Don Roberts is the COTR for the Air Force. Dr. Fred H. Martin is Project Manager for Intermetrics.

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-358, Volume VII (of seven) has been reviewed and is approved for publication.

APPROVED: Donald F. Roberts

DONALD F. ROBERTS
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-358, Vol VII (of seven)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ADA INTEGRATED ENVIRONMENT I COMPUTER PROGRAM DEVELOPMENT SPECIFICATION	5. TYPE OF REPORT & PERIOD COVERED: Interim Report 15 Sep 80 - 15 Mar 81
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0291
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Intermetrics, Inc. 733 Concord Avenue Cambridge MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62204F/33126F 55811908
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (COES) Griffiss AFB NY 13441	12. REPORT DATE: December 1981
	13. NUMBER OF PAGES: 20
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Donald F. Roberts (COES)

Subcontractor is Massachusetts Computer Assoc.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada	MAPSE	AIE
Compiler	Kernel	Integrated environment
Database	Debugger	Editor
KAPSE	APSE	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Ada Integrated Environment (AIE) consists of a set of software tools intended to support design, development and maintenance of embedded computer software. A significant portion of an AIE includes software systems and tools residing and executing on a host computer (or set of computers). This set is known as an Ada Programming Support Environment (APSE). This B-5 Specification describes, in detail, the design for a minimal APSE, called a MAPSE. The MAPSE is the foundation upon which an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

APSE is built and will provide comprehensive support throughout the design, development and maintenance of Ada software. The MAPSE tools described in this specification include an Ada compiler, linker/loader, debugger, editor, and configuration management tools. The kernel (KAPSE) will provide the interfaces (user, host, tool), database support, and facilities for executing Ada programs (runtime support system).

TABLE OF CONTENTS

INTERMETRICS INCORPORATED • 733 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

1.0 SCOPE

1.1 Identification

This specification establishes the requirements for the MAPSE Text Editor. The Text Editor provides basic editing facilities suitable for editing general text such as program source or program documentation.

1.2 Functional Summary

The design of the MAPSE editor is largely based on the highly popular ex or vi editor developed at the University of California, Berkeley, under various corporate and government grants. The system, which runs under UNIX, has been documented by William Joy of the Department of Electrical Engineering and Computer Science at UCB, and is in general distribution. This model was chosen due to its wide range of applicability and its wide acceptance in the user community. It is felt that an editor, being one of the most highly user-interactive parts of the MAPSE, should be based on a design which has been shown to be successful.

The MAPSE Text Editor is the basic tool for the creation and modification of textual material. The edited material resides in the MAPSE database and is created or modified interactively through the editor. The editor operates on a copy of the object in its "edit buffer". The original object itself is not modified until the user so requests. The editor may be operated in various modes as appropriate to the sophistication of the user and the characteristics of the terminal device:

- Command Mode: commands are entered when a '>' prompt is present and are executed each time a complete line is entered (terminated by a carriage return or "new line"). Sufficient commands are available to allow flexible editing of a file from even hardcopy, low-speed terminals. Command functions include: find, alter, insert, delete, input, output, move, copy and substitute.

- Text Input Mode: the editor gathers input, line by line, and places it in the edited file. No prompting is done and the user must leave Text Input Mode explicitly. This mode is the basic method for adding lines to a file.

INTERMETRICS INCORPORATED • 733 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

- Full Screen Mode: a superset of Command mode. On appropriate CRT terminals, the CRT screen is made a "window" into the file being edited; a contiguous block of the file's text is displayed. The terminal's cursor is used to "move around" in the text and cursor commands may be given (with effects such as "delete the character at the cursor location" or "move cursor forward one word"). All of the "line commands" of the basic Command mode are available: the user may issue these commands on the "command line" of the display and see the effects reflected in the block of text shown in the terminals full display.

The editor begins operation in the simpler Command mode. The user may invoke the more sophisticated features as needed.

In addition to standard editing (text manipulation) features, the editor provides a few simple additional capabilities, which enhance its utility in the KAPSE environment. These include:

- File/object handling. The ability to write all or part of the object under edit into another named object, plus the ability to read a named object into the editor's edit buffer (e.g., to include one file in another).

- KAPSE or user program invocation. The ability to "escape" to the KAPSE or user program from within the editor and, optionally, pass portions of the edited object to the program as input. Correspondingly, the output of such an invoked program may be read into the edit buffer. This allows, for instance (presuming the existence of a sorting program), the invocation of a sort program with the buffer contents as input and the replacement of the buffer contents with the sort program's output.

- User controlled options. A set of options that parameterize some of the editor's functions is provided. Generally, these are used to set default counts or modes for various commands. Examples are: (1) setting tabstops to be every "n" positions; or (2) setting a "wrap right margin" mode in which the editor automatically senses the approach of the right margin during text input and breaks the input line at a blank near the margin. This is useful for documentation input when the typist is more effective if margins are "automatic". (This mode is similar to those provided on many stand-alone word processors.)

- Edit Scripts. The ability to use a "canned" set of edit commands out of a file. Commonly used edit sequences can be saved and applied during later edit sessions.

2.0 APPLICABLE DOCUMENTS

Please note that the bracketed number preceding the document identification is used for reference purposes within the text of this document.

2.1 Government Documents

[G-1] Reference Manual for the Ada Programming Language, proposed standard document, July 1980.

2.2 Non-Government Documents

[I-1] System Specification for Ada Integrated Environment, Type A, Intermetrics, Inc., March 1981, IR-676.

Computer Program Development Specifications for Ada Integrated Environment (Type 5):

[I-2] Ada Compiler Phases, IR-677.

[I-3] KAPSE/Database, IR-678.

[I-4] MAPSE Command Processor, IR-679.

[I-5] MAPSE Generation and Support, IR-680.

[I-6] Program Integration Facilities, IR-681.

[I-7] MAPSE Debugging Facilities, IR-682.

[I-8] Technical Report (INTERIM), IR-684.

INTERMETRICS INCORPORATED • 733 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

3.0 REQUIREMENTS

3.1 Program Definition

The MAPSE Text Editor provides a framework within which different kinds of structured and unstructured text may be edited. It must interface with the file system and the general KAPSE environment on the one hand and the user (through the user's terminal) on the other.

A user invokes the Editor, specifying a database object that is to be manipulated. The editor creates a copy of the object in its edit buffer and interprets user commands to effect changes to the buffer's contents. The buffer is copied into the database (into its original object or another) at user request. The editor can be used to create an object if it does not exist. The detailed functional requirements of the MAPSE text editor are dictated primarily by the command language which it supports. The description of that command language is found in Section 3.2.

3.1.1 Source Management and the Text Editor

The MAPSE Text Editor is the primary means for creating and editing source text, both programs and documentation. As such, the Editor provides an important user interface to the KAPSE facilities used for source management.

Each text object in the KAPSE database is a component of a composite object, and has a set of distinguishing attributes which differentiate it from all other components of this composite object. To edit a particular text object, the user specifies an access path going through some window with a capacity that provides appropriate access rights to all or part of the composite object enclosing it [I-3]. The access path identifies the window and then specifies the distinguishing attributes of the component, in the general form:

```
window_name.dist_att_1.dist_att_2,. ... .dist_att_N
```

For example:

```
WORKSPACE.SHUTTLE.INITIALIZATION.PKG_SPEC
```

The editor can be used to create a totally new text object, or to edit an existing text object. In both cases, the editor automatically takes advantage of KAPSE facilities for source archiving, synchronization, and access control.

When creating a new text object, the editor uses the KAPSE history facilities to establish it as the beginning of a new source archive. Future edits to the object will all be remembered in this source archive, allowing for the possibility of recreating past revisions and preventing catastrophic loss.

When editing an existing object, the user may specify either that the revised contents should replace the original contents of the object, or that the revised contents should be stored under a new name (with some new value for a distinguishing attribute). In either case, the editor specifies (with the history attribute of the revision) that it is derived from the original object and is a member of the same source archive.

Synchronization is effected by opening the output object, whether it has the same or a new name, with reservation for EXCLUSIVE_WRITE. By so doing, the editor ensures that no other user can simultaneously create or edit this object.

The KAPSE access control verifies that the user's window(s) provide the necessary access rights -- SELECT_COMPONENT and READ of the input object, and CREATE_COMPONENT (if new) and WRITE of the output object.

As a final aid to effective source management, the editor allows the user to record for posterity the purpose of the editing session. The editor saves the user's declared purpose as the value of a non-distinguishing attribute labeled PURPOSE_FOR_REVISION. The editor, of course, cannot force the user to give meaningful comments, but by making the process painless and automatic, it gives the conscientious user a medium to record information which otherwise might be lost or forgotten.

As a result of this standard editing procedure, the user and his/her manager are provided with a useful record of the development of a piece of text, as well as being protected against improper or mistaken modification to important source documents.

3.1.2 Interfaces

The editor determines device characteristics from the KAPSE via GET_INPUT_INFO and GET_OUTPUT_INFOR. Within the limits of the device, the KAPSE translates the standard set of device control character sequences into the specific codes appropriate to the particular terminal. The editor uses SET_COL and SET_LINE of the extended TEXT_IO package [I-3, 3.2.8.4] to control cursor or print-head movement.

The editor is able to deal with a variety of devices (line-at-a-time hardcopy through full function "smart" CRTs) by knowing specific terminal characteristics. These characteristics include control (input and output) sequences for common functions such as:

- clear screen
- erase to end-of-line
- hardware character and line insert/delete

The editor uses standard KAPSE functions for manipulation of database objects. Also, the KAPSE is used for invoking user or system programs under the editor and passing/receiving data to/from those programs. The editor operates as a standard Ada program in the KAPSE environment.

3.2 Detailed Functional Requirements

Due to the highly interactive nature of a text editor, the traditional breakdown of requirements into Inputs, Processing, and Outputs has been combined below under the general topic of the command language and its effects on the edited object and the editor's environment.

3.2.1 Invocation

The editor is invoked as a standard Ada program, specifying the object to be edited as an argument as in:

```
: EDIT X
```

The argument (X) is prepared for editing and becomes the current file name.

An optional second argument may be supplied, which specifies the location from which the editor will initially read commands. This second argument is used to operate the editor in a "batch" mode as in:

```
: EDIT X, SCRIPT
```

If the optional second argument is omitted, the default command source is the user's terminal and a conventional interactive edit session proceeds.

3.2.2 The Edit Buffer

In order to perform its tasks, the editor sets aside a temporary work space, called a "buffer", separate from the user's permanent database object. Before starting to work on an existing object, the editor makes a copy of it in the buffer, leaving the original untouched. All editing changes are made to the buffer copy, which must be written back into the database in order to update the old version or create a new object. The buffer disappears at the end of the editing session. If the user attempts to end the editing session prior to saving a changed buffer, the editor requests verification from the user.

3.2.3 Basic Editing Modes - Command and Text Input

Command and Text Input modes are available on all terminals. They are line-oriented and are entered a line at a time in response to a prompt.

The text in the edit buffer is considered to be unique lines, each line identifiable by its location in the buffer; i.e., its count, starting at 1. The count always reflects the current buffer contents. This means that addition or deletion of lines may affect the number (count) of subsequent lines. The line number does not physically exist in the edited text; it is merely an index used by the editor. (A submode of the editor can be used to cause the line numbers to be displayed whenever a line is displayed.)

3.2.3.1 Command Structure

Most commands are English words. The first character (or two) are usually acceptable abbreviations. (The ambiguity of abbreviations is resolved in favor of the more commonly used commands.) For example, to delete the current line the user would type: delete. To delete the eighth line of the buffer, 8delete (or 8d) is used.

Most commands accept prefix addresses, specifying the lines in the file upon which they are to have an effect. The forms of those addresses are discussed below. A number of commands also may take a trailing count, specifying the number of lines to be involved. If a command takes any other information or parameters, this information is always given after the command name. Examples: 10d deletes line 10 while 10d10 deletes 10 lines starting at line 10.

3.2.3.2 Command Addressing Primitives

The following notation is used to address specific lines in the edit buffer. These address primitives are combined with command names to direct execution of the command.

Primitive	Function/Meaning
.	The current line. Most commands leave the current line as the last line which they affect. The default address for most commands is the current line.
n	The n^{th} line in the editor's buffer.
$	The last line in the buffer.

Primitive	Function/Meaning
+n -n	An offset relative to the current buffer line.
/pattern/ ?pattern?	Scan forward or backward, respectively, for a line containing "pattern", a sequence of characters (including "wild cards") which are to be matched.
'' 'x	"Marked" lines. Any line may be marked (the mark command) with a letter from a to z. A marked line is addressed by 'x when x is the marking letter. The form '' is a special case which is always maintained by the editor to address the previous current line.

3.2.3.3 Combining Addressing Primitives

Addresses to commands consist of a series of two or more addressing primitives separated by ',' or ';'. Such addresses are evaluated left to right. When addresses are separated by ';', the current line ('.') is set to the value of the addressing expression which precedes the ';', before the next address is interpreted. If a command takes two addresses, the first addressed line must precede the second in the buffer.

Examples of line addresses:

.	The current line
.+1	The line after the current line
5	Line 5
5,10	Lines 5 through 10
.-1,+6	Lines 4 through 11 if . is 5
$-10;.+3	Tenth to last line through seventh to last line (same as $-10,$-7)

3.2.3.4 Basic Command Descriptions

The following form is a prototype for all line commands:

<address> command <parameters> <count> <flags>

All parts are optional; the degenerate case (a carriage return or new line with nothing typed before it) is the empty command, which is interpreted as a special case and causes the next (.+1) line in the buffer to be displayed.

Flags may appear optionally after commands. The flags allowed are '#', 'p', and 'l'. If an optional flag is used, the command abbreviated by these characters (number, print, or list) is executed after the command completes.

In the following command descriptions, the default addresses are shown in parentheses, which are not, however, part of the command.

Append

(.)append abbr: a
<text>
.

Places editor into Text Input mode. Lines are read from the terminal and placed after the addressed line. A single '.' on an input line terminates Text Input mode and returns the editor to Command mode.

Example:

```
$append    adds line to end of buffer
0append    adds lines to beginning of
           buffer.
```

Change

(.,.)change<count> abbr: c
<text>
.

Replaces the specified lines with the input <text>. Input is terminated by a single '.'. If no <text> is input, the effect is as a delete.

Copy

(.,.)copy<addr> abbr: co

A copy of the specified lines is placed after <addr>.

Example:

```
5,10copy0  puts a copy of line 5 through 10 at
           the beginning of the buffer.
```

Delete

(.,.)delete<buffer><count>
abbr: d

Removes the specified lines from the edit buffer. Optionally, a "named buffer" may be specified, in which case the deleted lines are saved into an internal editor buffer (there are a-z of them). Internal named buffers may be further manipulated by the put command and the yank command.

File

file<newname> abbr: f

Prints the current file name and number of lines in the buffer. If <newname> is specified, the current file name is replaced with <newname>.

Full Screen

(.)fullscreen<count>
abbr: fu

Enters full screen mode on an appropriate terminal. The addressed line appears at the top of the full screen window into the edit buffer. The <count>, if present specifies an initial window size in lines.

Global

(1,$)global /pattern/<cmds>
abbr: g

First marks each line, among those specified, which matches the given pattern. Then the given command list is executed with '.' initially set to each marked line. The global command itself may not appear in <cmds>.

Insert

(.)insert abbr: i
<text>
.

Places the given text before the specified line. This command differs from append only in the placement of the text.

Join

(.,.+1)join<count> abbr: j

Places the text from a specified range of lines together on one line.

List

(.,.)list<count> abbr: l

Prints the specified lines in an unambiguous way: a few non-printing characters (e.g., tab and backspace) are represented by mnemonic overstrikes. All other non-printing characters are printed in octal.

Mark

(.)mark<x>

Marks the specified line as x (a single lower case letter). The addressing form 'x then addresses this line.

Move

(.,.)move<addr> abbr: m

The move command repositions the specified lines to be after <addr>.

Number

(.,.)number<count> abbr: nu or #

Prints each specified line preceded by its buffer line number.

Print

(.,.)<count> abbr: p

Prints the specified lines.

Put

(.)put<buffer> abbr: pu

Puts back previously deleted or yanked lines. Normally used with delete to effect movement of lines or with yank to effect duplication of lines. If no <buffer> is specified, then the last deleted or yanked text is restored. By using a named buffer, text may be restored that was saved in that buffer at any previous time in the edit session.

Quit

quit abbr: q

Causes the editor to terminate. No automatic write of the edit buffer to a database object is performed. However, a warning message is issued if the file has changed since the last write command and the editor does not quit. Another quit command immediately after the warning message will cause the editor to discard its buffer and terminate.

Read

(.)read<file> abbr: r

Places a copy of the text of the given file in the editing buffer after the specified line. If no file name is present, the current file name is used.

(.)read!<command> abbr: r

Invokes the command processor to process the specified command. The standard output of the command is read into the buffer after the specified line.

Set

set<parameter>

With no arguments, prints those options whose values have been changed from their defaults; with <parameter> 'all' it prints all of the option values. Legal values for <parameter> are given in Section 3.2.2.6.

Source

source<file> abbr: so

Reads and executes commands from the specified file. Source commands may be nested.

Substitute

(.,.)substitute /<pattern>/<replacement>/<options><count>
abbr: s

On each specified line, the first instance of <pattern> is replaced by replacement pattern <replacement>. If the global indicator option character 'g' appears, then all instances are substituted. If the confirm indication character 'c' appears, then before each substitution, the line to be substituted is typed with the string to be substituted marked with '|' characters. By typing a 'y' one can cause the substitution to be performed as shown. Any other input causes no change to take place.

Undo

undo abbr: u

Reverses the changes made in the buffer by the last buffer editing command.

Write

(1,$)write<file> abbr: w

Write the edit buffer back to <file>. If <file> is omitted, the text goes to the current file (usually the originally edited file).

(1,$)write!<command>
abbr: w

Writes the specified lines as standard input to <command>.

Yank

(.,.)yank<buffer><count>
abbr: ya

Places the specified lines in the named buffer for later retrieval via put.

Escape to Command Processor

!<command>

The remainder of the line after the '!' character is sent to the command processor for execution.

(addr,addr)!<command>

Takes the specified address range and supplies it as standard input to <command>; the resulting standard output then replaces the input lines.

Display Line Number

(.)=

Prints the line number of the addressed line.

Display Line

(.+1)

An address alone causes the addressed line to be printed. A blank line (no address) prints the next line in the file.

3.2.3.5 Substitute and Replacement Patterns

(a) Pattern Expressions. A pattern expression specifies a set of strings of characters. A member of this set of strings is set to be matched by the pattern expression.

(b) Basic Pattern Expression Summary. The following basic constructs are used to construct pattern expressions:

char — An ordinary character matches itself. The characters '^' at the beginning of a line, '*' as any character other than the first, '.', '\', '[', and '~' are not ordinary characters and must be escaped (preceded) by '/' to be treated as such.

^ — At the beginning of a pattern forces the match to succeed only at the beginning of a line.

$ — At the end of a pattern expression forces the match to succeed only at the end of a line.

. Matches any single character except the new line character.

[<string>] Matches any (single) character in the class defined by <string>. Most characters in <string> define themselves. A pair of characters separated by '-' in <string> defines the set of characters collating between the specified lower and upper bounds. (E.g., '[a-z]' as a pattern expression matches any (single) lower case letter if the collating sequence is ASCII.) If the first character of <string> is '^', then the construct matches those characters not in the defined set. To place any of the characters '^', '[', or '-' in <string>, you must escape them with a preceding '\'.

(c) Combining Pattern Expression Primitives. The concatenation of two pattern expressions matches the leftmost and then longest string which can be divided with the first piece matching the first pattern expression and the second piece matching the second. Any of the (single character matching) pattern expressions mentioned above may be followed by the character '*' to form a pattern expression which matches any number of adjacent occurrences (including 0) of characters matched by the pattern expression it follows.

The character '~' may be in a pattern expression, and matches the text which defined the replacement part of the last substitute command.

(d) Substitute Replacement Patterns. The basic metacharacters for the replacement pattern are '&' and '~': Each instance of '&' is replaced by the characters which the pattern expression matched. The metacharacter '~' stands, in the replacement pattern, for the defining text of the previous replacement pattern.

3.2.3.6 Option Descriptions

The options described below are modified by the set command.

(a) Ada. Default is noAda. When on, changes the definition of a "word" in full screen mode (e.g., for use in 'dw', 'cw', etc.) to be that of an Ada lexical unit as defined in Chapter 2 of the Ada LRM. In addition, the definition of a "paragraph" in full screen mode is redefined to be delimited by matching pairs of reserved words (e.g., begin-end, loop-end loop, if-end if, etc.).

(b) Autoindent (ai). Default is noai. Autoindent can be used to ease the preparation of structured program text. At the beginning of each append, change, or insert command or when a new line is opened or created by an append, change, insert, or substitute operation within full screen mode, the editor looks at the line being appended (after the first line changed or the line inserted before) and calculates the amount of white space at the start of the line. It then aligns the cursor at the level of indentation so determined.

If the user then types lines of text in, they will continue to be justified at the displayed indenting level. If more white space is typed at the beginning of a line, the following line will start aligned with the first non-white character of the previous line. To back the cursor up to the preceding tab stop one can hit ^D.

Specially processed in this mode is a line with no characters added to it, which turns into a completely blank line (the white space provided for the autoindent is discarded).

(c) List. Default is nolist. With list, all printed lines will be displayed unambiguously as in the list command.

(d) Number(nu). Default is nonumber. Causes all output lines to be printed with their line numbers. In addition, each input line will be prompted for by supplying the line number it will have.

(e) Prompt. Default is prompt. Command mode input is prompted with a '>'.

(f) Tabstop (ts). Default is ts=8. The editor expands tabs in the input file to be on tabstop boundaries for the purposes of display.

(g) Wrapscap (ws). Default is ws. Wrapscap searches using the pattern expressions in addressing will wrap around past the end of the file.

(h) Wrapmargin (wm). Default is wm=0. Wrapmargin defines a margin for automatic wrapover of text during input in full screen mode.

3.2.4 Full Screen Mode

On an appropriate terminal, the user may elect to operate the editor in full screen mode via the full screen command. In this mode the CRT screen becomes a window into the editors buffer. All of the non-fullscreen commands are still available. However, when the cursor is moved around the CRT screen an entire new set of "keystroke" commands become available. These commands cause movement or changes in the visual image displayed on the screen. It is always possible to enter non-fullscreen commands by using the '>' keystroke command, which places the cursor at the bottom line of the display and then accepts a non-fullscreen command.

Most of the full screen keystroke commands move the cursor around in the edit buffer. There are commands to move the cursor forward and backward in units of words, sentences and paragraphs. A small set of operators, like 'd' for delete and 'c' for change, are combined with the motion commands to perform operations such as delete word or delete paragraph, in a simple and natural way. This regularity and the mnemonic assignment of commands to keys make the editor command set easier to remember and use.

The various types of movement and modification commands are grouped together and described below. Note the use of the notation '^n' for the control-n character.

3.2.4.1 Movement within the Buffer

(a) Scrolling and Paging

^D	Scroll Down
^U	Scroll Up
^F	Forward a page
^B	Backward a page

(b) Searching, goto

/<text>	position cursor at the first occurrence of this string on a line after the comment line.
?<text>	same as /<text> except searches back in buffer.
n	go to next line with an occurrance of the previously mentioned <text>.
G	Go to. If preceded by a number (e.g., 1G) goes to the first line of the buffer. A 'G' with no prefixed number goes to the end of the buffer.

(c) Moving Around the Screen

+	advance to next line (return key has some function), first non-white character.
-	move to preceeding line, first non-white character.
H	move to top line (Home) of current screen, first non-white character.
M	move to Middle of current screen, first non-white character.
L	move to Last line of screen, first non-white character.
^N	move to next line maintaining column position.
^P	move to previous line maintaining column position.

(d) Moving Within a Line

w	advance cursor to next word.
b	backup cursor a word at a time.
e	advance to end of current word.
<space>	move right one character.
<backspace>	move left one character.

(e) Inserting

i	enters insert mode. Everything typed until an ESC (Escape key) is typed is inserted to the left of the cursor.
a	enters insert mode to the right of the cursor (appends) until ESC is typed.
o	open new lines after the current line.
O	open new lines before the current line.

3.2.4.2 Corrections

x delete (x-out) the character under the cursor.

r replace the character at the cursor position with the next character typed.

s substitute characters (terminated by ESC) for the character at the cursor position.

Note: the s and x commands may be preceded by a numeric count to indicate the number of characters to be involved.

d delete operator. It is followed by a "type" indication as in 'dw' to delete a word or db to delete the character preceeding the cursor (delete backwards).

c change operator - similar to d but accepts text up to ESC to replace indicated context. (E.g., CW changes the next word).

. repeats the last changing command.

3.2.4.3 Line Operations

dd deletes the current line.

cc change a whole line.

Note that dd and cc may be preceeded by counts.

3.2.4.4 Undo

u reverses the last change mode.

U restores current line to its state before last changes were made to it.

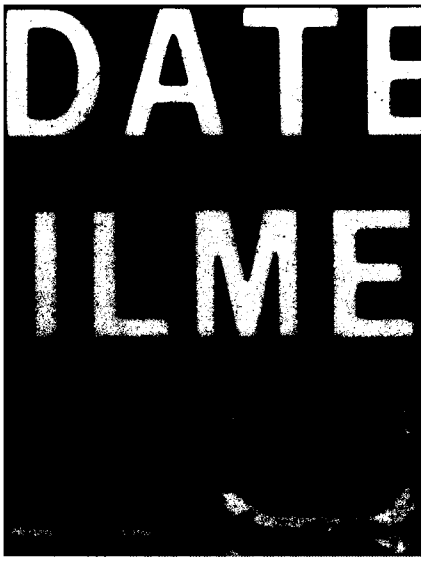
3.2.4.5 Higher Level Text Objects

)	move forward one sentence.
(	backward sentence.
}	forward paragraph.
{	backward paragraph.
xy	yank copy of text object into buffer 'x' (may be a-z).
xp	put text from buffer x after cursor.

Note: A paragraph begins after each empty line unless the 'Ada' option is set. See Section 3.2.3.6.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


DATE
ILME