

Full Screen Editor for NOS/VE

GD
CONTROL
DATA



Tutorial/Usage

60464015

Common Parameters

Parameter Name	Values
BOUNDARY (B)	LINE (LINES or L) STREAM (S)
COLUMN (COLUMNS or C)	integer CURRENT (C) FIRST_MARK (FM) LAST_MARK (LM) MARK (M) MAXIMUM (MAX)
LINE (LINES or L)	integer line identifier (modset.sequence) ALL (A) CURRENT (C) FIRST (F) FIRST_MARK (FM) FIRST_SCREEN (FS) LAST (L) LAST_MARK (LM) LAST_SCREEN (LS) MARK (M) SCREEN (S)

Full Screen Editor for NOS/VE

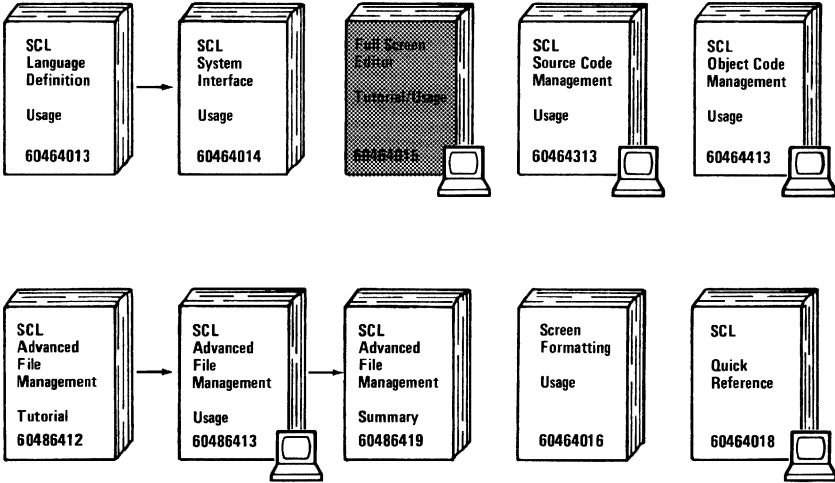
Tutorial/Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60464015

Related Manuals

SCL Manual Set:



indicates that the manual is available online.



indicates a reading sequence.

©1984, 1985 by Control Data Corporation.
All rights reserved.
Printed in the United States of America.

Manual History

This manual is Revision C, printed December 1985. It reflects NOS/VE Version 1.1.3 at PSR level 644. New features documented in this revision are:

- Addition of the SET_EPILOG command.
- Removal of the EPILOG parameter from the EDIT_FILE and EDIT_LIBRARY commands.
- Addition of POSITION_BACKWARD, POSITION_FORWARD, LIST_BACKWARD, LIST_FORWARD, LIST_LINES, LOCATE_ALL, LOCATE_NEXT, LOCATE_STRING, MARK_BOX, SET_PARAGRAPH_MARGINS, FORMAT_PARAGRAPHS, CENTER_LINES, and EXCHANGE_SCREEN_WIDTH subcommands.
- Addition of the \$PARAGRAPH_MARGINS function.
- Addition of a section on calling the editor from within a procedure.
- Addition of three new terminal types, the CDC722-30, the Zenith Z29, and the IBM (or compatible) PC with Connect.

Change bars reflect the latest revision level.

Previous

Revision	System Version	Date
A	1.1.1	July 1984
B	1.1.2	March 1985



Contents

About This Manual	7
Audience	7
Organization	7
Conventions	10
Additional Related Manuals	12
Ordering Manuals	12
Submitting Comments	12

Part I. Tutorial

Capabilities	1-1
Viking 721 Terminal	2-1
The Basics	3-1
Getting Started	3-1
Creating a File	3-6
Moving around the Screen ..	3-8
Changing the Screen Content	3-9
Moving to the End of a Line	3-11
Moving around within the File	3-12
Entering Subcommands ...	3-19
Getting Help	3-19
Deleting	3-23
Inserting	3-27
Searching	3-32
Marking	3-39
Copying	3-41
Moving	3-44
Undoing	3-46
Stopping an Editing Session	3-47

Part II. Usage

Before You Continue	4-1
Subcommand Syntax	4-1

Common Parameters	4-2
External and Working Files	4-5
Open and Closed Files	4-6
The VETO Parameter	4-7
The Mask Character	4-9
Editing Source Code Utility Decks	4-9

Common Editing

Functions	5-1
Starting the Editor	5-1
Creating a File	5-4
Entering Subcommands ...	5-6
Getting Help	5-8
Stopping	5-9
Moving the Cursor	5-11
Listing	5-14.1
Paging	5-14.4
Searching	5-15
Inserting	5-22
Deleting	5-30
Replacing	5-38
Marking	5-42
Copying	5-46
Moving	5-61
Undoing	5-65
Creating Multipartition Files	5-67
Text Formatting	5-68

Function Keys

Editing Keys	6-1
CDC Standard Function Keys	6-2
Programmable Function Keys	6-3

Selecting Editor Options ... 7-1

- Changing the Screen
 - Display 7-1
- Changing Tab Settings 7-10
- Changing Line Width 7-12
- Changing the Verify
 - Option 7-13
- Changing the Characters
 - Allowed in a Word 7-14
- Changing How Lines Are
 - Listed in Line Mode 7-15
- Displaying Status
 - Information 7-16

Editing SCU Decks 8-1

- Starting the Editor 8-2
- Opening Decks 8-4
- Closing a Deck 8-10
- Stopping the Editor 8-11

Creating Procedures 9-1

- Structure 9-1
- Subcommands and
 - Functions 9-4
- Examples 9-10
- Calling the Editor from Within
 - a Procedure 9-12

Prologue and Epilogue

- Files 10-1
- Prologue File 10-1
- Epilogue File 10-1

**Using Other Terminals
in Screen Mode 11-1**

- CDC 722 11-1
- CDC 722-30 11-5
- DEC VT100 11-9
- IBM PC 11-13
- Zenith Z19 or Heathkit
 - H19 11-17
 - Zenith Z29 11-22

Appendixes

Glossary A-1

Character Set B-1

**Subcommand Strings That
Define Function Keys C-1**

- Viking 721 Subcommand String
 - Settings C-1
- CDC 722 Subcommand String
 - Settings C-3
- CDC 722-30 Subcommand
 - String Settings C-4
- DEC VT100 Subcommand
 - String Settings C-5
- IBM PC Subcommand String
 - Settings C-6
- Zenith Z19/Heathkit H19
 - Subcommand String
 - Settings C-7
- Zenith Z29 Subcommand String
 - Settings C-9

**Viking 721 Terminal
Settings D-1**

Index Index-1

About This Manual

The CONTROL DATA® Network Operating System/Virtual Environment (NOS/VE) enables you to edit files with the Full Screen Editor (FSE) Utility. FSE enables you to edit files or Source Code Utility (SCU) decks both page by page (screen mode) and line by line (line mode).

This manual explains how to access and use FSE in both screen mode and line mode. Examples and illustrations help you with each step.

Audience

This manual is intended for you if you are familiar with the interactive use of NOS/VE. Some knowledge of how to manipulate NOS/VE files is helpful but is not required. All examples use the CDC® Viking 721 terminal; knowledge of this terminal is helpful but not essential. For information on the interactive use of NOS/VE, refer to the System Command Language (SCL) System Interface manual.

Organization

This manual is one in a set of manuals that describe the System Command Language (SCL). The organization contains brief descriptions of some of the SCL manuals, followed by a detailed description of this manual.

SCL Manuals

The following manuals form the core of the SCL manual set.

- SCL Language Definition
- SCL System Interface
- Full Screen Editor
- SCL Source Code Management
- SCL Object Code Management
- SCL Advanced File Management
- Screen Formatting
- SCL Quick Reference

SCL Language Definition defines the complete SCL language specification. It describes SCL language elements, expressions, variable management, command stream structuring, language-dependent commands and functions, procedure creation, and command list manipulation.

SCL System Interface describes the basic system interface to NOS/VE using SCL. It describes system access, interactive processing, access to online documentation, file and catalog management, job management, tape management, and terminal attributes.

Full Screen Editor describes the text editor available on NOS/VE. Organized in two parts, the manual first describes the basics of editing in screen mode and then more comprehensively describes the editor. Topics included are creating files, replacing text, searching, undoing, creating procedures, and so on.

SCL Source Code Management describes the NOS/VE Source Code Utility, a development tool used to organize and maintain libraries of ASCII source code.

SCL Object Code Management describes the storage and manipulation of units of object code within NOS/VE.

SCL Advanced File Management Usage describes three file management tools: Sort/Merge, the keyed-file utilities, and the File Management Utility (FMU). Sort/Merge sorts and merges records; FMU reformats record data; and the keyed-file utilities copy, display, and create keyed files (such as indexed-sequential files).

Screen Formatting describes the `DEFINE_TERMINAL` command and the statements used to define terminals for use with full screen applications such as the Full Screen Editor.

SCL Quick Reference provides a quick reference for the SCL commands, functions, and statements described in the SCL manual set.

Full Screen Editor

Full Screen Editor is divided into two parts. Part I, Tutorial, describes the basics for the novice user.

Chapter 1 introduces the editor and its capabilities.

Chapter 2 introduces the Viking 721 terminal, which is used exclusively for all examples.

Chapter 3 describes the basics of screen editing such as creating a file, and searching, replacing, and moving around in the file.

Part II, Usage, is for the more experienced user. It describes all the editor capabilities and subcommands. This information is grouped by the functions they perform rather than alphabetically by subcommand names. The subcommand names are listed alphabetically on the inside back cover.

Chapter 4 describes concepts that you will need to know before reading the rest of part II.

Chapter 5 describes some common editing functions (such as starting the editor, inserting, deleting, replacing, and undoing), all of which contain a complete description of the topic.

Chapter 6 describes the function keys and how to change the definitions of the programmable function keys.

Chapter 7 describes how to set various editor options to customize how you use the editor.

Chapter 8 describes how to edit Source Code Utility (SCU) decks.

Chapter 9 describes how to create procedures to be used with the editor and lists special functions to be used within procedures. It also describes how to call the editor from within a procedure.

Chapter 10 describes the editor's prologue and epilogue files.

Chapter 11 describes how to use the editor in screen mode on other supported terminals.

Appendix A contains definitions of terms used in this manual as well as terms not used in this manual, but helpful in understanding NOS/VE.

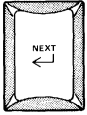
Appendix B lists the ASCII character set supported by NOS/VE.

Appendix C is a listing of the subcommand strings composing the various FSE functions.

Appendix D describes the Viking 721 terminal settings needed to ensure proper operation of FSE.

Conventions

Throughout, this manual uses representations of actual keys to show you when you should press a key. For example,



is represented as



and





is shown as just



When two keys are shown side by side, it means you should hold down the first key while pressing the second. For example,



means hold down  (the 721's shift key) and press .

Sometimes you have to press more than one key in succession to execute a certain function. This is indicated with a + sign. For example,



means hold down  and press : afterwards press .

Vertical bars in the margin indicate changes or additions to the text from the previous revision.

- A dot next to the page number indicates that a significant amount of text (or the entire page) has changed from the previous revision.

Some function keys have labels on both the keyboard and the screen. For example, the key labeled only **F1** on the keyboard has the following screen label:

F1 **MARK**

The screen label is used in this manual.

The keys are sometimes combined. For example, to page backward on the DEC VT100 terminal, you would use:

SHIFT F1 **BKW** + **RETURN**

Within text, cursors are shown as:

█

For example in line mode editing, the cursor is shown as:

The█cursor

In screen mode examples, the cursor is shown as:

```
position
the █
cursor
```

The subcommand descriptions within this manual follow a standard, concise format. The format is described in chapter 4 of this manual and is the same format used for other SCL commands (refer to the SCL Language Definition manual for detailed information). In descriptions of subcommands or parameters, valid abbreviations are shown in parentheses.

Within the formats shown in this manual, UPPERCASE characters represent reserved words; they must appear exactly as shown. Lowercase characters represent names and values that you supply.

Required names and parameters are in **boldface** type. Optional parameters are shown in *italics*.

All numbers are assumed to be decimal unless otherwise noted.

Interactive examples are shown in a type font that resembles computer output. User input within interactive examples is shown in blue.

Additional Related Manuals

For detailed information on diagnostic messages refer to the NOS/VE Diagnostic Messages manual, publication number 60464613, or its online counterpart.

For detailed hardware descriptions of the Viking 721 terminal, refer to the 721-10/20/30 Hardware Reference Manual, publication number 62940020, and the Control Data 721-21/31 Owner's Manual, publication number 62950101.

Ordering Manuals

Control Data printed manuals are available through Control Data sales offices or through:

Control Data Corporation
Literature and Distribution Services
308 N. Dale Street
St. Paul, MN 55103

Submitting Comments

The last page of this manual is a comment sheet. Please use it to give us your opinion on the manual's usability, to suggest specific improvements, and to report technical or typographical errors. If the comment sheet has already been used, you can mail your comments to:

Control Data Corporation
Publications and Graphics Division ARH219
4201 Lexington Avenue North
St. Paul, MN 55126-6198

Please indicate whether or not you want a written response.

Additionally, if you have access to SOLVER, an online facility for reporting problems, you can use it to submit comments about the manual. When entering your comments, use SC8 as the product identifier.

This chapter is an overview of some of the basic capabilities of the Full Screen Editor.



Using the NOS/VE Full Screen Editor on most video display terminals, you can display a page of text on the screen, move through the file page by page, and make most of your changes with the touch of a key. This is called screen editing.

If you are working on a printing terminal, you will use the other type of editing the editor is capable of, line editing. In contrast to screen editing, you see only a limited number of lines at any one time. Line editing is also available for use on video display terminals.

This manual describes how to edit files. Most of the features apply to both files and Source Code Utility (SCU) decks. Information specific to decks is discussed in chapter 8, Editing SCU Decks.

Using the editor you can:

- Display and edit more than one file at a time.
- Search for and replace text according to the column in which the text appears.
- Move or copy parts of a file to the same or another file.
- Undo changes you've made to a file.
- Search for and replace words.
- Create procedures containing both editor subcommands and SCL commands.
- Format text.

When you make changes using the editor, the editor uses a copy of your file, changes it, and then rewrites your file. When the editor works with a copy of your file, it automatically deletes all blanks after the last visible character in each line. For most users, this does not affect the use of the file. In fact, it can actually decrease the file size (depending on how many blanks you had at the end of lines in your file). Some utilities, however, may use a specific file structure that attaches significance to blanks at the end of lines. If you need to edit these files, you need to be aware of the requirements of the file and then see if the Full Screen Editor can meet those requirements before editing. Examples of such files are the bound version of manuals that are called by the EXPLAIN command and files used by Sort/Merge.

This chapter introduces the Viking 721 terminal and lists other supported terminals.









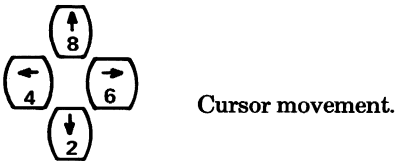
Throughout this manual, examples and explanations apply to the Viking 721 terminal.



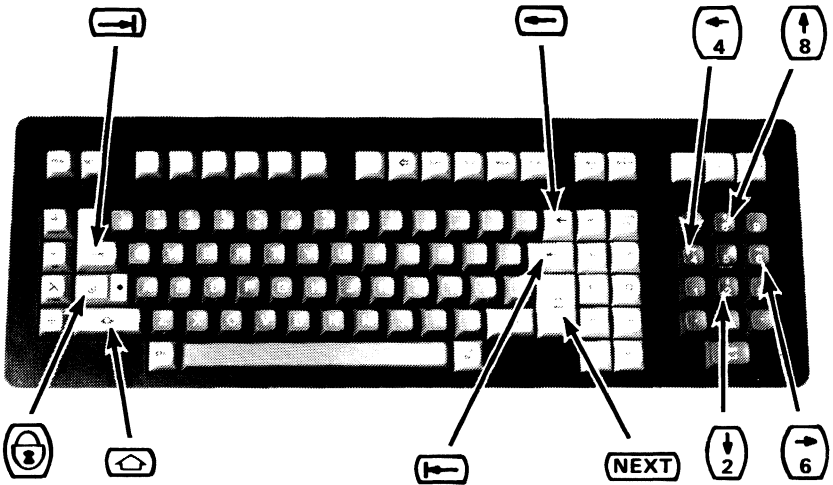
The Viking 721 terminal must be set up correctly to log in to NOS/VE and use the editor. Appendix D shows how the terminal should be set up; however, if your terminal has been used with NOS/VE before, try using it without changing any of the settings.

The Viking 721 key symbols mean:

Key	Meaning
	Return, carriage return, or new line.
	Backspace.
	Shift.
	Shift lock.
	Tab forward.
	Tab backward.



The locations of these keys are:



The editor, in screen mode, supports several other terminals. These terminals are:

- CDC 722
- CDC 722-30
- DEC VT100 or equivalent
- IBM PC or equivalent
- Zenith Z19
- Zenith Z29
- Heathkit H19

If you are using one of these terminals in screen mode, refer to chapter 11, Using Other Terminals in Screen Mode, whenever needed to determine what keys to use to perform the functions described in the following chapters. Information on how to create a file that allows you to use other terminals for screen editing can be found in the Screen Formatting manual.



This chapter describes how to perform the basics of screen mode editing on a Viking 721 terminal using function keys and a few subcommands. For comprehensive descriptions of these functions, refer to part II, Usage.

Getting Started	3-1
Creating a File	3-6
Moving around the Screen	3-8
Changing the Screen Content	3-9
Moving to the End of a Line	3-11
Moving around within the File	3-12
Increasing the File Length	3-12
Moving to the First or Last Line	3-15
Moving from Screen to Screen	3-16
Entering Subcommands	3-19
Getting Help	3-19
Deleting	3-23
Deleting Characters	3-23
Deleting Lines	3-24
Deleting Words	3-24
Deleting Blocks of Empty Lines	3-26
Inserting	3-27
Inserting Characters	3-27
Inserting Lines	3-28
Inserting Words	3-29
Inserting Blocks of Empty Lines	3-30
Searching	3-32
Searching for a Text String	3-32
Searching for the Next Occurrence	3-35
Searching for All Occurrences	3-36
Marking	3-39
Copying	3-41
Moving	3-44
Undoing	3-46
Stopping an Editing Session	3-47



Getting Started

To get the editor started, enter the EDIT_FILE command; its format is:

```
EDIT_FILE (EDIF)
  FILE=file
  INPUT=file
  OUTPUT=file
  PROLOG=file
  STATUS=status variable
```

The FILE (F) parameter specifies the name of the file you want to edit and is required. If the file you specify does not exist or is busy, a new file is created. The other parameters are described in part II, Usage.

For example, to edit the permanent file ZAP, which already exists, enter either:

```
/edit_file file=$user.zap
```

or

```
/edif $user.zap
```

The following prompt appears:

```
ef/
```

This is the line mode prompt. At this point, you can enter any FSE subcommands to edit your file in line mode or you can enter screen mode. The rest of part I, Tutorial, describes screen mode.

To start screen mode, enter the `ACTIVATE_SCREEN` subcommand specifying the terminal model you are using. The format of the subcommand is:

```
ACTIVATE_SCREEN (ACTS)
  MODEL=name
  STATUS=status variable
```

The `MODEL (M)` parameter specifies the terminal model you are using. You may set the terminal model with the `SET_TERMINAL_ATTRIBUTES` command and include it in your user `PROLOG` so you won't need to set it again. If it has not been specified using `SETTA` or on a previous `ACTIVATE_SCREEN` or `SET_SCREEN_OPTIONS` subcommand, this parameter is required. Valid entries are:

Entry	Terminal
CDC721	CDC Viking 721
CDC722	CDC 722
CDC722_30	CDC 722-30
VT100	DEC VT100
PC_CONNECT	IBM PC or equivalent
Z19	Zenith Z19 or Heathkit H19
Z29	Zenith Z29

For example, to specify the Viking 721 terminal, enter:

```
ef/activate_screen model=cdc721
```


A screen appears showing the beginning of your file. In the following figure, permanent file ZAP appears as:

```

File: ZAP Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Fozy score and seven.

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information Like:

o Text

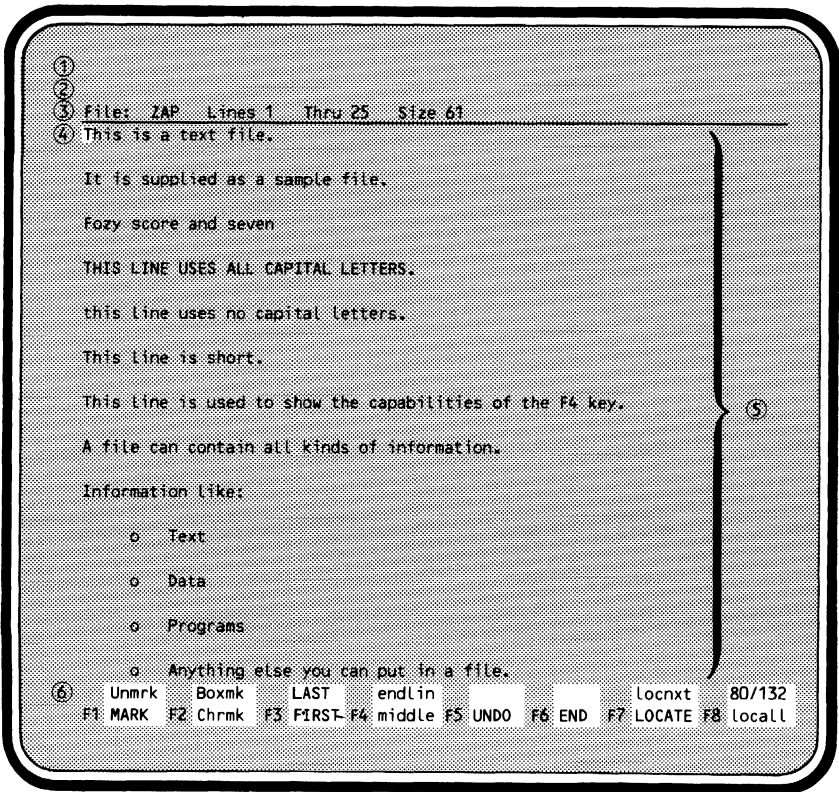
o Data

o Programs

o Anything else you can put in a file.
Unmrk  Boxmk  LAST  endlin  80/132
F1 MARK  F2 Chrmk  F3  FIRST  F4 middle  F5  UNDO  F6  END  F7  LOCATE  F8  Locall

```

The format of each screen displayed by FSE is:



① Subcommand Line

The line on which you enter subcommands. To move the cursor to this line, press **(HOME)**; to return to the file, press **(NEXT)**.

② Message Line

The line on which FSE displays messages. These messages might be informative messages, error messages, or prompts asking you to do something.

③ File Header

The line containing the file name, the lines displayed on the screen, and the file size.

- ④ Cursor Your exact position in the file.
- ⑤ File Text The contents of the file.
- ⑥ Programmable Function The labels currently assigned to function keys
Key Prompts F1 through F8. These keys are described later
 in this manual.

To get more editor status information, use the `DISPLAY_EDITOR_STATUS` subcommand described in chapter 7, *Selecting Editor Options*.

Creating a File

To create a file, start the editor with the EDIT_FILE (EDIF) command specifying the name of a file that does not exist. A screen appears with the name of the file displayed.

Create permanent file BERT on your USER catalog by entering:

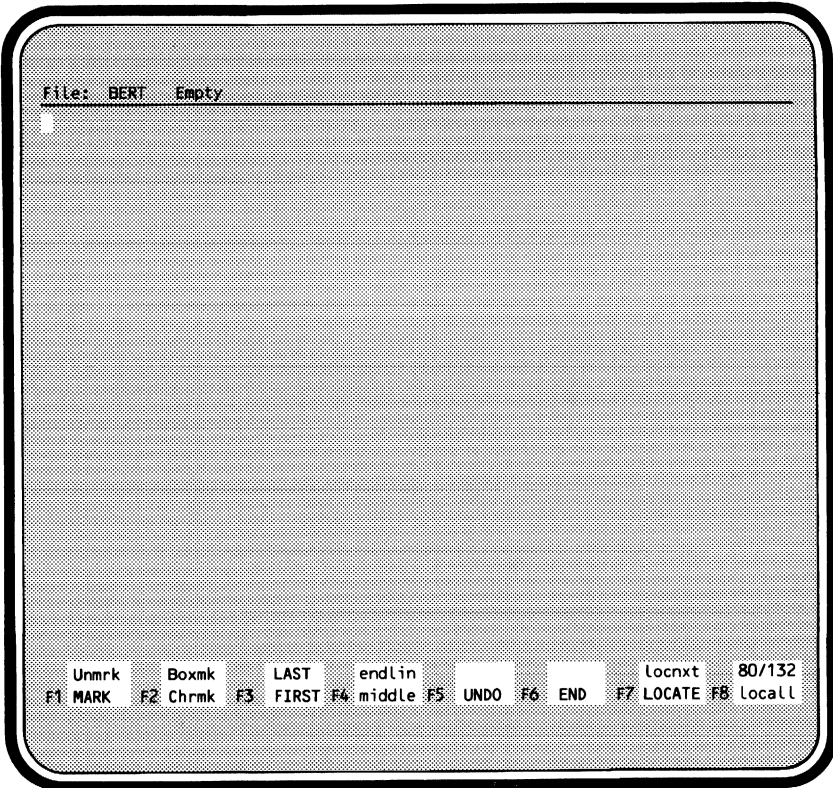
```
/edif $user.bert
```

(assuming file BERT does not already exist).

Enter the ACTIVATE_SCREEN subcommand to start screen mode editing on a Viking 721 terminal.

```
ef/activate_screen model=cdc721
```

The following screen appears.



The editor positions the cursor on the first line. You can then type in whatever you want to appear in the file. Each time you press the **(NEXT)** key, the editor positions the cursor at the beginning of the next line. At the same time, it updates the file size on the file header line. Enter a few lines of text into file BERT, pressing the **(NEXT)** key at the end of each line. For a sample of what to enter, refer to file ZAP shown earlier. Your file will look similar to:

```

File: BERT  Lines 1  Thru 5  Size 5
-----
This is a text file.

It is supplied as a sample file.

Fozy score and seven

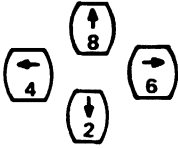
```


Unmrk Boxmk LAST endlin locnxt 80/132
F1 MARK F2 Chrnk F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 locall

File BERT is now five lines long (blank lines are counted when they are at the beginning of the file or are surrounded by text).

Moving around the Screen

To move the cursor around the screen, use the arrow keys on the keypad to the right of the main keyboard. The numbers on the keys are 2, 4, 6, and 8 and appear as:



The direction which the arrow points corresponds to the direction the cursor moves when the key is pressed. You do not have to hold down the  key to use these keys. Also, if you hold the arrow keys down they automatically repeat. You can use the arrow keys to move the cursor off the screen to the left and the right. The cursor reappears on the opposite side of the screen. Try moving the cursor up and down. If you move the cursor off the screen, it will always return.

Changing the Screen Content

If you want to change what's on the screen, use the cursor positioning (arrow) keys to move the cursor to the place where you want to make the change and type over what is there. The change is made as you type.

For example, to correct the errors in the file BERT, position the cursor on the character in error:

Fo~~z~~y score and seven

Press:

u

The following results:

Four~~z~~y score and seven

The z is replaced with u and the cursor moves one position to the right (in this example, positioning the cursor at the y).

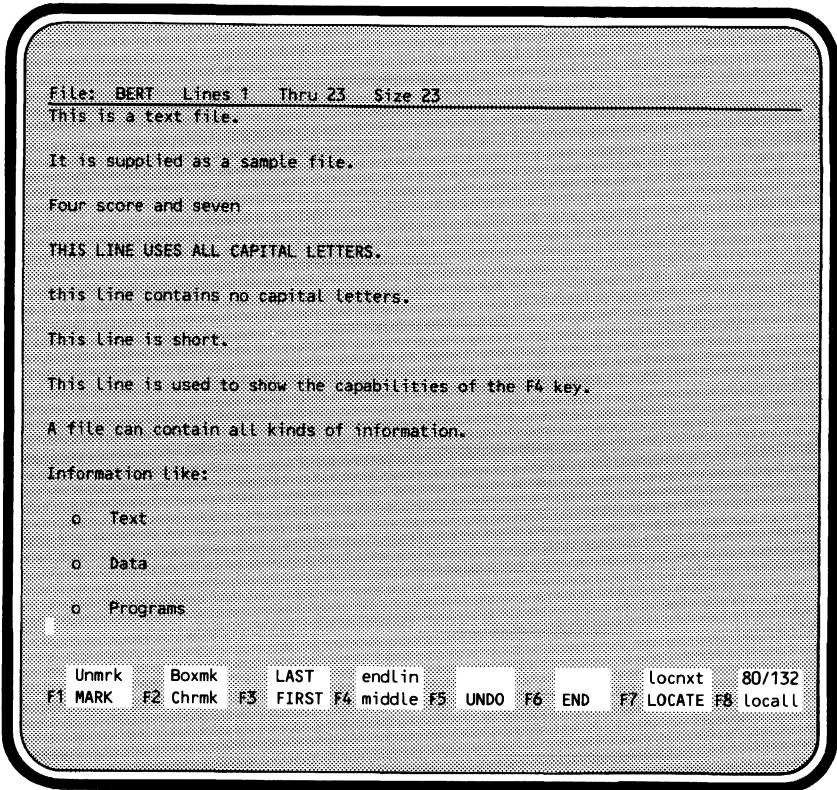
Press:

r

The result is:



Four~~z~~ score and seven

Practice using the character keys, the cursor positioning keys, and the **(NEXT)** key by filling up the first 23 lines of file BERT with text.



File BERT is used in the examples that follow to explain other concepts. You can use the file you have created to practice what you learn.


Moving to the End of a Line

To move the cursor to the end of a line, you do not need to hold down the cursor positioning key until it reaches the end of the line. Use  F4 **endlin** instead. This moves the cursor to the end of the current line. The  F4 **endlin** key is the function key labeled F4 that is located in the top row of the keyboard. It also is shown on the screen as the fourth highlighted (inverse video) box.

For example, to move the cursor in file BERT to the end of the following line

This line is used to show the capabilities of the F4 key.

position the cursor on the line and press:

 F4 **endlin**

The result is:

This line is used to show the capabilities of the F4 key.

Moving around within the File

In screen mode, the editor automatically increases the file's length as you add lines to the file. The editor also allows you to easily move around within the file once the file is too large to fit on the screen. To move around within the file you use function keys. Some function keys are located above the regular keys; others are located on the left and right. These keys enable you to:

- Move through the file one screen at a time.
- Move quickly to the first or last line of the file.

Increasing the File Length

Once you have entered the last line on the screen into a file and pressed the **(NEXT)** key, the editor automatically repositions the lines displayed on the screen so you can continue entering lines. The editor positions the last line on the screen (the 25th line on the Viking 721 terminal) in the middle of the screen. You then have half the screen on which to enter lines.

For example, enter into file BERT a blank line as line 24 and the following line as line 25:

- o Anything else you can put in a file.

When you press **(NEXT)** at the end of line 25, the editor displays the following screen:

```

File: BERT Lines 13 Thru 25 Size 25
-----
This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:

o Text

o Data

o Programs

o Anything else you can put in a file.

Unmrk   Boxmk   LAST   endlin   locnxt   80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO  F6 END   F7 LOCATE F8 locall

```

Each time you fill up the screen, the same process takes place. Once you get to the bottom of the screen and press **(NEXT)**, the editor repositions the file.

Enter lines into file BERT until the file size is at least 60 lines. This will allow you to try the examples that follow. The following text can be entered.

This is the second screen of file BERT.

See the top line of this screen?

It was the bottom line of the first screen.

a

b

c

d

e

f

g

h

If you press FWD again, this line will appear at the top of the next screen.

j

k

l

m

n

o

Moving to the First or Last Line

If you are in the middle of a file and want to get back to the first line, press the function key:

F3 **FIRST**

The screen then shows the first lines of the file.

If you need to get to the last line of the file, press:

 F3 **LAST**

This positions the cursor on the last line and vertically centers the line on the screen.

Try moving to the beginning and end of file BERT.

Moving from Screen to Screen

To move through your file screen by screen, use the **(FWD)** and **(BKW)** keys located on the left side of the regular keyboard. **(FWD)** moves you forward one screen. **(BKW)** moves you backward one screen.

For example, you're at the following screen in file BERT.

```

File: BERT Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Four score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:

  o Text

  o Data

  o Programs

  o Anything else you can put in a file.
Unmrk  Boxmk  LAST  endlin  locnxt  80/132
F1 MARK  F2 Chrmk  F3 FIRST F4 middle F5 UNDO  F6 END  F7 LOCATE F8 locall

```

To move forward to the next screen, press:

(FWD)

The next screen appears. Notice that the last line of the previous screen is now the top line of this screen.

```

File: BERT Lines 25 Thru 49 Size 61
o Anything else you can put in a file.

This is the second screen of file BERT.
See the top line of this screen?

It was the bottom line of the first screen.

a
b
c
d
e
f
g
h

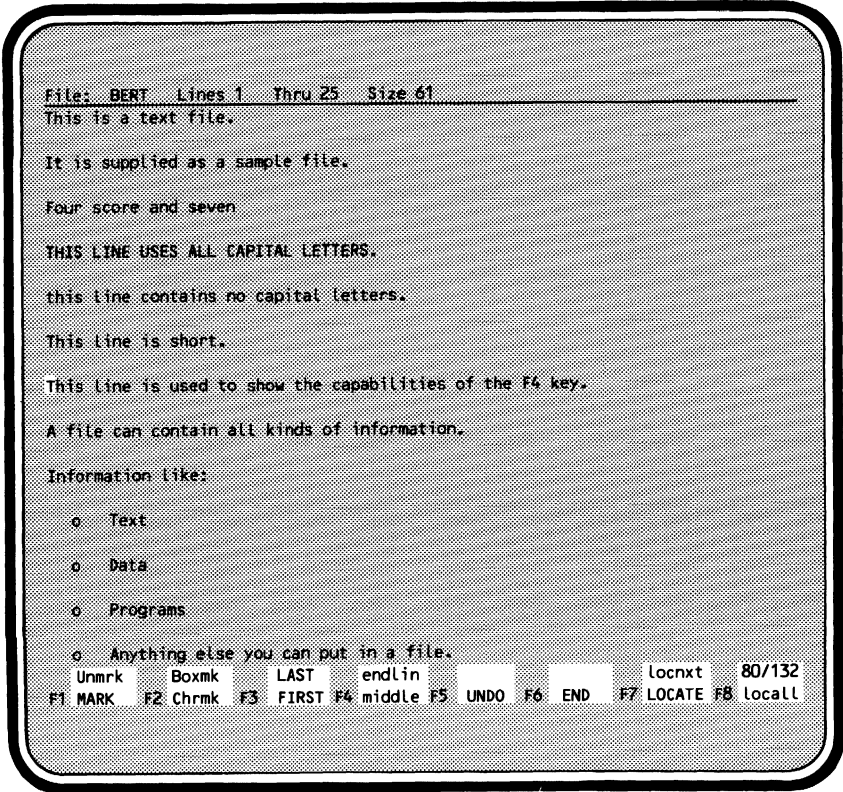
If you press FWD again, this line will appear at the top of the next screen.
Unmrk  Boxmk  LAST  endlin  Locnxt  80/132
F1 MARK  F2 Chrmk  F3  FIRST F4 middle F5  UNDO  F6  END  F7 LOCATE F8 Locall

```

To move back to the previous screen, press:

(BKW)

You are returned to where you started.



Entering Subcommands

Most of the basic editing functions can be done using function keys. However, there will be times when the function keys provided do not meet your editing needs. It is then that you will need to enter a subcommand.

To enter subcommands when screen editing, press:

(HOME)

The **(HOME)** key is on the keypad to the right of the main keyboard in the middle of the arrow keys (the **(5)** key). **(HOME)** moves the cursor to the subcommand line (the top line of the screen). Anything you enter on this line is processed when you press **(NEXT)**.

When you enter a subcommand, the line and column at which the cursor was positioned when you pressed **(HOME)** is used as the current position. Editor subcommands are listed on the inside of the back cover. The only subcommands described in the Tutorial part of this manual are HELP, QUIT, ACTIVATE_SCREEN, and one form of SET_SCREEN_OPTIONS.

Getting Help

An online help file is available through the editor. If you can't recall how a subcommand works, press

(HELP)

or press the **(HOME)** key to move the cursor to the subcommand line and enter the subcommand:

help

Both access a file containing descriptions of the editor subcommands and their parameters. When you press

(HELP)

or enter

help

you are positioned at the beginning of the HELP file. You can then page through the file by using the **(FWD)** and **(BKW)** keys described earlier in this chapter.

If you want to be positioned at a description of a specific subcommand, enter the HELP subcommand followed by the name of the subcommand for which you need help. For example, if you were editing file BERT:

```

File: BERT Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Four score and seven
THIS LINE USES ALL CAPITAL LETTERS.
this line contains no capital letters.
This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:

o Text
o Data
o Programs
o Anything else you can put in a file.
Unmkr  Boxmk  LAST  endlin  locnxt  80/132
F1 MARK  F2 Chrmk  F3  FIRST F4 middle F5  UNDO  F6  END  F7 LOCATE F8 localL

```

To get help on the INSERT_LINES subcommand, press

HOME

and enter:

help insert_lines

The HELP file appears on the bottom half of the screen with the cursor positioned at the description of the INSERT_LINES subcommand.

```

Use EDIT key to erase Help file
File: BERT Lines 1 Thru 12 Size 61
-----
This is a text file.

It is supplied as a sample file.

Four score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

File: SCU_EDITOR_HELP Lines 676 Thru 687 Size 1761
-----
INSERT_LINES

Purpose: Inserts lines of text at a specified location in
         the current file.

Format:  INSERT_LINES (INSL or I)
         NEW_TEXT=string
         PLACEMENT=keyword value
         INSERTION_LOCATION=integer
         UNTIL=string
         STATUS=status variable

Unmrk   Boxmk   LAST   endlin   Locnxt   80/132
F1 MARK  F2 Chrmk  F3 FIRST F4 middle F5 UNDO  F6 END   F7 LOCATE F8 Locall

```

To continue reading the help file, press:

(FWD)

To return to editing the file or files you were editing, press:

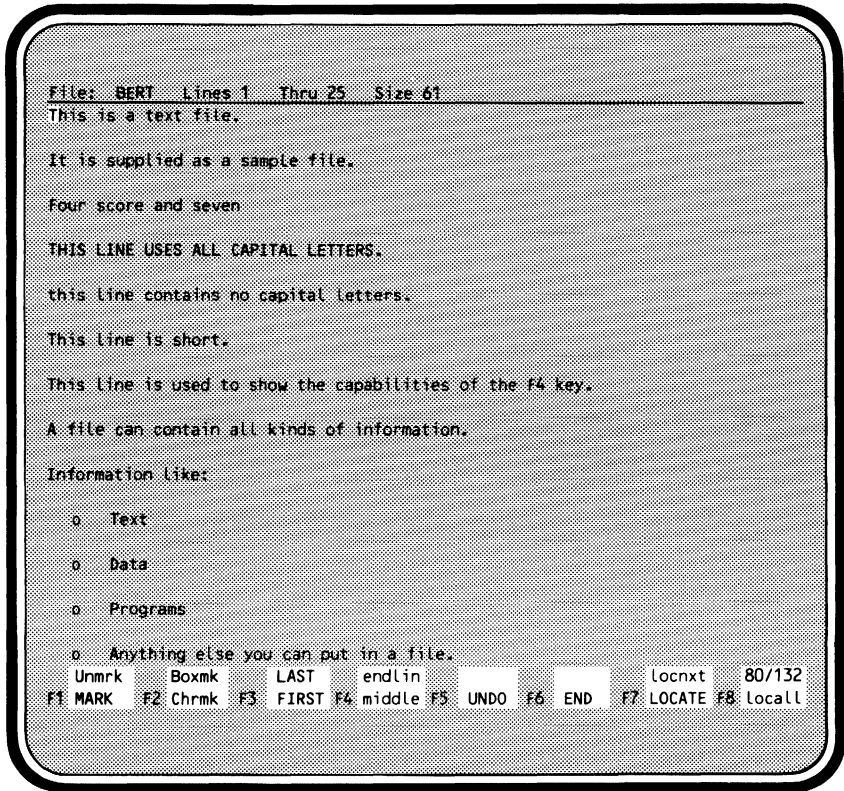
(EDIT)

This clears the screen of the HELP file.

In the previous example, to return to editing just file BERT, press

EDIT

and the description of INSERT_LINES is gone.



Deleting

Using function keys, you can delete characters, lines, words, and groups of blank lines. After reading about each topic, use a practice file to try them out.

Deleting Characters

To delete individual characters, use the **DELETE** key located above and to the right of the regular keyboard. Position the cursor on the character you want to delete, and press:

DELETE

For example, to delete the extra `n` in `Clinnic`,

`Corby Jones Clinnic`

position the cursor on the `n` to be deleted

`Corby Jones Clin@nic`

and press:

DELETE

The following results:

`Corby Jones Clinic`

Deleting Lines

To delete lines, use the  **DELETE** keys. For example, to delete the line

```
This line doesn't belong
```

from

```
Dr. Hugo Quackenbush
The Marx Clinic
This line doesn't belong
10 Downing Street
Minot, NM 77834
```


position the cursor anywhere on the line to be deleted:

```
This line doesn't belong
```

and press  **DELETE**. The following results:

```
Dr. Hugo Quackenbush
The Marx Clinic
10 Downing Street
Minot, NM 77834
```

Deleting Words

To delete a word, use  F10 **delwrd**. When you start the editor, the prompts for programmable function keys F9 through F16 are not displayed. To display the programmable function key prompts for F9 through F16, press

 **HOME**


and enter:

```
set_screen_options menu_row=2
```

The screen then displays two rows of prompts:

Unmrk	Boxmk	LAST	endlin					Locnxt	80/132
F1 MARK	F2 Chrmk	F3 FIRST	F4 middle	F5 UNDO	F6	END	F7 LOCATE	F8 locall	
delel	delwrd					Move	dedent	center	
F9 insel	10 inswrd	11 Break	12 Join	13		14 Copy	15 INDENT	16 FORMAT	

Deleting Blocks of Empty Lines

To delete a group of empty lines, use  F9 **dele**. This deletes the current blank line and any following blank lines until a nonblank line is encountered.

For example, the blank lines between the first and the second line need to be deleted:

The first line.

The second line.

Position the cursor on the first blank line to be deleted:

The first line.



The second line

Press:

 F9 **dele**

The blank lines are deleted resulting in:

The first line.

 The second line.

Inserting

Using function keys, you can insert text into your file including characters, lines, words, and blocks of blank characters. Actually, on the Viking 721 terminal, the text itself isn't inserted; space is inserted enabling you to enter your text. Other supported terminals enable you to enter what's called insertion mode, which allows you to insert the actual characters rather than spaces over which you type.

Inserting Characters

To insert a character, use the **(INSRT)** key located above and to the right of the regular keyboard. To do so, move the cursor to the character before which the new character is to appear and press **(INSRT)**. This inserts a blank character over which you can type the new character.

For example, a **B** needs to be inserted between the **Y** and the **I** in

CYIL

Position the cursor on the **I** :

CYIL

Press:

(INSRT)

This inserts a blank:


CY IL

You can then type the **B** over the blank.

CYBIL

(After you type **B**, the cursor moves to the next character.)

Inserting Lines

The quickest way to insert a line is to use the  and **INSRT** keys. When you press



a blank line is inserted before the current line. You can then type in the text of the new line.

For example, to insert

Corby Jones Clinic


between

Dr. Leo Miller
2703 Jones Circle


position the cursor on the line before which you want the new line to appear (in this example, the line that reads 2703 Jones Circle) and press:



A blank line is inserted:

Dr. Leo Miller

 2303 Jones Circle

Then, just type the new line of text.

Dr. Leo Miller
 Corby Jones Clinic 
 2703 Jones Circle

If you want to insert several lines of text, press



several times.

Inserting Words

To insert a word, use F10 **inswrđ**. F10 **inswrđ** inserts 32 blank characters over which you can type the word or words to be inserted.

For example, to insert

Jones

between Corby and Clinic ,

Corby Clinic



position the cursor to the C in Clinic and press:

F10 **inswrđ**

32 blanks are then inserted over which you can type the new word:

Corby Jones

Clinic

To delete the remaining blanks, position the cursor on the first blank you want deleted and press  F10 **delwrđ**. This deletes all blanks until a nonblank character is encountered. In this example, move the cursor to the second blank character after Jones and press  F10 **delwrđ**. The following results:

Corby Jones Clinic

Inserting Blocks of Empty Lines

To insert a block of empty lines, use F9 **insel** . The number of blank lines inserted depends on the size of the screen. When you press F9 **insel** , empty lines are inserted before the current line, leaving two lines of text at the top and bottom of the screen.

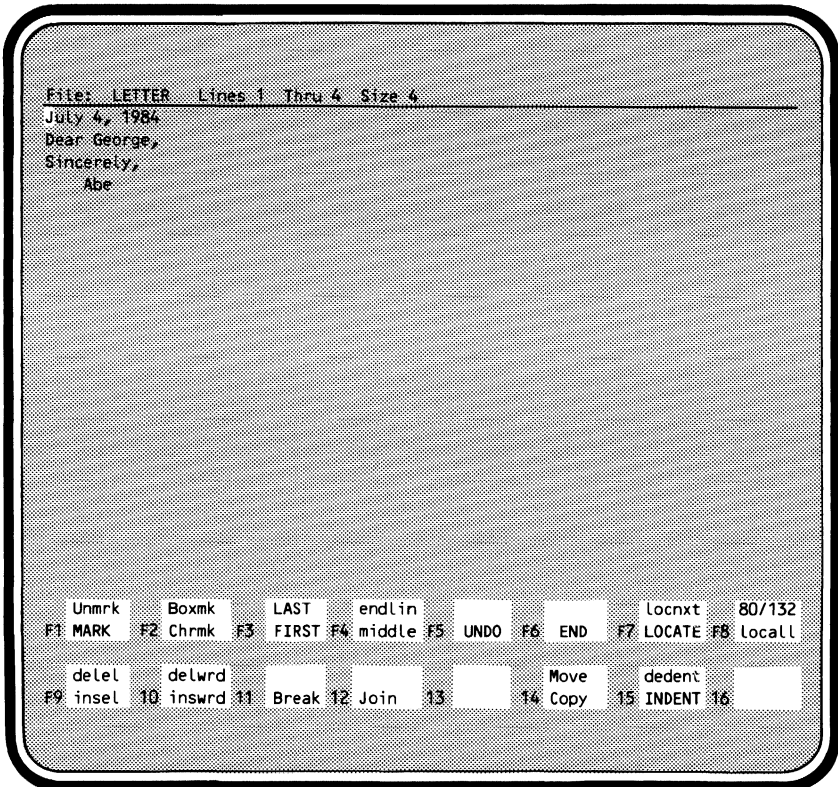
For example, to insert empty lines between the line that reads

Dear George,

and the line that reads

Sincerely,

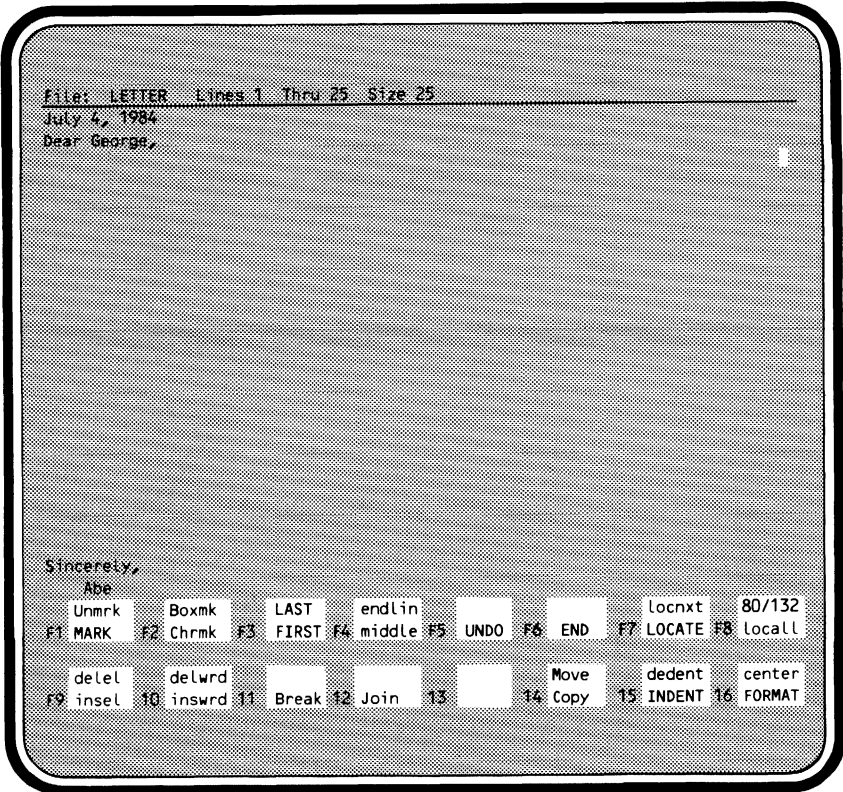
on the following screen:



position the cursor on the line that says Sincerely, and press:

F9 **inseL**

The following screen results:



Searching

Using function keys, you can perform three different types of searching. You can do searches for a text string, searches for the next occurrence of a string you specified earlier, and searches for all occurrences of a string.

Searching for a Text String

The easiest way to search for text is to use F7 **LOCATE**. When you press F7 **LOCATE**, the cursor moves to the subcommand line and you are prompted to enter the text you want to locate. The editor searches for the next occurrence of the text and, if found, positions the cursor at the first character of the string. You should enter the text exactly the way you want to find it in the text. For example, to find the word `Fred` in a file, you cannot enter `FRED` or `fred`; it must be entered as `Fred`.

Another example: you are editing file `BERT` and want to locate `file`. Press:

F7 **LOCATE**

The cursor moves to the subcommand line and you are prompted to enter the text you want to locate.

```

Enter Search String
File: BERT Lines 1 Thru 25 Size 61
-----
This is a text file. (Assuming this is the current position.)

It is supplied as a sample file.

Four score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:

o Text

o Data

o Programs

o Anything else you can put in a file.
Unmrk Boxmk LAST endlin locxt 80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO f6 END F7 LOCATE F8 locall

```

Type file and press **(NEXT)**. The editor searches forward from the current line (in this case we'll say it was the top line) and, when the string is found, positions the cursor at the beginning of the string.

```
File: BERT Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Four score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.



A file can contain all kinds of information.

Information like:

o Text
o Data
o Programs
o Anything else you can put in a file.

Unmrk   Boxmk   LAST   endlin   locnxt   80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 locall
```


Searching for the Next Occurrence

To search for the next occurrence of a string in a file, press  F7 **locnxt**. In this example, to locate the next occurrence of file (assuming file was the last string specified using F7 **LOCATE**), press  F7 **locnxt**. The cursor is positioned at the next occurrence.

```

File: BERT Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Four score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:


o Text

o Data

o Programs

o Anything else you can put in a file.
Unmrk  Boxmk  LAST  endlin  locnxt  80/132
F1 MARK  F2 Chrmk  F3 FIRST F4 middle F5 UNDO  F6 END  F7 LOCATE F8 locall

```

To locate any subsequent occurrences, just keep pressing  F7 **locnxt**. If there are no more occurrences of the string, an error message is displayed.

Searching for All Occurrences

To search for all occurrences of a string, use **F8** **LocalL** . When you press **F8** **LocalL** , you are prompted to enter the string you want to find. You can then enter the string you want to find or, to locate the previously specified string, press **(NEXT)**.

For example, to locate all occurrences of the string `move` within file `FRED`, press:

F8 **LocalL**

You are prompted to enter the text for which the editor is to search:

```

Enter Search String
File: FRED Lines 1 Thru 25 Size 120
-----
This is file FRED.

It is also supplied as a sample file.

This file explains how to move around the file or how to move text.

To move around the file, use the arrow keys, the FWD and BKM keys, and so on.

To move text, use the FUNCTION key defined as MOVE or the MOVE text subcommand.

To move text from one file to another, you can use a split screen. With split
screen editing you can display more than one file on the screen. This enables
you to see more than one file at a time.

To split the screen, enter:

    set_screen_options splits=2

This will put the file you were editing on the top and bottom of the screen.
You can then enter the EDIT_FILE command to create an empty file or to access
another existing file on the bottom half of the screen.

Once the files are on the screen you can move marked text from one file to the
other using the MOVE function key.
Unmrk   Boxmk   LAST   endlin   locnxt   80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 localL

```

Enter:

move

The directory of all the lines in the file containing move is displayed:

```

Press Next/Return to continue, or a line number to stop and position.
Displaying Located Lines
-----
 6 This file explains how to move around the file or how to move text.
 8 To move around the file, use the arrow keys, the FWD and BKW keys, and s
10 To move text, use the FUNCTION key defined as MOVE or the MOVE text subc
12 To move text from one file to another, you can use a split screen. With
24 Once the files are on the screen you can move marked text from one file
34 to move the marked text.
47 moves the word cannonball
117 shows the moved text.

Unmrk   Boxmk   LAST   endlin   [ ]   [ ]   Locnxt   80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 Local1

```

To look at the next screen of located lines, press **(NEXT)**. If there isn't a next screen, pressing **(NEXT)** positions the cursor on the last line of the listed lines. If you want to position the cursor at a line contained in the list of lines, enter the line number on the subcommand line or position the cursor on the line and press **(NEXT)**. You are returned to the file text with the line you selected.

For example, to go to line 10, enter the following on the subcommand line:

```
10 + (NEXT)
```

The following screen appears:

```
File: FRED Lines 1 Thru 25 Size 120
-----
This is file FRED.

It is also supplied as a sample file.

This file explains how to move around the file or how to move text.

To move around the file, use the arrow keys, the FWD and BKW keys, and so on.

To move text, use the FUNCTION key defined as MOVE or the MOVE text subcommand.

To move text from one file to another, you can use a split screen. With split
screen editing you can display more than one file on the screen. This enables
you to see more than one file at a time.

To split the screen, enter:

    set_screen_options splits=2

This will put the file you were editing on the top and bottom of the screen.
You can then enter the EDIT_FILE command to create an empty file or to access
another existing file on the bottom half of the screen.

Once the files are on the screen you can move marked text from one file to the
other using the MOVE function key.
  Unmrk   Boxmk   LAST   endlin   Locnxt   80/132
F1 MARK  F2 Chrmk  F3 FIRST F4 middle F5 UNDO  F6 END   F7 LOCATE F8 locall
```

Marking

To use some of the other functions provided by the function keys, you need to know how to mark text. Marking tells the editor which text to use with a later operation, such as a copy or move function.

Using function keys, you can mark lines of text or a sequence of character strings. To mark lines, use F1 **MARK**. This highlights text in inverse video on terminals with that capability. For example, to mark the line

```
This is the line to mark.
```

in the following text,

```
first line
second line
This is the line to mark
fourth line
fifth line
```

position the cursor anywhere on the line to be marked:

```
This is the line to mark
```

Press:

```
F1 MARK
```

The line is highlighted to show you that it is marked:

```
first line
second line
This is the line to mark
fourth line
fifth line
```

To mark a range of lines, press F1 **MARK** twice: once on the first line to be marked and again on the last line to be marked. For example, to mark all the lines from the second line to the fourth, position the cursor on the second line:

```
First Line
Second line
Third line
Fourth line
Fifth Line
```

Press:

```
F1 MARK
```

The second line is highlighted:

```

First line
Second line
Third line
Fourth line
Fifth line

```

Move the cursor to the fourth line and press F1 **MARK** again. All lines from the second through the fourth are highlighted:

```

First line
Second line
Third line
Fourth line
Fifth line

```

You can also mark strings of characters using F2 **Chrmk** . This also highlights the characters you have marked. For example, to mark the phrase

marked text,

in the following text,

```

If you want to move marked
text, you should use the
MARK function.

```

position the cursor on the m in marked:

If you want to move marked

Press F2 **Chrmk** The m in marked is highlighted:


If you want to move marked

Move the cursor by pressing the arrow keys to the second t in text and press F2 **Chrmk** . All characters between the two markers are now highlighted as well:

```

If you want to move marked
text you should use the
MARK function.

```

The lines or characters you mark will stay marked until you explicitly unmark them, mark some other text, move the marked text, or until you stop the editor. To unmark text, use the  F1 **Unmrk** keys. For example, to unmark the text marked previously with F2 **Chrmk** key, just press:

 F1 **Unmrk**

The lines are no longer marked and no longer highlighted.

Copying

To copy the current line to the next line, just press **F14** **(COPY)**; marking is not necessary (an example of this procedure is provided later in this section).

To copy text other than the current line, use **F1** **MARK** or **F2** **Chrmk** and **F14** **(COPY)**. Mark the text you want to copy. (Marking is described earlier in this chapter.) Position the cursor at the line or character before which you want the text to appear and press:

F14 **(COPY)**

For example, to copy

This is the line to copy

to just before the last line of the following text,

Don't copy this line
 This is the line to copy
 Don't copy this line
 Don't copy this line

position the cursor on the line to be copied and press **F1** **MARK**. The line is marked:

This is the line to copy

Position the cursor on the line before which the line is to be copied:

Don't copy this line
 This is the line to copy
 Don't copy this line
 Don't copy this line

Press **F14** **(COPY)**. The following results:

Don't copy this line
 This is the line to copy
 Don't copy this line
 This is the line to copy
 Don't copy this line

The marked text remains marked after the copy is complete. Use

(←) **F1** **Unmrk** to unmark the text.

If you have marked characters to be copied, position the cursor to the character before which the copied text is to appear. For example, to copy the word `seven` in

```
Four seven score
and years ago,
```

to after the word `and`, position the cursor on the space in front of `seven`,

```
Four seven score
```

press:

```
F2 Chrmk
```

The character is highlighted:

```
Four seven score
```

Then, move the cursor to the `n` in `seven` and press:

```
F2 Chrmk
```

The characters from the blank to the `n` in `seven` are highlighted:

```
Four seven score
```

To copy the marked characters to after the word `and`, move the cursor to the location you want them copied:

```
Four seven score
and years ago,
```


Press F14 **COPY** .

The word seven is copied:

Four **seven** score
and **seven** years ago,

The marked characters remain highlighted until you unmark them, mark different text, or stop the editor.

To copy the current line to the next line, press F14 **COPY** . No marking is needed. For example, to copy the line

Repeat this line

from the following text,

First line
Second line
Repeat this line
Third line
Fourth line


position the cursor on the line to be copied:


First line
Second line
Repeat this line
Third line
Fourth line

Press F14 **COPY** . The line is copied to the next line:

First line
Second line
Repeat this line
Repeat this line
Third line
Fourth line

Moving

To move lines or characters, use F1 **MARK** or F2 **Chrmk** and  F14 **Move** (information on how to display function key prompts for the F9 through F16 keys is included in the Deleting section of this chapter). You first need to mark the text to be moved. Then, position the cursor to the point you want the text to appear and press:

 F14 **Move**

For example, to move the line that reads

Move this one

to just before the last line of the following text,

```
Don't move this line
Move this one
Don't move this line
Don't move this line
```

position the cursor on the line to be moved.

```
Move this one
```

Press F1 **MARK**. The line to be moved is highlighted.

```
Move this one
```

Position the cursor on the line before which the moved line is to appear.

```
Don't move this line
Move this one
Don't move this line
Don't move this line
```


Press  F14 **Move**. The following results.

```
Don't move this line
Don't move this line
Move this one
Don't move this line
```

If you have marked characters to be moved, position the cursor at the character before which the moved text is to appear. For example, to move the word `seven` in

Four `seven` score
and `years ago`,

to after the word `and`, position the cursor on the space in front of `seven`:

Four  `seven` score

Press:

F2 

The character is highlighted:

Four  `seven` score



Then, move the cursor to the `n` in `seven` and press:

F2 



The characters from the blank to the `n` in `seven` are highlighted:

Four  `seven` score


To move the marked characters to after the word `and`, move the cursor to the location you want them moved:

Four  `seven` score
and  `years ago`,

Press:

 F14 

The word `seven` is moved to the destination, disappearing from its former location:

Four `score`
and  `seven` years ago,

Undoing

To undo a change that you have made during your current editing session, use F5 **UNDO** . Each pressing of F5 **UNDO** cancels all changes you made to your file since the last time you pressed **(NEXT)** or a function key that provides a carriage return (all Viking 721 function keys provide carriage returns). With each succeeding press of F5 **UNDO** , the changes are undone in the reverse order you made them. F5 **UNDO** undoes changes you have made to the file text, and restores previous current positions.

For example, the following changes were made to a file in the order shown:

1. All abc's were changed to xyz's using the REPLACE_TEXT subcommand. **(NEXT)** was pressed.
2. The word **water** was changed to **juice** by typing **juice** over **water**.
3. The first line of the file was deleted using the **(HOME)** **(DELETE)** keys.

Each time you press F5 **UNDO** , the changes are undone as shown:

The first time
F5 **UNDO** is pressed.

The first line of the file is returned and the
word **juice** is changed back to **water**.

The second time

The xyz's are changed back to abc's.

NOTE

To undo changes to files edited earlier in your current job and not closed using QUIT or END, you must position the cursor in the file and then press F5 **UNDO**. FSE does not automatically return to a previously edited file to undo changes.

Stopping an Editing Session

There are two ways to stop an editing session. You can stop the editor and leave all the changes you have made to any edited files intact. You can also stop the editor and undo all the changes you made to the files you edited.

To stop the editor and save changes to edited files, press:

F6 **END**

The screen is cleared and after the editing changes are processed the system prompt appears:

/

To stop the editor and undo all of the changes made to all edited files, press

HOME

and enter the subcommand:

quit false

This cancels all changes you have made to all of your edited files. The screen is cleared and the system prompt appears:

/

For information on stopping an editing session when you are editing decks refer to Chapter 8, Editing SCU Decks.



Before you read the other chapters in part II, you should be familiar with a few of the concepts you will encounter. These concepts are:

- Subcommand Syntax 4-1
- Common Parameters 4-2
 - BOUNDARY (B) 4-2
 - COLUMN (COLUMNS or C) 4-3
 - LINE (LINES or L) 4-4
 - STATUS 4-4
- External and Working Files 4-5
- Open and Closed Files 4-6
- The VETO Parameter 4-7
- The Mask Character 4-9
- Editing Source Code Utility Decks 4-9



Subcommand Syntax

The editor subcommands have the same general syntax as SCL commands. That is, most subcommand names contain a verb describing the function they perform followed by an object specifying the target of the function. The verb and the object are separated by an underline character. For example:

```
delete_line
```

As you may have noted, you can compare a subcommand to a sentence describing the action. This can prove helpful if you can't quite recall the syntax of a subcommand; just try entering a sentence that describes what you want to do.

Also, subcommands can be abbreviated. All subcommands can be abbreviated by joining the first three letters of the verb with the first letter of the object. Several subcommands can be abbreviated to only the first letter of the verb and the first letter of the object. Still fewer subcommands can be abbreviated to a single letter. For example, the following subcommands can be abbreviated as shown here.

Full Name	Abbreviated Names
DELETE_WORD	DELW
INSERT_CHARACTER	INSC
INSERT_LINE	INSL or I
MARK_CHARACTER	MARC or MC

All valid abbreviations for all the subcommands are included in their descriptions and also on the inside back cover.

In the subcommand descriptions in part II, *italics* mean that the parameter is optional. Parameters that appear in **bold type** are required.

Common Parameters

Many subcommands use the same parameters. Several of these parameters have a large number of values. For this reason, the following common parameters and their values are also listed on the inside front cover.

BOUNDARY (B)

The BOUNDARY (B) parameter has the following values:

Value	Meaning
LINE (LINES or L)	The text boundaries of a subcommand are limited by lines.
STREAM (S)	The text boundaries of a subcommand are limited by characters specified by an accompanying COLUMNS parameter.

The following example boundaries may help clarify the descriptions.

LINE boundary:

```

Four score and
seven years ago
our forefathers brought
forth, on this continent,
a new nation.
  
```

← Beginning LINE boundary.

← Ending LINE boundary.

STREAM boundary:

```

Four score and
seven years ago
our forefathers brought
forth, on this continent,
a new nation.
  
```

Beginning STREAM boundary.

Ending STREAM boundary.

COLUMN (COLUMNS or C)

The COLUMN (COLUMNS or C) parameter has the following values:

Value	Meaning
integer	Any integer from 1 through 256.
CURRENT (C)	The current column.
FIRST_MARK (FM)	The first column of the marked text.
LAST_MARK (LM)	The last column of the marked text.
MARK (M)	All marked columns in the marked text.
MAXIMUM (MAX)	The highest possible number value for a column (equivalent to 256).

For COLUMN parameters that allow a range, you can specify two of the above values (except MARK) to form a range. For example, to specify a range of columns from column 1 to the current column, you could enter:

```
1..current
```

LINE (LINES or L)

The LINE (LINES or L) parameter has the following values:

Value	Meaning
integer	A line in the file. The integer can be from 1 through the number of lines in the file.
line identifier (modset.sequence)	A line in an SCU deck. The line identifier consists of a modification name and a sequence number, separated by a period.
ALL (A)	All lines of the file.
CURRENT (C)	The current line.
FIRST (F)	The first line of the file.
FIRST_MARK (FM)	The first line of the marked text.
FIRST_SCREEN (FS)	The top line of the file displayed on the screen.
LAST (L)	The last line of the file.
LAST_MARK (LM)	The last line of marked text.
LAST_SCREEN (LS)	The bottom line of the file displayed on the screen.
MARK (M)	All marked lines.
SCREEN (S)	All lines of that portion of the file displayed on the screen.

For LINE parameters that allow a range, you can specify two of the above values (except ALL, MARK, and SCREEN) to form a range. For example, to specify a range of lines from the first line of the screen to line 250, enter:

```
first_screen..250
```

STATUS

The STATUS parameter specifies the status variable to contain the completion status of the subcommand. All SCL commands and editor subcommands use this parameter. It is not included in the descriptions of the parameters for the subcommands in this manual. For information on the STATUS parameter and its values, refer to the SCL Language Definition manual.

External and Working Files

Part II uses the terms external files and working files. You need to know what these terms mean within the context of editing.

An external file is a file outside the editor. When the editor edits a file it uses a copy of an external file. This copy is called the working file.

When you stop your editing session, you have a choice whether to keep changes or discard them. The changes you make to the working file are not made to the external file until you enter an explicit `WRITE_FILE` command, or end your editing session with an `END` or `QUIT` subcommand or the F6 **END** key. If you make changes to a working file and end your editing session with a `QUIT FALSE` subcommand, the changes are not made to the external file.

Unless you specify how to position a file on a given subcommand, the open position of an external file is determined by the `SCL SET_FILE_ATTRIBUTES` command. If you have not entered a `SET_FILE_ATTRIBUTES` command with the `OPEN_POSITION` parameter specified, by default your external files except `OUTPUT` are positioned to the beginning-of-information. `OUTPUT` is positioned to the end-of-information.

A quick way to update the external copy of a working file without leaving the editor, is with the `WRITE_FILE` subcommand. (The format is described in the Copying section of chapter 5, Common Editing Functions.) To use the subcommand to update the external copy of the working file, enter

```
write_file
```

The working file then overwrites the external copy without stopping your editing session.

Open and Closed Files

Within part II, you will also encounter the terms open files and closed files. These terms have special meaning within the editor. Open files are files that have been accessed via the `EDIT_FILE` command. Most of the files you will be concerned about are open files. If a file is open, you can return to it during your editing session and undo changes. If you end your editing session with the `END` or `QUIT` subcommand, or the F6 **END** key, changes to all open files are made permanent. If you end your editing session with a `QUIT FALSE` subcommand, changes to all open files are canceled.

Closed files are files that have been explicitly closed using the `END_FILE` subcommand (this subcommand is described in chapter 5, Common Editing Functions). When a file is closed, any changes made permanent may not be undone.

The `END` and `QUIT` subcommands and the F6 **END** key are described further in the Stopping section of chapter 5, Common Editing Functions.

The VETO Parameter

The VETO parameter is available on subcommands such as REPLACE and LOCATE. The VETO parameter enables you to select which lines you want the subcommand to affect. This can be very helpful in screen mode if many lines are affected by, for example, a LOCATE_TEXT subcommand. In screen mode when you specify TRUE, the editor displays a directory of the located lines. From this display, you can select a line at which you want to be positioned. For example, you are editing file FRED and enter a LOCATE_TEXT subcommand that locates all the lines containing the string move. The following display appears:

```

Press Next/Return to continue, or a line number to stop and position.
Displaying Located Lines
-----
 6 This file explains how to move around the file or how to move text.
 8 To move around the file, use the arrow keys, the FWD and BKW keys, and s
10 To move text, use the FUNCTION key defined as MOVE or the MOVE text subc
12 To move text from one file to another, you can use a split screen. With
24 Once the files are on the screen you can move marked text from one file
34 to move the marked text.
47 moves the word cannonball
117 shows the moved text.

Unmk  Boxmk  LAST   endlin  Locnxt  80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO  F6 END  F7 LOCATE F8 locall

```

You can then enter the number of the line at which you want to be positioned or place the cursor on the line in the menu and press return to select that line position.

When you specify FALSE, the editor tells you how many occurrences were found and positions you at the last occurrence.

You can also use the VETO parameter in line mode. For example, if you enter a REPLACE_TEXT subcommand, you are prompted line by line to determine if you want the change for that line to occur. The following prompt appears:

REPLACE?

There are several valid responses to this prompt. These responses are:

Response	Meaning
CONTINUE (C)	The current line and any subsequent lines affected by the REPLACE subcommand are changed with no further interaction.
NO (N)	Skips the current line and goes on to the next.
QUIT (Q)	Stops the replacement.
YES (Y)	Replaces only the current line, and locates the next occurrence.

The Mask Character

Some commands described in part II search the file for a specified text string. With these subcommands, you can use what's called a mask character to serve as a wild card character; that is, it matches any other character. For example, to specify a string to be replaced, searched for, and so on, you could enter something like:

```
'F##d'
```

Strings that match this string would be:

```
Ford Fred Food Find Fund
```

or any other four-character string beginning with F and ending with d.

When you start editing, no mask character is set. You can set the mask character using the SET_MASK subcommand. The format of the subcommand is:

```
SET_MASK (SETM)
  CHARACTER=string or keyword value
  STATUS=status variable
```

The CHARACTER parameter specifies the mask character. Values can be any character or the keyword NONE. When NONE is specified, the mask feature is turned off. This parameter is required.

Editing Source Code Utility Decks

Any of the features and functions described in part II can be used to edit decks on SCU libraries. There are a few functions in chapter 9 that apply specifically to decks, and these are noted. Editing decks is described in detail in chapter 8.



This chapter describes the most common of the editing functions. Each section describes how to perform a function using function keys (when applicable) and subcommands.

Starting the Editor	5-1
Creating a File	5-4
Entering Subcommands	5-6
Getting Help	5-8
Stopping	5-9
Stopping with Function Keys	5-9
Stopping with Subcommands	5-9
Moving the Cursor	5-11
Moving the Cursor with Function Keys	5-11
Moving the Cursor with Subcommands	5-11
Moving the Cursor Using POSITION_CURSOR	5-12
Moving the Cursor Backward	5-14
Moving the Cursor Forward	5-14
Listing	5-14.1
Listing Lines Previous to the Current Line	5-14.1
Listing Lines Following the Current Line	5-14.2
Listing a Specific Line or Lines	5-14.3
Paging	5-14.4
Searching	5-15
Searching with Function Keys	5-15
Searching with Subcommands	5-15
Searching Using LOCATE_TEXT	5-16
Searching for All Occurrences of Text	5-18.1
Searching for the Next Occurrence of Text	5-18.2
Searching for the Next Occurrence of a String	5-18.2
Searching for Empty Lines	5-19
Searching for Wide Lines	5-20
Setting Search Margins	5-21
Inserting	5-22
Inserting with Function Keys	5-22
Inserting with Subcommands	5-22
Inserting Characters	5-23
Inserting Lines	5-24

Inserting Empty Lines	5-25
Inserting Words	5-26
Indenting Text	5-27
Inserting Files	5-28
Deleting	5-30
Deleting with Function Keys	5-30
Deleting with Subcommands	5-30
Deleting Characters	5-31
Deleting Lines	5-32
Deleting Blocks of Text	5-34
Deleting Words	5-36
Deleting Empty Lines	5-36
Deleting Characters from the Beginning of Lines	5-37
Replacing	5-38
Replacing Lines	5-38
Replacing Blocks of Text	5-40
Marking	5-42
Marking with Function Keys	5-42
Marking Lines with Subcommands	5-43
Marking a Box with Subcommands	5-43
Marking Characters with Subcommands	5-44
Unmarking with Subcommands	5-44
Saving Positions with Subcommands	5-45
Copying	5-46
Copying with Function Keys	5-46
Copying Blocks of Text with Subcommands	5-55
Copying Text Between Working Files and External Files	5-57
Moving	5-61
Moving with Function Keys	5-61
Moving with Subcommands	5-62
Undoing	5-65
Undoing with a Function Key	5-65
Undoing with Subcommands	5-65
Creating Multipartition Files	5-67
Text Formatting	5-68
Text Formatting with Function Keys	5-68
Text Formatting with Subcommands	5-68
Breaking Text	5-69
Joining Text	5-70
Setting Paragraph Margins	5-71
Formatting Paragraphs	5-72
Centering Lines	5-73

Starting the Editor

To start the editor, use the `EDIT_FILE` command. The format of the command is:

```
EDIT_FILE (EDIF)  
FILE=file  
INPUT=file  
OUTPUT=file  
PROLOG=file  
STATUS=status variable
```

The **FILE** (F) parameter specifies the name of the file you want to edit. If it does not already exist in your working catalog, it is created. The file must be a sequential file with a record type of CDC variable (V). Also, it can't be an object file. By default, files created by NOS/VE have these attributes. You can determine a file's record type by entering the `DISPLAY_FILE_ATTRIBUTE` command with the `DISPLAY_OPTION` parameter equal to `RECORD_TYPE`. If the file contains end-of-partition delimiters, the editor changes the delimiters to WEOP directives. This allows you to manipulate the end-of-partition delimiters during the editing session. Refer to *Creating Multipartition Files* later in this chapter for more information. This parameter is required.

The **INPUT** (I) parameter specifies the file used as input to the editor. This file can be positioned. This file contains optional editor subcommands used to manipulate the edit file. If **INPUT** is omitted, file `$COMMAND` is assumed. File `$COMMAND` is usually connected to the terminal.

The **OUTPUT** (O) parameter specifies the file to which you want to write any output that may result from your editing session. This file can be positioned. If **OUTPUT** is omitted, `$OUTPUT` is assumed.

The **PROLOG** (P) parameter specifies the name of the file containing subcommands you want executed each time you start the editor. Within this file you can put editor subcommands that you want executed every time you start the editor. For example, if you know you want to use screen mode on a DEC VT100 terminal, add the appropriate subcommand and you will automatically start the editor in screen mode. Chapter 10, *Prologue and Epilogue Files*, describes the prologue file in more detail. If **PROLOG** is omitted, `$USER.SCU_EDITOR_PROLOG` is assumed.

If you would like to specify a file containing FSE subcommands to be executed each time you leave the editor (an epilogue file), use the SET_EPILOG subcommand. Chapter 10, Prologue and Epilogue Files, describes setting an epilogue file in more detail.

For example, to start the editor and edit file ZAP, enter:

```
/edit_file file=$user.ZAP
```

The following prompt appears:

```
ef/
```

To get into screen mode, enter the ACTIVATE_SCREEN subcommand specifying the type of terminal you are using. Format of the subcommand is:

ACTIVATE_SCREEN (ACTS)

MODEL=name

STATUS=status variable

The MODEL (M) parameter specifies the type of terminal you are using. Valid entries are:

Entry	Terminal
CDC721	Control Data Viking 721
CDC722	Control Data 722
CDC 722_30	Control Data 722-30
VT100	DEC VT100
PC_CONNECT	IBM PC (or equivalent)
Z19	Zenith Z19 or Heathkit H19
Z29	Zenith Z29

If the MODEL parameter has not been specified on an earlier ACTIVATE_SCREEN or SET_SCREEN_OPTIONS subcommand or on the TERMINAL_MODEL parameter of the SET_TERMINAL_ATTRIBUTES command, it is required. To automatically set the terminal model, include the SET_TERMINAL_ATTRIBUTES command in your user prolog.

For example, to get into screen mode on a Viking 721 terminal, enter:

```
ef/activate_screen model=cdc721
```

The first screen of file ZAP then appears:

```

File: ZAP Lines 1 Thru 25 Size 61
-----
This is a text file.

It is supplied as a sample file.

Fozy score and seven

THIS LINE USES ALL CAPITAL LETTERS.

this line contains no capital letters.

This line is short.

This line is used to show the capabilities of the F4 key.

A file can contain all kinds of information.

Information like:

o Text

o Data

o Programs

o Anything else you can put in a file.
Unmrk   Boxmk   LAST   endlin   [ ]   [ ]   Locnxt   80/132
F1: MARK F2: Chrnk F3: FIRST F4: middle F5: UNDO F6: END F7: LOCATE F8: Locall

```

Once you are in screen mode, you can also enter the `ACTIVATE_SCREEN` subcommand to repaint the screen.

If you know you will be using the editor in screen mode on a particular terminal model, you can add the `ACTIVATE_SCREEN` subcommand to what's called a prologue file. This file contains subcommands or procedures that are executed every time you start the editor. For example, if you know you will be using the editor in screen mode on a DEC VT100, you would add the following `ACTIVATE_SCREEN` subcommand to your prologue file.

```
activate_screen model=vt100
```

Each time you start the editor with the `EDIT_FILE` subcommand, you are put in screen mode with the terminal set up as a VT100.

Creating a File

You can create a file in either screen or line mode.

Screen Mode:

To create a file in screen mode, enter the `EDIT_FILE` command specifying a file that does not exist. For example, file `ERNIE` does not exist on your working catalog. To create it on a Viking 721 terminal, enter:

```
/edit_file file=ernie
```

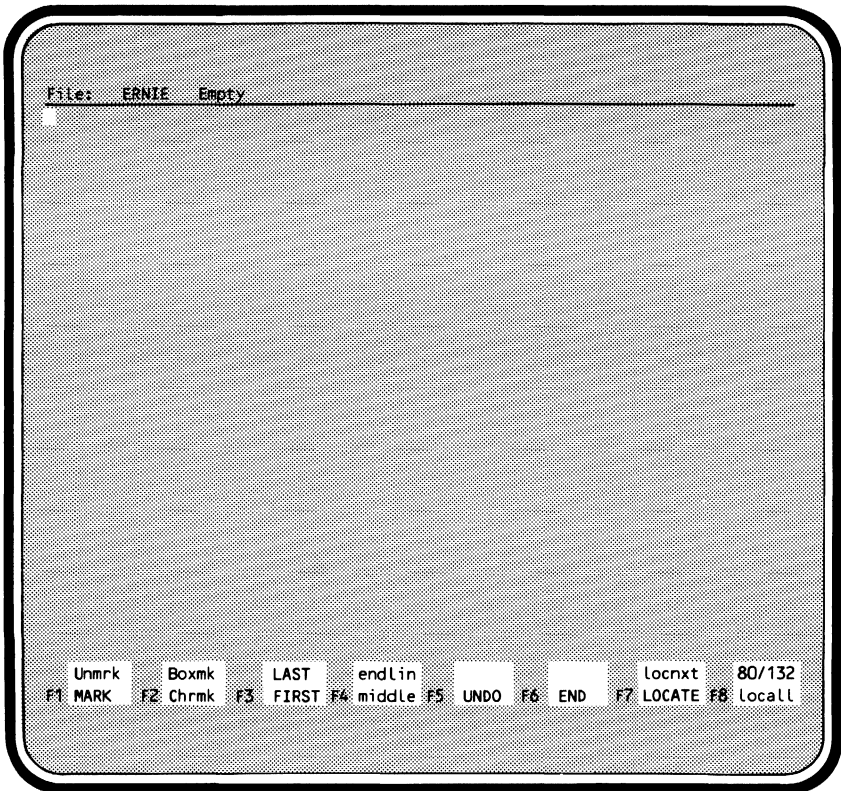
The following prompt appears:

```
ef/
```

Activate screen mode by entering:

```
ef/activate_screen model=cdc721
```

The following screen appears, showing you that it is an empty file:



To enter text into the file, just type what you want to appear in the file.

Line Mode:

To create a file in line mode, enter the same `EDIT_FILE` command and then use the `INSERT_LINES` subcommand to enter the file text. For example, to create file `ERNIE`, enter:

```
/edit_file file=ernie
```

You are prompted with:

```
ef/
```

To add text to file `ERNIE`, enter the `INSERT_LINES` subcommand:

```
ef/insert_lines
```

You are then prompted with:

```
?
```

Enter your text:

```
? Text to go into ERNIE.
```

Press:

```
(NEXT)
```

When you press `(NEXT)`, the following appears:

```
?
```

You then enter text you intend to appear on line 2 and press `(NEXT)`. If you don't enter any text and press `(NEXT)`, a blank line is inserted. Continue until you have added all the text you want to add. To stop the insert, enter what's known as the `UNTIL` character (or characters). This character (or characters) tells the editor to stop the insertion. The initial setting for the `UNTIL` character is `**`. For example, you are inserting text as shown in our previous example. To stop the insert, add `**` to the end of the last line to be inserted:

```
? That is all.**
```

Press:

```
(NEXT)
```

The insert is ended and the line mode prompt appears:

```
ef/
```

Refer to the `Inserting` section for more information on inserting text.

Entering Subcommands

You can enter subcommands in either screen or line mode.

Screen Mode:

To enter subcommands in screen mode, press:

(HOME)

This moves the cursor to the subcommand line (top line of the screen). With the cursor on this line, you can enter any subcommand. Besides editor subcommands, you can enter any SCL command. You cannot, however, continue a subcommand or command from the subcommand line to the next. The entire entry must be made on the subcommand line. If you enter a noneditor command that results in output written on your screen, the output is written over the the existing screen. To restore the screen to its original form, press:

(↑) (CLEAR) + (NEXT)

Another way to restore the screen to its original form is to use the REFRESH_ROW subcommand. Format of the subcommand is:

**REFRESH_ROW (REFR)
ROW=keyword value or list of range of integer**

The ROW (ROWS or R) parameter specifies the row of the screen to rewrite and is required. The range of rows can be 1 through the number of rows your terminal is capable of displaying, or the keyword value ALL. If you specify ALL, the entire screen is rewritten. If your terminal requires firmware to be offline loaded, use the ACTIVATE_SCREEN subcommand again or press:

(↑) (CLEAR) + (NEXT)

If there is an error in the syntax of the subcommand you enter or the subcommand causes an error, an error message is displayed on the line below which you entered the subcommand. The erroneous subcommand you tried to enter remains on the subcommand line. You can then type the correction over the subcommand and try again. If you want to abandon the entire entry, press **(NEXT)**, and the error message and the entry are erased.

Line Mode:

To enter subcommands in line mode, enter the subcommand after the line mode prompt and press **(NEXT)**:

```
ef/insert_lines
```

Press:

(NEXT)

In line mode, you can also enter any SCL command.

Getting Help

Online help is available through the editor in the form of a HELP file. The HELP file contains descriptions of all the editor subcommands and parameters.

To access the HELP file, you can use either the **(HELP)** key, (in screen mode only) or the HELP subcommand. To use the **(HELP)** key, just press it whenever you want help. The screen is split with the HELP file displayed on the bottom half of the screen and the text you are editing on the top half. If you already have more than one file displayed on the screen (this feature is described in chapter 7, Selecting Editor Options), the HELP file replaces the file at the bottom of the screen. You can then page through the file or use subcommands to find the text you want to read.

When you press **(HELP)**, only the first 10 or 12 lines of the HELP file are displayed. To find help quickly on a particular subcommand, use the HELP subcommand.

The HELP subcommand enables you to position the cursor at the text you want to read. The format of the subcommand is:

HELP (HEL)

TOPIC=keyword value

STATUS=status variable

The TOPIC (T) parameter specifies the editor subcommand on which you want help. When you specify this parameter, the cursor is positioned at the first line of text describing the specified subcommand. If TOPIC is omitted, the cursor is positioned at the first line of the file.

Stopping

To stop the editor, you can use function keys (screen mode only) or subcommands. With subcommands, you can specify whether or not you want the changes you've made to be permanent and you can also close just the current file.

If a file contains WEOP directives and has a record type of V (variable) then whenever the editor makes changes to a file permanent it also changes the WEOP directives to end-of-partition delimiters. Refer to Creating Multipartition Files later in this chapter for more information.

Stopping with Function Keys

The following function keys enable you to stop the editor.

Key	Function
F6 END	Stops the editor. Changes you have made to any open files are made permanent.
STOP	Deactivates screen mode. This key leaves you in the editor in line mode.

Stopping with Subcommands

There are several subcommands you can use to close files.

To stop the editor and close all edited files, you can use the **END** or **QUIT** subcommand. The subcommands do exactly the same thing and have the same format. The format is:

```
END or QUIT (QUI)
  WRITE_FILE=boolean
  STATUS=status variable
```

The **WRITE_FILE** (**WF**, **WRITE_DECK**, or **WD**) parameter specifies if you want changes to all open files made permanent. If **WRITE_FILE** is omitted, **TRUE** is assumed and the changes are made permanent. If **FALSE** is specified, no changes to open files are made permanent. For information on closing decks refer to Closing a Deck in chapter 8.

To close only the current file and continue editing other files, use the `END_FILE` subcommand. This closes the current file, making it impossible to undo any changes that have already been made, and frees the resources that were committed to it. It is also useful if you want to stay in the editor issuing commands that need to use the current file in its edited state. The format is:

```
END_FILE (ENDF)
  WRITE_FILE=boolean
  STATUS=status variable
```

The `WRITE_FILE` (`WF`, `WRITE_DECK`, or `WD`) parameter specifies if you want changes to this file made permanent. If `WRITE_FILE` is omitted, `TRUE` is assumed and all changes are made permanent in the current file. If `FALSE` is specified, changes are not made permanent.

The `END_FILE` and `END_DECK` subcommands perform the same function. Closing decks is described under `Closing a Deck` in chapter 8.

To stop screen mode without stopping the editor, use the `DEACTIVATE_SCREEN` subcommand. The format is:

```
DEACTIVATE_SCREEN (DEAS)
  STATUS=status variable
```

When you enter this subcommand, the screen is blanked and the line mode prompt appears:








```
ef/
```

Moving the Cursor

Several keys and subcommands are available to move the cursor around the file.

Moving the Cursor with Function Keys

The function keys available to move the cursor are:

Key	Function
	Moves the cursor up.
	Moves the cursor down.
	Moves the cursor right.
	Moves the cursor left.
	Moves the cursor forward to the next tab setting.
	Moves the cursor backward to the previous tab setting.
HOME	Moves the cursor to the subcommand line.
 F4 <code>endlin</code>	Moves the cursor to the end of the current line.

Moving the Cursor with Subcommands

You can also move the cursor to a specific line or backward or forward from the current position using subcommands. The general subcommand, `POSITION_CURSOR`, does the same thing with parameters that the other subcommands perform specifically. For example, you could use `POSITION_CURSOR LINE=24` to move the cursor to line 24 or you could use `LIST_LINE LINE=24`. You have many more options using the additional parameters of `POSITION_CURSOR`, but `LIST_LINE` is easier to remember.

Moving the Cursor Using POSITION_CURSOR

Using this subcommand in screen mode, you can move the cursor to a nontext line. The format of the subcommand is:

POSITION_CURSOR (POSC or P)

TEXT=range of string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

COLUMN=range of integer or keyword value

BOUNDARY=keyword value

DIRECTION=keyword value

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

ROW=integer

STATUS=status variable

The TEXT (T) parameter specifies a block of text at which to position the cursor. If TEXT is omitted, the new cursor position is determined by the LINE, COLUMNS, and BOUNDARY parameters.

The NUMBER (N) parameter specifies the number of times the search is to be repeated. Values can be a number or the keyword ALL (A). If NUMBER is omitted, ALL is assumed if you have specified a range for the LINE parameter; otherwise, a value of 1 is assumed.

The LINE (LINES or L) parameter can specify one of two things. If a single line number is specified (such as 100), the cursor is positioned at that line. If a range of lines is specified, the editor searches for the text string specified with the TEXT parameter within that range of lines. Values can be an integer or any of the LINE keyword values described in the Common Parameters section of chapter 4. If you specify a value of only one line, the search is limited to that line. If LINE and DIRECTION are omitted, CURRENT..LAST is assumed. If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed.

The **COLUMN** (**COLUMNS** or **C**) parameter specifies the range of columns to be searched to locate the specified text or word. Values can be an integer from 1 through 256, or any of the **COLUMN** keyword values described in the Common Parameters section of chapter 4. If **COLUMNS** is omitted, the editor does not supply a value. When you supply a value, the **BOUNDARY** parameter assumes a value of **STREAM**.

The **BOUNDARY** (**B**) parameter specifies the type of boundary that will limit the search. Values can be **LINE** or **STREAM**, as described in the Common Parameters section of chapter 4. If **BOUNDARY** is omitted, **LINE** is assumed. If a value for **COLUMNS** is specified and **BOUNDARY** is omitted, **STREAM** is assumed.

The **DIRECTION** (**D**) parameter specifies whether to search forward or backward from the current line. Values can be **FORWARD** (**F**) or **BACKWARD** (**B**). If you do not specify a value, **FORWARD** is assumed.

The **UPPER_CASE** (**UC**) parameter determines the significance of capitalization in a search. When the value is **TRUE**, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, **B** matches to both **B** and **b**. If the value is **FALSE**, the editor searches for the string exactly as you entered it. If you do not specify a value, **FALSE** is assumed.



The WORD (W) parameter, when the value is TRUE, instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the WORD parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The beginning and end of each line are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters #, \$, @, and _ are allowed as characters in words. When WORD is omitted, FALSE is assumed.

The REPEAT_SEARCH (RS) parameter instructs the editor to use the values used for the last TEXT, UPPER_CASE, and WORD parameters. TRUE instructs the editor to use the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them. In other words, if you specify TRUE for REPEAT_SEARCH and specify values for TEXT, UPPER_CASE, or WORD, the new values for TEXT, UPPER_CASE, and WORD are used. FALSE instructs the editor to use the parameters entered with the current POSITION_CURSOR subcommand. If REPEAT_SEARCH is omitted, FALSE is assumed.

The ROW (ROWS or R) parameter enables you to move the cursor in relation to the screen instead of in relation to the file text. When the value specified is a nontext row (like the subcommand line), the current position in the file remains the same. For terminals without a **(HOME)** key, using the ROW parameter enables you to simulate the **(HOME)** key.

The following examples show how you might use the POSITION_CURSOR subcommand.

```
position_cursor line=500
```

Positions the cursor at line 500 of the file.

```
position_cursor number=3 direction=b
```

Moves the current position backward three lines from the current line.

```
position_cursor lines=current..last number=2 column=1
```

Moves the cursor to the first column of the next line.

```
position_cursor row=2
```

Moves the cursor to the second line of the current screen.

```
position_cursor line=first
```

Moves the cursor to the first line in the file.

Moving the Cursor Backward

To position the cursor backward a specified number of lines, use the POSITION_BACKWARD subcommand. The format is:

POSITION_BACKWARD (POSB or PB)
NUMBER=integer

The NUMBER (N) parameter specifies the number of lines to move backward. If NUMBER is omitted, a value of 1 is assumed.

The following example shows how you might use the POSITION_BACKWARD subcommand.

```
position_backward n=25
```

Moves the cursor backward 25 lines from the current line.

Moving the Cursor Forward

To position the cursor forward a specified number of lines, use the POSITION_FORWARD subcommand. The format is:

POSITION_FORWARD (POSF or PF)
NUMBER=integer

The NUMBER (N) parameter specifies the number of lines to move forward. If NUMBER is omitted, a value of 1 is assumed.

The following example shows how you might use the POSITION_FORWARD subcommand.

```
position_forward n=63
```

Moves the cursor forward 63 lines from the current line.

Listing

To list lines on the editor, there are three subcommands available. Lines previous to the current cursor position, following the current position, or selected by line number from anywhere in the file may be listed.

Listing Lines Previous to the Current Line

While in line mode, to display a range of lines ending with the current line, use the `LIST_BACKWARD` subcommand. In effect, it enables you to view a number of lines just before the current line and end up where you started. The format is:

LIST_BACKWARD (LISB or LB)
NUMBER=integer or keyword value

The `NUMBER (N)` parameters specifies the number of lines to list. Values can be numbers or the keyword `ALL`. `ALL` lists all the lines from the beginning of the file to the current position. If `NUMBER` is omitted, a value of 1 is assumed.

The following example shows how you might use the `LIST_BACKWARD` subcommand.

```
list_backward n=15
```

Lists 15 lines ending with the current line.

Listing Lines Following the Current Line

While in line mode, to display a range of lines beginning with the current line, use the LIST_FORWARD subcommand. The format is:

```
LIST_FORWARD (LISF or LF)  
NUMBER=integer or keyword value
```

The NUMBER (N) parameter specifies the number of lines to list. Values can be numbers or the keyword ALL. ALL lists all the lines from the current position to the end of the file. If NUMBER is omitted, a value of 1 is assumed.

The following example shows how you might use the LIST_FORWARD subcommand.

```
list_forward n=15
```

Lists 15 lines beginning with the current line.

Listing a Specific Line or Lines

To list a specified line or range of lines, in line mode, use the `LIST_LINES` subcommand. In screen mode, use the `LIST_LINES` subcommand to position the cursor at the specified line. The format is:

LIST_LINES (LISL or LL)
LINE=integer or keyword value

The `LINE` (`LINES` or `L`) parameter specifies the line or range of lines to list. If a range of lines is specified while in screen mode, the cursor is positioned at the last line in the range. If `LINE` is omitted, `CURRENT` is assumed.

The following example shows how you might use the `LIST_LINES` subcommand.

```
list_lines 1=25..40
```

Lists lines 25 through 40.

Paging

Paging applies to screen mode only. Paging is moving through the file a screen at a time. To view the next screen of text, press:

(FWD)

The bottom line of the current screen appears at the top of the next screen.

To view the previous screen of text, press:

(BKW)

The top line of the current screen appears at the bottom of the previous screen.

Another way to move more than a few lines at a time is with the **(UP)** and **(DOWN)** keys. When you press

(UP)

the file is repositioned so that the line containing the cursor is at the top of the screen.

When you press

(DOWN)

the file is positioned so that the line containing the cursor is at the bottom of the screen.

Pressing either key repeatedly moves you through the file a half page at a time.


The position of the cursor after pressing **(FWD)**, **(BKW)**, **(UP)**, or **(DOWN)** is the middle of the screen (not the top or bottom of the screen).

Searching

There are several function keys and subcommands you can use to search for text strings within a file.

Searching with Function Keys

The function keys used for searching are:

Key	Function
F7 LOCATE	Prompts you for a string to find and then locates it.
 F7 locnxt	Searches for the next occurrence of a previously specified string.
F8 locall	Prompts you for a string to find, locates all lines containing the string, and displays them in a directory-type display. To locate all occurrences of the last specified string, press (NEXT) when prompted to enter a search string.

Searching with Subcommands

You can also locate text, empty lines, and wide lines using subcommands. The general subcommand `LOCATE_TEXT` does the same thing using parameters that the other subcommands perform specifically. For example, you could use `LOCATE_TEXT TEXT='start' NUMBER=ALL` to find all occurrences of `start`, or you could use `LOCATE_ALL TEXT='start'`.

There are specific subcommands to locate all occurrences of text, locate the next occurrence of text, locate the next occurrence of a string, and limit the columns in which you want the search to occur.

Searching Using LOCATE_TEXT

To locate blocks of text, use the LOCATE_TEXT subcommand. The format is:

LOCATE_TEXT (**LOCT** or **L**)
TEXT=range of string
NUMBER=integer or keyword value
LINE=range of integer or keyword value
COLUMN=range of integer or keyword value
BOUNDARY=keyword value
DIRECTION=keyword value
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
VETO=boolean
STATUS=status variable

The TEXT (T) parameter specifies strings of text in the first and last lines of a block of text to be located. If you enter only one string, the block of text to be located will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found and the cursor is positioned at the beginning of the first string. If TEXT is omitted, the lines to be located will be determined by the NUMBER, LINE, and DIRECTION parameters.

The NUMBER (N) parameter specifies the number of blocks of text to be found. Values for this parameter can be a number or the keyword ALL (A). If you specify a range of values for the LINE parameter, the NUMBER parameter assumes a value of ALL; otherwise, a value of 1 is assumed. In line mode, the NUMBER parameter is used to display a range of lines. Refer to Printing Lines in Line Mode later in this chapter for more information.

The LINE (L) parameter specifies a range of lines to be searched. Values can be an integer or any of the LINE keyword values described in the Common Parameters section of chapter 4. If you specify a value of only one line, the search is limited to that line. If LINE and DIRECTION are omitted, CURRENT..LAST is assumed. If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed. In line mode, the LINE parameter can specify the lines to print. Refer to Printing Lines in Line Mode later in this chapter for more information.

The COLUMN (COLUMNS or C) parameter specifies the range of columns to search. Values can be an integer from 1 through 256, or any of the COLUMN keyword values described in the Common Parameters section of chapter 4. If COLUMNS is omitted, CURRENT is assumed.

The **BOUNDARY (B)** parameter specifies the type of boundary that will limit the search. Values can be **LINE** or **STREAM** as described in the Common Parameters section of chapter 4. If **BOUNDARY** and **COLUMNS** are omitted, **LINE** is assumed. If a value for **COLUMNS** is specified and **BOUNDARY** is omitted, **STREAM** is assumed.

The **DIRECTION (D)** parameter specifies whether to search forward or backward from the current line. Values can be **FORWARD (F)** or **BACKWARD (B)**. If you do not specify a value, **FORWARD** is assumed.

The **UPPER_CASE (UC)** parameter determines the significance of capitalization in a search. When the value is **TRUE**, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, **B** matches to both **B** and **b**. If the value is **FALSE**, the editor searches for the string exactly as you entered it. If you do not specify a value, **FALSE** is assumed.

The **WORD (W)** parameter, when the value is **TRUE**, instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the **WORD** parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters **@**, **#**, **\$**, and **_** are allowed as characters in words. When **WORD** is omitted, **FALSE** is assumed.

The **REPEAT_SEARCH (RS)** parameter instructs the editor to use the values used for the last **TEXT**, **UPPER_CASE**, and **WORD** parameters. **TRUE** instructs the editor to use the same **TEXT**, **UPPER_CASE**, and **WORD** parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify **TRUE** for **REPEAT_SEARCH** and specify values for **TEXT**, **UPPER_CASE**, or **WORD**, the new values for **TEXT**, **UPPER_CASE**, and **WORD** are used. **FALSE** instructs the editor to use the parameters entered with the current **LOCATE_TEXT** subcommand. If **REPEAT_SEARCH** is omitted, **FALSE** is assumed.

The **VETO (V)** parameter instructs the editor to turn on or off the **VETO** parameter described in chapter 4. When you specify **TRUE**, the editor displays a directory of located lines. If **VETO** is omitted, **FALSE** is assumed.

The following examples show how you might use the `LOCATE_TEXT` subcommand.

```
locate_text 'PROCEND'
```

Locates the next occurrence of `PROCEND`.

```
loct 'TITLE' direction=b
```

Locates the previous occurrence of `TITLE`.

```
loct line=250
```

Positions the cursor on line 250 of the current file or deck.

```
loct repeat_search=true
```

Locates the string you last specified as a value for the `TEXT` parameter.

```
loct 'PARAMETER' number=all veto=true
```

Locates all occurrences of `PARAMETER` from the current position to the end of the file and displays the lines in a directory-type display.

```
loct 'one'..'twenty'
```

Locates the next block of text beginning with `one` and ending with `twenty`.

```
l n=5
```

In line mode, prints the current line and four subsequent lines. In screen mode, positions the cursor four lines forward.

Searching for All Occurrences of Text

To search the entire file to locate all occurrences of a specified string, use the `LOCATE_ALL` subcommand. In screen mode, all occurrences are then listed, enabling you to position the cursor at a specific line or enter the line number desired. In line mode, all occurrences are listed and you are positioned at the last occurrence of the string. The format is:

LOCATE_ALL (LOCA or LA)
TEXT=string

The `TEXT (T)` parameter specifies the text string you want to find. If `TEXT` is omitted, the last text string specified, if any, is assumed.

The following example shows how you might use the `LOCATE_ALL` subcommand.

```
locate_all text='find this text'
```

Locates all occurrences of the string, find this text, in the file and lists them.

Searching for the Next Occurrence of Text

To locate the next occurrence of a previously specified string, use the `LOCATE_NEXT` subcommand. The search begins one column after the current column. The format is:

`LOCATE_NEXT (LOCN or LN)`

Searching for the Next Occurrence of a String

To search for the next line containing a specified string, beginning at the current line and column, use the `LOCATE_STRING` subcommand. The format is:

`LOCATE_STRING (LOCS or LS)`
TEXT=string

The `TEXT (T)` parameter specifies the text string to locate. If `TEXT` is omitted, the last string parameter specified, if any, is used.

Searching for Empty Lines

To find empty lines (lines with no characters), use the `LOCATE_EMPTY_LINES` subcommand. The subcommand format is:

```
LOCATE_EMPTY_LINES (LOCEL)
  NUMBER=integer or keyword value
  LINE=range of integer or keyword value
  DIRECTION=keyword value
  VETO=boolean
  STATUS=status variable
```

The `NUMBER (N)` parameter specifies the number of empty lines to find. Values for this parameter can be numbers or the keyword `ALL`. If you specify a `LINE` parameter, this parameter assumes a value of `ALL`; otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines to search. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If you specify a value of only one line, the search is limited to that line. If `LINE` and `DIRECTION` are omitted, `CURRENT..LAST` is assumed. If you omit `LINE` and specify `BACKWARD` for the `DIRECTION` parameter, `CURRENT..FIRST` is assumed.

The `DIRECTION (D)` parameter specifies whether to search forward or backward from the current line. Values can be `FORWARD (F)` or `BACKWARD (B)`. If you do not specify a value for `DIRECTION`, `FORWARD` is assumed.

The `VETO (V)` parameter instructs the editor to turn on or off the `VETO` parameter described in chapter 4. Values can be `TRUE` or `FALSE`. When you specify `TRUE`, the editor displays a directory of located lines. If `VETO` is omitted, `FALSE` is assumed.

The following examples show how you might want to use the `LOCATE_EMPTY_LINES` subcommand.

```
locate_empty_lines number=5
```

Locates the next five empty lines.

```
locel line=20..40
```

Locates all empty lines between lines 20 and 40.

```
locel number=10 line=mark
```

Locates the next 10 empty lines in the marked text.

Searching for Wide Lines

To locate lines that are wider than the margins specified by the `SET_LINE_WIDTH` subcommand, use the `LOCATE_WIDE_LINE` subcommand. Format of the subcommand is:

```
LOCATE_WIDE_LINE (LOCATE_WIDE_LINES or LOCWL)
  NUMBER=integer or keyword value
  LINE=range of integer or keyword value
  DIRECTION=keyword value
  VETO=boolean
  STATUS=status variable
```

The `NUMBER (N)` parameter specifies the number of wide lines to be found. Values for this parameter can be numbers or the keyword `ALL (A)`. If you specify a `LINES` parameter, this parameter assumes a value of `ALL`; otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines to be searched. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If you specify a value of only one line, the search is limited to that line. If `LINE` and `DIRECTION` are omitted, `CURRENT..LAST` is assumed. If you omit `LINE` and specify `BACKWARD` for the `DIRECTION` parameter, `CURRENT..FIRST` is assumed.

The `DIRECTION (D)` parameter specifies whether to search forward or backward from the current line. Values can be `FORWARD (F)` or `BACKWARD (B)`. If `DIRECTION` is omitted, `FORWARD` is assumed.

The `VETO (V)` parameter instructs the editor to turn on or off the `VETO` parameter described in chapter 4. Values can be `TRUE` or `FALSE`. When you specify `TRUE`, the editor displays a directory of located lines. If `VETO` is omitted, `FALSE` is assumed.

The following examples show how you might use the `LOCATE_WIDE_LINES` subcommand.

```
locate_wide_lines number=10 veto=true
```

Locates and displays a directory of the next 10 wide lines.

```
locwl line=first_screen..last veto=true
```

Locates and displays a directory of all wide lines between the top line of the current screen and the last line of the file.

```
locwl number=10 line=mark veto=true
```

Locates and displays a directory of the next 10 wide lines in the marked text.

Setting Search Margins

If you need to limit the number of columns to be searched in subsequent subcommands that use string searches, use the `SET_SEARCH_MARGINS` subcommand. Format is:

`SET_SEARCH_MARGINS (SETSM)`

MARGIN_COLUMN=range of integer or keyword value
STATUS=status variable

The `MARGIN_COLUMN` (`MARGIN_COLUMNS` or `MC`) parameter specifies the columns in which to perform the search. Values can be any number or any keyword from the list of values for the `COLUMN` parameter in the Common Parameters section of chapter 4. If you specify two values, the search is done from the first column through the last column specified. If you specify a single integer, only that column is searched. If `MARGIN_COLUMN` is omitted, columns 1 through 256 are assumed.

For example, to set the search margins to columns 1 and 7, enter:

```
set_search_margin margin_column=1..7
```

The same subcommand could be entered as:

```
setsm mc=1..7
```

Inserting

There are several function keys and subcommands which allow you to insert text into your file. Described in this section are the function keys and subcommands that perform insert functions. Functions include inserting lines, characters, words, and blocks of blank lines.

Inserting with Function Keys

The following keys perform the described insert functions.

Key	Function
INSRT	Inserts a blank character over which you can type the character to be inserted.
↵ INSRT	Inserts a blank line over which you can type the new text.
F9 inseL	Inserts blank lines over which you can type new text. The number of blank lines inserted depends on the terminal and the current number of files displayed on the screen. Two lines of text are left at the top and bottom of the screen with the inserted empty lines in between.
F10 inswrđ	Inserts 32 blank characters over which you can type a new word or phrase. If you don't need this many blank characters, use the ↵ F10 delwrđ keys to delete any extra blanks.
F15 INDENT	Inserts two blank characters in front of any marked text. If no text is marked, two blank characters are inserted before the current line.

Inserting with Subcommands

There are several subcommands that enable you to insert characters, lines, words, text, and text from other files.

Inserting Characters

To insert characters, use the `INSERT_CHARACTER` subcommand. This subcommand inserts a string of characters before a specified location in the file. Format of the subcommand is:

```
INSERT_CHARACTER (INSERT_CHARACTERS, INSC, or IC)
  NEW_TEXT=string
  INSERTION_LOCATION=integer or keyword value
  INSERTION_COLUMN=integer or keyword value
  STATUS=status variable
```

The `NEW_TEXT` (`NT`) parameter specifies the text to be inserted. If `NEW_TEXT` is omitted, one space character is assumed.

The `INSERTION_LOCATION` (`IL`) parameter specifies the line in which the text is to be inserted. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `INSERTION_COLUMN` (`INSERTION_COLUMNS` or `IC`) parameter specifies the column before which you want the insertion to begin. Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `INSERTION_COLUMN` is omitted, `CURRENT` is assumed.

The following are examples of how you might use the `INSERT_CHARACTER` subcommand.

```
insert_characters 'Short comment'
```

Inserts the text (Short comment) in front of the current column on the current line.

```
insc new_text='Last line' insertion_location=last column=7
```

Inserts the text (Last line) before column 7 on the last line of the file.

Inserting Lines

To insert lines, use the `INSERT_LINES` subcommand. The format of this subcommand is:

```
INSERT_LINES (INSL or I)
  NEW_TEXT=string
  PLACEMENT=keyword value
  INSERTION_LOCATION=integer or keyword value
  UNTIL=string
  STATUS=status variable
```

The `NEW_TEXT` (NT) parameter specifies the new line of text to be inserted. If `NEW_TEXT` is omitted, the text to be inserted is taken from the command input file.

The `PLACEMENT` (P) parameter indicates if you want the insertion to occur before or after the location specified by the `LOCATION` parameter. Values can be `BEFORE` (B) or `AFTER` (A). If `PLACEMENT` is omitted, `AFTER` is assumed. The function key inserts `BEFORE`.

The `INSERTION_LOCATION` (IL) parameter specifies the line after which or before which the insertion is to occur. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `UNTIL` (U) parameter specifies a character that stops the insert. If the `NEW_TEXT` parameter is omitted, you are prompted to enter input until the editor encounters the character(s) you specify with this parameter as the last character(s) in a line. If `UNTIL` is omitted, `**` is assumed.

The following are examples of how you might use the `INSERT_LINES` subcommand.

```
insert_lines 'NEW LINE'
```

Inserts the text (NEW LINE) after the current line.

```
i new_text='Insert' position=before
```

Inserts the text (Insert) before the current line.

```
inl nt='First line' insertion_location=first position=before
```

Inserts the text (First line) before the first line of the file.

```
insert_lines insertion_location=45 position=before until='#'
```

Inserts lines from the command input file before line 45 until a # character is encountered as the last character in a line.

Inserting Empty Lines

To insert empty lines, use the `INSERT_EMPTY_LINES` subcommand. The format of the subcommand is:

INSERT_EMPTY_LINES (INSEL)

NUMBER=integer

INSERTION_LOCATION=integer or keyword value

PLACEMENT=keyword value

STATUS=status variable

The `NUMBER (N)` parameter specifies the number of empty lines to insert. Values can be any integer from 1 through 262,143. If `NUMBER` is omitted, a value of 1 is assumed.

The `INSERTION_LOCATION (IL)` parameter specifies the line at which the insertion is to occur. Values can be an integer or any of the `LINE` keyword values described in the `Common Parameters` section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `PLACEMENT (P)` parameter specifies whether the insertion is to occur after or before the specified line or lines. Values can be `BEFORE (B)` or `AFTER (A)`. If `PLACEMENT` is omitted, `AFTER` is assumed.

The following examples show how you might use the `INSERT_EMPTY_LINES` subcommand:

```
insel number=20 insertion_location=30 placement=before
```

Inserts 20 empty lines before line 30.

```
insel number=2 insertion_location=50
```

Inserts two empty lines after line 50.

Inserting Words

To insert words, use the `INSERT_WORD` subcommand. This subcommand inserts a string or 32 blank characters before a specified column in a line. Format of the subcommand is:

```
INSERT_WORD (INSW)
  NEW_TEXT=string
  INSERTION_LOCATION=integer or keyword value
  INSERTION_COLUMN=integer or keyword value
  STATUS=status variable
```

The `NEW_TEXT (NT)` parameter specifies the string to be inserted. If `NEW_TEXT` is omitted, 32 space characters are assumed.

The `INSERTION_LOCATION (IL)` parameter specifies the line in which the word is to be inserted. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `INSERTION_COLUMN (INSERTION_COLUMNS or IC)` parameter specifies the column before which the insertion is to occur. Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `INSERTION_COLUMN` is omitted, `CURRENT` is assumed.

The following examples show how you might use the `INSERT_WORD` subcommand:

```
insert_word
```

Inserts 32 spaces in front of the current column of the current line.

```
insw new_text='LINE' insertion_location=10 insertion_column=1
```

Inserts the word `LINE` in front of line 10.

Indenting Text

There is also a subcommand designed specifically to insert blank characters in front of lines of text; the `INDENT_TEXT` subcommand. Format of the subcommand is:

```
INDENT_TEXT (INDT)
  OFFSET=integer
  NUMBER=integer or keyword value
  LINE=range of integer or keyword value
  STATUS=status variable
```

The `OFFSET (O)` parameter specifies the number of columns to indent the specified block of text. If you specify a negative value, that number of characters will be deleted from the beginning of the lines in the block of text. If `OFFSET` is omitted, a value of 1 is assumed.

The `NUMBER (N)` parameter specifies the number of lines to be indented. Values may be an integer or any of the `LINE` keyword values specified in the Common Parameters section of chapter 4. If you specify a range for the `LINE` parameter, the `NUMBER` parameter assumes a value of `ALL`. Otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines to be indented. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single line is specified, only that line is indented. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The following examples show how you might use the `INDENT_TEXT` subcommand.

```
indent_text offset=5 line=a
```

Indents all lines five spaces.

```
indent_text line=mark
```

Indents all marked lines one space.

```
indt
```

Indents the current line one space.

```
indent_text offset=-7 line=25..L
```

Deletes the first 7 characters from lines 25 through the last line.

Inserting Files

To insert the text of another file into the current file, use the `READ_FILE` command. Format of the subcommand is:

READ_FILE (REAF)

FILE=file

INSERTION_LOCATION=integer or keyword value

PLACEMENT=keyword value

MULTI_PARTITION=boolean

STATUS=status variable

The **FILE (F)** parameter specifies the name of the file from which the text is to be inserted. The entire file will be inserted. This parameter is required.

The **INSERTION_LOCATION (IL)** parameter specifies the line at which the insertion is to occur. Values can be an integer or any of the **LINE** keyword values described in the Common Parameters section of chapter 4 with the exception of **ALL**, **MARK**, and **SCREEN**. Ranges are not allowed. If **INSERTION_LOCATION** is omitted, **CURRENT** is assumed.

The **PLACEMENT (P)** parameter specifies if you want the insertion to occur before or after the line specified by the **INSERTION_LOCATION** parameter. Values may be **AFTER (A)** or **BEFORE (B)**. If **PLACEMENT** is omitted, **AFTER** is assumed.

The **MULTI_PARTITION (MP)** parameter specifies whether the editor is to change the end-of-partition delimiters in the file to **WEOP** directives. When the value is **TRUE**, the editor changes the delimiters to **WEOP** directives. When the value is **FALSE**, the editor stops reading the file at the first end-of-partition boundary it encounters. If **MULTI_PARTITION** is omitted, **FALSE** is assumed. Refer to **Creating Multipartition Files** later in this chapter for more information.

The following examples show how you might use the `READ_FILE` command.

```
read_file file=ernie insertion_location=320
```

Inserts the contents of file `ERNIE` into the current file immediately after line 320.

```
read_file file=bert insertion_location=last_mark position=before
```

Inserts the contents of file `BERT` into the current file immediately before the last marked line.

NOTE

The `READ_FILE` subcommand reads the external copy of the specified file. If you have been editing a file within the editor and have not made the changes permanent and then specify that file on a `READ_FILE` subcommand, an external copy is inserted, not the changed working copy.



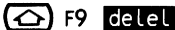
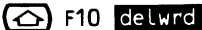
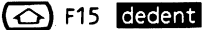
For more information on copying and moving parts of files, refer to the sections on Copying and Moving later in this chapter.

Deleting

There are several function keys and subcommands which enable you to delete such things as characters, lines, and blank lines from your file. The function keys and subcommands are described in this section.

Deleting with Function Keys

The following function keys delete text:

Key(s)	Function
 DLETE	Deletes the current character.
 DLETE	Deletes the current line.
 F9 delel	Deletes all empty lines, starting with the current line, until a nonempty line is encountered.
 F10 delwrd	Deletes the current word. If the cursor is on an alphanumeric character or a #, \$, @, or _ that character and any surrounding characters are deleted until a nonalphanumeric character (or blank character) is encountered. If the cursor is on a blank character, all blanks are deleted until a nonblank character is encountered.
 F15 dedent	Deletes the first two characters in each line of all marked text.

Deleting with Subcommands

There are several subcommands that delete characters, lines, words, and text.

Deleting Characters

To delete characters, use the `DELETE_CHARACTERS` subcommand. Format of the subcommand is:

DELETE_CHARACTERS (DELC or DC)

NUMBER=integer or keyword value

LINE=integer or keyword value

COLUMN=integer or keyword value

STATUS=status variable

The `NUMBER (N)` parameter specifies the number of characters to be deleted. Values may be an integer or the keyword `ALL`. If you omit the `NUMBER` parameter and specify a range for either the `LINE` or `COLUMN` parameters, `ALL` is assumed. Otherwise, if `NUMBER` is omitted, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a line in which characters will be deleted. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `LINE` is omitted, `CURRENT` is assumed.

The `COLUMN (COLUMNS or C)` parameter specifies the columns of the specified line(s) to be deleted. Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `COLUMN` is omitted, `CURRENT` is assumed.

Deleting Lines

To delete lines, use the `DELETE_LINES` subcommand. This subcommand enables you to delete single lines or a range of lines. Format of the subcommand is:

DELETE_LINES (DELL)

TEXT=range of string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

STATUS=status variable

The `TEXT (T)` parameter specifies a block of text to be deleted, beginning with the line containing the first string to the the line containing the second string. If `TEXT` is omitted, a value is not supplied by the editor and the lines to be deleted are determined by the `NUMBER` and `LINE` parameters.

The `NUMBER (N)` parameter specifies the number of lines to be deleted from the current line forward. Values may be numbers or the keyword `ALL`. If you omit the `NUMBER` parameter and specify a range of lines for the `LINE` parameter, this parameter assumes a value of `ALL`. Otherwise, a value of 1 is assumed. If you specify a range for the `TEXT` parameter, the `NUMBER` parameter specifies the number of text blocks to delete.

The `LINE (LINES or L)` parameter specifies a range of lines to be deleted. Values may be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If you specify a single integer or keyword value, only that line is deleted. If you specify `LINE=MARK`, marked lines are deleted in their entirety, even if the boundary implied by the mark is `STREAM`. If `LINE` is omitted, `ALL` is assumed.

The `UPPER_CASE (UC)` parameter determines the significance of capitalization in a search. When the value is `TRUE`, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, `B` matches to both `B` and `b`. If the value is `FALSE`, the editor searches for the string exactly as you entered it. If you do not specify a value, `FALSE` is assumed.

The **WORD (W)** parameter specifies whether to search for the specified text string as a word. When the value is **TRUE**, it instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the **WORD** parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters **@**, **#**, **\$**, and **_** are allowed as characters in words. When **WORD** is omitted, **FALSE** is assumed.

The **REPEAT_SEARCH (RS)** parameter instructs the editor to use the values used for the last **TEXT**, **UPPER_CASE**, and **WORD** parameters. **TRUE** instructs the editor to use the same **TEXT**, **UPPER_CASE**, and **WORD** parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify **TRUE** for **REPEAT_SEARCH** and specify values for **TEXT**, **UPPER_CASE**, or **WORD**, the new values for **TEXT**, **UPPER_CASE**, and **WORD** are used. **FALSE** instructs the editor to use the parameters entered with the current **DELETE_LINES** subcommand. If **REPEAT_SEARCH** is omitted, **FALSE** is assumed.

Deleting Blocks of Text

To delete blocks of text, use the `DELETE_TEXT` subcommand. Format of the subcommand is:

DELETE_TEXT (DELT or D)

TEXT=range of string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

COLUMN=range of integer or keyword value

BOUNDARY=keyword value

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

STATUS=status variable

The `TEXT (T)` parameter specifies the text strings which form the boundaries of the block of text to be deleted. If a single text string is specified, the block to be deleted will contain one line. If two text strings are specified, the search for the second string begins after the first string is found. When the second string is found, the text between the two strings (including the strings) is deleted. When you specify `TEXT`, the `BOUNDARY` parameter is `LINE`. If `TEXT` is omitted, the `LINE` and `NUMBER` parameters determine the lines to be deleted.

The `NUMBER (N)` parameter specifies the number of blocks to be deleted. Values may be an integer or the keyword `ALL (A)`. If `NUMBER` is omitted, and a range of values is specified for the `LINE` parameter, `ALL` is assumed. Otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines to be deleted. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single integer is specified, only that line is deleted. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The `COLUMN (COLUMNS or C)` parameter specifies the columns to be deleted in the specified lines. Values can be numbers from 1 through 256 or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4. If `COLUMN` is omitted, the `DELETE_TEXT` subcommand deletes complete lines.

The `BOUNDARY (B)` parameter specifies the type of boundary that will limit the search. Values can be `LINE` or `STREAM` as described in the Common Parameters section of chapter 4. If `BOUNDARY` is omitted, `LINE` is assumed. If a value for `COLUMN` is specified, `STREAM` is assumed.

The `UPPER_CASE` (`UC`) parameter determines the significance of capitalization in a search. When the value is `TRUE`, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, `B` matches to both `B` and `b`. If the value is `FALSE`, the editor searches for the string exactly as you entered it. If you do not specify a value, `FALSE` is assumed.

The `WORD` (`W`) parameter specifies whether to search for the specified text as a word. When the value is `TRUE`, it instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the `WORD` parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters `@`, `#`, `$`, and `_` are allowed as characters in words. When `WORD` is omitted, `FALSE` is assumed.

The `REPEAT_SEARCH` (`RS`) parameter instructs the editor to use the values used for the last `TEXT`, `UPPER_CASE`, and `WORD` parameters. `TRUE` instructs the editor to use the same `TEXT`, `UPPER_CASE`, and `WORD` parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify `TRUE` for `REPEAT_SEARCH` and specify values for `TEXT`, `UPPER_CASE`, or `WORD`, the new values for `TEXT`, `UPPER_CASE`, and `WORD` are used. `FALSE` instructs the editor to use the parameters entered with the current `DELETE_TEXT` subcommand. If `REPEAT_SEARCH` is omitted, `FALSE` is assumed.

The following are examples of how you might use the `DELETE_TEXT` subcommand.

```
d
```

Deletes the current line.

```
d text='first'..'last'
```

Deletes all lines from the line containing `first` to the line containing `last`.

Deleting Words

To delete words, use the `DELETE_WORDS` subcommand. What is deleted depends on where the cursor is positioned. If positioned on an alphanumeric character, the current word is deleted. If positioned on any other character, that single character is deleted. If the cursor is positioned on a blank character, the current character and any following blanks are deleted. Subcommand format is:

```
DELETE_WORD (DELW or DW)
  LINE=integer or keyword value
  COLUMN=integer or keyword value
  STATUS=status variable
```

The `LINE` (`LINES` or `L`) parameter specifies a line in which the deletion is to occur. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `LINE` is omitted, `CURRENT` is assumed.

The `COLUMN` (`COLUMNS` or `C`) parameter specifies the column to begin the deletion. Values can be an integer or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `COLUMN` is omitted, `CURRENT` is assumed.

The following are examples showing how you might use the `DELETE_WORD` subcommand.

```
delete_word line=50
```

Deletes the first word in line 50 (assuming the current column is 1).

```
delete_word line=200 column=7
```

Deletes the word that has a character in column 7 of line 200.

Deleting Empty Lines

To delete a block of blank lines, use the `DELETE_EMPTY_LINES` subcommand. This deletes lines until a nonblank line is encountered. Format is:

```
DELETE_EMPTY_LINES (DELEL)
  LINE=range of integer or keyword value
  STATUS=status variable
```

The `LINE` (`LINES` or `L`) parameter specifies the line at which the deletion of blank lines is to begin and end. Values may be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If `LINE` is omitted, `CURRENT` is assumed. If the line you specify is not a blank line, nothing happens.

Deleting Characters from the Beginning of Lines

You can use the `INDENT_TEXT` subcommand to delete characters from the beginning of lines. Format of the subcommand is:

```
INDENT_TEXT (INDT)
  OFFSET=integer
  NUMBER=integer or keyword value
  LINE=range of integer or keyword value
  STATUS=status variable
```

The `OFFSET (O)` parameter specifies the number of columns to indent the block of text. Values can be an integer. If you specify a negative value, that number of characters will be deleted from the beginning of the lines in the block of text. If `OFFSET` is omitted, a value of 1 is assumed.

The `NUMBER (N)` parameter specifies the number of lines to be indented. Values may be a line number or any of the `NUMBER` keywords specified in the Common Parameters section of chapter 4. If you specify a range for the `LINE` parameter, this parameter assumes a value of `ALL`. Otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines to be indented. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single line is specified, only that line is indented. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The following examples show how you might use the `INDENT_TEXT` subcommand.

```
indent_text offset=-5 line=all
```

Deletes the first 5 characters from all lines ranging from the first line to the last line.

```
indt offset=-10
```

Deletes the first 10 characters from the current line.

```
indt offset=-7 line=25..l
```

Deletes the first 7 characters from lines 25 through the last line.

Replacing

There are two ways to replace old text with new text. For screen mode, the easiest way is to just type the new text over the old. For line mode and screen mode you can also use subcommands.

Replacing Lines

To replace lines of text, use the `REPLACE_LINES` subcommand. This deletes the old lines and replaces them with the lines you specify. Format of the subcommand is:

REPLACE_LINES (REPL)

TEXT=range of string

NEW_TEXT=string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

UNTIL=string

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

STATUS=status variable

The `TEXT (T)` parameter specifies the text you want to replace. If a range of text is specified, the lines containing the entire range is replaced with the string supplied in the `NEW_TEXT` parameter. If `TEXT` is omitted, the `LINE` and `NUMBER` parameters determine the lines to be replaced.

The `NEW_TEXT (NT)` parameter specifies the new line of text that is to replace the specified line(s). If you omit this parameter, you are prompted to enter text line by line until the editor encounters the character(s) specified by the `UNTIL` parameter.

The `NUMBER (N)` parameter specifies the number of lines to replace. Values can be an integer or the keyword `ALL`. If you omit this parameter and specify a range for the `LINE` parameter, the assumed value is `ALL`. If a range of text is specified, the `NUMBER` parameter indicates the number of blocks of text to replace. Otherwise, a value of 1 is assumed.

The `LINE (LINES or L)` parameter specifies a range of lines in which the replacement is to occur. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single value is specified, only that line is replaced. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The UNTIL (U) parameter specifies a character that stops input of replacement text. If the NEW_TEXT parameter is omitted, you are prompted to enter input until the editor encounters the character(s) you specify with this parameter as the last character(s) of a string. If UNTIL is omitted, ** is assumed.

The UPPER_CASE (UC) parameter determines the significance of capitalization in a search. When the value is TRUE, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, B matches to both B and b. If the value is FALSE, the editor searches for the string exactly as you entered it. If you do not specify a value, FALSE is assumed.

The WORD (W) parameter specifies whether to search for the specified text string as a word. When the value is TRUE, it instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the WORD parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters @, #, \$, and _ are allowed as characters in words. When WORD is omitted, FALSE is assumed.

The REPEAT_SEARCH (RS) parameter instructs the editor on how to use the values for the last TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters. TRUE instructs the editor to use the same TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify TRUE for REPEAT_SEARCH and specify values for TEXT, NEW_TEXT, UPPER_CASE, or WORD, the new values for TEXT, NEW_TEXT, UPPER_CASE, and WORD are used. FALSE instructs the editor to use the parameters entered with the current REPLACE_LINES subcommand. If REPEAT_SEARCH is omitted, FALSE is assumed.

The following are examples of how you might use the REPLACE_LINE subcommand:

```
replace_line new_text='text' line=30..l
```

Replaces lines 30 to the end of the file with a line that says: text.

```
repl
```

Replaces the current line with text you are prompted to enter until the editor encounters ** at the end of one of the replacement lines.

Replacing Blocks of Text

To replace blocks of text, use the `REPLACE_TEXT` subcommand. The format of the subcommand is:

```
REPLACE_TEXT (REPT or R)
  TEXT=string
  NEW_TEXT=string
  NUMBER=integer or keyword value
  LINE=range of integer or keyword value
  UPPER_CASE=boolean
  WORD=boolean
  REPEAT_SEARCH=boolean
  VETO=boolean
  STATUS=status variable
```

The `TEXT` (T) parameter specifies the text string to replace in the specified block of text. If `TEXT` is omitted, `REPEAT_SEARCH` is required.

The `NEW_TEXT` (NT) parameter specifies the replacement text for the string specified in the `TEXT` parameter. If `NEW_TEXT` is omitted, the string specified in the `TEXT` parameter is deleted.

The `NUMBER` (N) parameter specifies the number of times the original text is to be replaced within the block of text. Values may be an integer or the keyword `ALL` (A). If you omit this parameter and specify a range of values for the `LINE` parameter, `ALL` is assumed. Otherwise, a value of 1 is assumed.

The `LINE` (LINES or L) parameter specifies the range of lines affected by the replacement. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single line is specified, only that line is replaced. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The `UPPER_CASE` (UC) parameter determines the significance of capitalization in a search. When the value is `TRUE`, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, `B` matches to both `B` and `b`. If the value is `FALSE`, the editor searches for the string exactly as you entered it. If you do not specify a value, `FALSE` is assumed.

The WORD (W) parameter, when the value is TRUE, instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the WORD parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters @, #, \$, and _ are allowed as characters in words. When WORD is omitted, FALSE is assumed.

The REPEAT_SEARCH (RS) parameter instructs the editor on how to use the values for the last TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters. TRUE instructs the editor to use the same TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify TRUE for REPEAT_SEARCH and specify values for TEXT, NEW_TEXT, UPPER_CASE, or WORD, the new values for TEXT, NEW_TEXT, UPPER_CASE, and WORD are used. FALSE instructs the editor to use the parameters entered with the current REPLACE_TEXT subcommand. If you omit this parameter, TEXT is required. Also, if REPEAT_SEARCH is omitted, FALSE is assumed.

The VETO (V) parameter enables you to display a directory of replaced lines, allowing you to veto any of the displayed lines affected by the subcommand and also to choose a line at which you want the cursor to be positioned, as described in the VETO Parameters section of chapter 4. If VETO is omitted, FALSE is assumed.

The following are examples of how you might use the REPLACE_TEXT subcommand.

```
replace_text text='water' new_text='wine'
```

Changes the first occurrence of water to wine from the current line to the last.

```
replace_text repeat_search=true
```

Uses the same values for TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters specified on a previous subcommand.

```
rept text='$' new_text='#'
```

Replaces the first occurrence of \$ with # from the current line to the last.

```
r text='Jill' new_text='Betty' line=50..last
```

Replaces all occurrences of Jill with Betty from line 50 to the end of the file.

Marking


Marking text enables you to specify a group of text to be processed later by another subcommand. When marked text is referenced by another subcommand, the editor remembers the lines, columns, and file containing the marked text. When the marked text is contained in a file other than the current file, the file containing the marked text is made current. Text remains marked until you do one of the following:

- Mark a new region of text.
- Unmark the text with the UNMARK subcommand.
- Delete or move all of the marked text.
- Close the file containing the marked text.
- Undo.

There are several function keys and subcommands which enable you to mark and unmark text bounded by lines or characters.

Marking with Function Keys

Use the function keys provided to mark text in screen mode. These keys are:

Key	Function
F1 MARK	Marks a line of text to be processed later. The marked text is highlighted on terminals with that capability. To mark a range of lines, position the cursor on the first line of the range and press F1 MARK , move the cursor to the last line, and press F1 MARK again. All lines in the range are then highlighted.
 F1 Unmrk	Unmarks any marked text.
F2 Chrmk	Marks a character in the text to be processed later. The marked text is highlighted on terminals with that capability. To mark a range of characters, position the cursor on the first character of the range and press F2 Chrmk , move the cursor to the last character of the range, and press F2 Chrmk again. The text, from the first character through the last, is then highlighted.

Marking Lines with Subcommands

Use the MARK_LINES subcommand to mark a line to be processed later. The format of the subcommand is:

MARK_LINES (MARL or ML)
LINE=range of integer or keyword value
STATUS=status variable

The LINE (LINES or L) parameter specifies a line or range of lines to be marked. Values can be an integer or any of the LINE keyword values described in the Common Parameters section of chapter 4. If a line is specified, only that line is marked. If a single line is specified and another single line is already marked, the lines between the two will become marked. If a range is specified, the entire range is marked and any other marks are unmarked. Marked text can be processed by subcommands that insert, delete, move, copy, and replace text. If LINE is omitted, the current line is assumed.

Marking a Box with Subcommands

Use the MARK_BOX subcommand to mark a rectangular area of text. The format of the subcommand is:

MARK_BOX (MARB or MB)
LINE=range of integer or keyword value
COLUMN=integer or keyword value
STATUS=status variable

The LINE (LINES or L) parameter specifies the lines in which the corners of the box reside. Values can be a line number or any of the LINE keyword parameters described in the Common Parameters section of chapter 4. If LINE is omitted, CURRENT is assumed.

The COLUMN (COLUMNS or C) parameter specifies the column in which a corner of the box resides. Values can be any number from 1 to 256, or any of the COLUMN keyword parameters described in the Common Parameters section of chapter 4. If COLUMN is omitted, CURRENT is assumed.

For example, to mark a box with dimensions: 5 lines by 1 column, enter:

```
mark_box lines=4..8 columns=12
```

The marked area covers lines 4, 5, 6, 7, and 8 at column 12.

NOTE

At this time, the only operations supported for box marks are the \$MARK_FIRST_COLUMN, \$MARK_FIRST_LINE, \$MARK_LAST_COLUMN, \$MARK_LAST_LINE, and \$MARK_TYPE functions. These allow users to implement their own SCL procedures to operate on the rectangular area of text. None of the CDC-supplied editor subcommands support box marks.

Marking Characters with Subcommands

Use the `MARK_CHARACTER` subcommand to mark specific characters. This subcommand specifies column boundaries for text to be processed later by another subcommand. Format of the subcommand is:

```
MARK_CHARACTER (MARC or MC)
  LINE=range of integer or keyword value
  COLUMN=range of integer or keyword value
  STATUS=status variable
```

The `LINE` (`LINES` or `L`) parameter specifies the lines in which the marked characters reside. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If `LINE` is omitted, `CURRENT` is assumed.

The `COLUMN` (`COLUMNS` or `C`) parameter specifies the columns to be marked within the specified line(s). Values can be any number from 1 to 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4. You may not specify a range. If `COLUMN` is omitted, `CURRENT` is assumed.

The following subcommands show how you might use the `MARK_CHARACTERS` subcommand.

```
mark_character
  Marks the current character in the current line.

mark_character line=40..50 column=30
  Marks column 30 of line 40 through line 50.

mark_character column=7
  Marks column 7 of the current line.
```

Unmarking with Subcommands

To explicitly unmark lines or characters you have previously marked, use the `UNMARK` subcommand. This subcommand cancels the marks on the file. The format of the subcommand is:

```
UNMARK (UNM)
  STATUS=status variable
```

You can implicitly unmark text, mark a new region of text, or delete the marked text when you enter `UNDO`. Also when you enter the `END_FILE` subcommand, you can close a file containing marked text.

Saving Positions with Subcommands

The following subcommands enable you to save the current column, line, and file name and to return to that file position later.

To save a position in a file, move the cursor to the position you want to save, press **(HOME)**, and enter the `SAVE_POSITION`. It saves the current column, line, and file name for reference later by the `RESTORE_POSITION` subcommand. Format of the subcommand is:

SAVE_POSITION (SAVP)
STATUS=status variable

The `SAVE_POSITION` can save only one position. A position is saved until you either enter another `SAVE_POSITION` subcommand or enter an `EXCHANGE_POSITION` subcommand, or you close the file.

Pressing the **(DATA)** key is the equivalent to entering the `SAVE_POSITION` subcommand.

To return to this position later, use the `RESTORE_POSITION` subcommand. Format of the subcommand is:

RESTORE_POSITION (RESP)
STATUS=status variable

To save the current position in the file you are editing while you return to a previously saved position, use the `EXCHANGE_POSITION` subcommand. The format of the subcommand is:

EXCHANGE_POSITION (EXCP)
STATUS=status variable

Pressing the **(BACK)** key is the equivalent to entering the `EXCHANGE_POSITION` subcommand. You must save a position with the `SAVE_POSITION` subcommand before you can use the `EXCHANGE_POSITION` subcommand.

Copying

You can copy text to different spots in the same file or to a different working file. You can also copy text to and from external files.

Copying with Function Keys

To copy text in screen mode, use either the F1 **MARK** or F2 **Chrmk** key along with the F14 **COPY** key. The keys perform the following functions.

Key	Function
F1 MARK	Marks the current line as a boundary for text to be copied. Move the cursor up or down to the second boundary and press F1 MARK again to create a block of lines to be copied. If only one line is to be copied, press F1 MARK only once.
F2 Chrmk	Marks the current character as a boundary of a series of characters to be copied. Move the cursor to the second boundary and press F2 Chrmk again to create the range of characters to be copied. If only one character is to be copied, press F2 Chrmk only once.
F14 COPY	Copies any marked text to immediately before the current line if you used the F1 MARK key or the current character if you used the F2 Chrmk key. If no marks are set, the current line is copied to the next line.

To copy text from one file to another, use the `SET_SCREEN_OPTIONS` subcommand, F1 **MARK**, and F14 **COPY**. Using the `SET_SCREEN_OPTIONS` subcommand, you can display more than one file on the screen at a time (the `SET_SCREEN_OPTIONS` subcommand is described in detail in chapter 7, Selecting Editor Options). You can then mark the lines you want to copy, move the cursor to the position in the file at which you want the copied text to appear, and press F14 **COPY**.

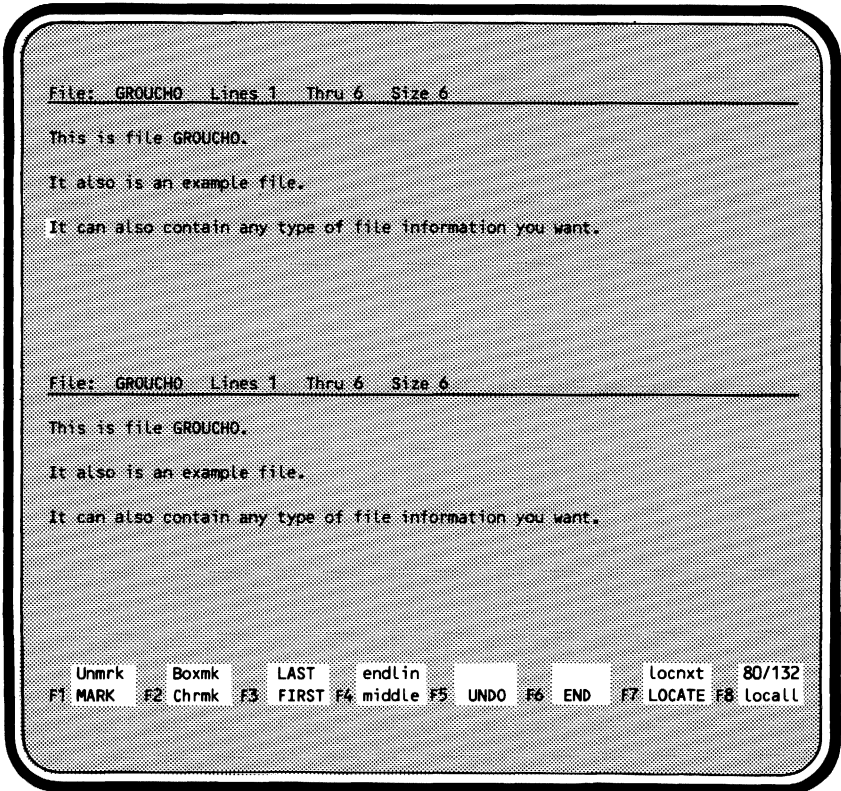
For example, you are editing file GROUCHO and want to copy 5 lines from file HARPO. First, press

HOME

and enter

```
set_screen_options splits=2
```

This splits the screen with two areas displaying text of file GROUCHO:



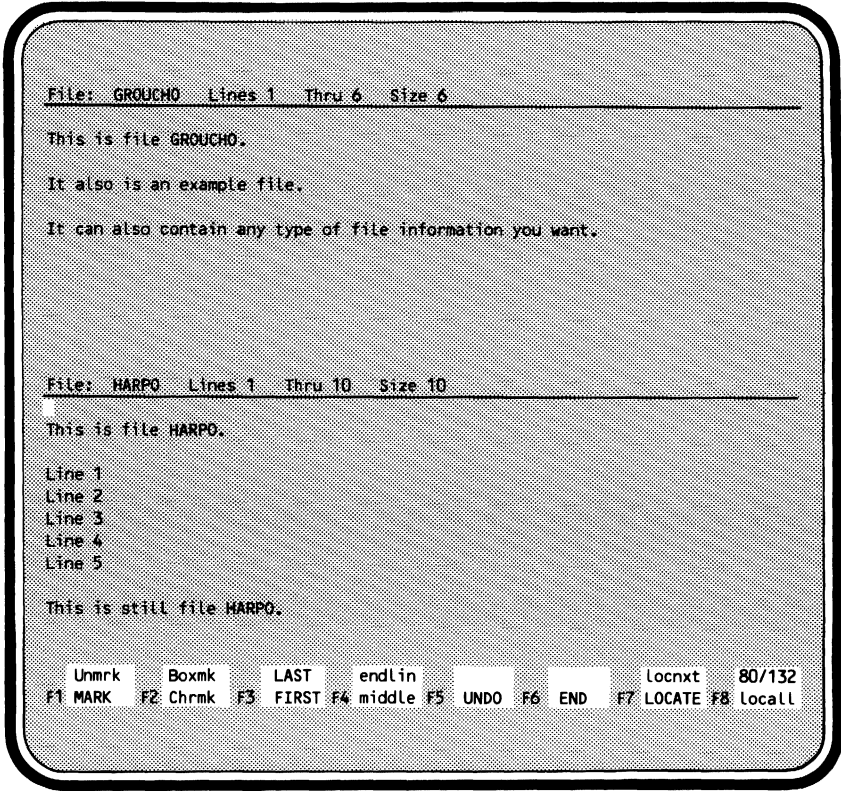
Move the cursor to the text area of the lower split of the screen and press:

(HOME)

Enter:

```
edit_file harpo
```

File HARPO appears:



Move the cursor to the first line of text you want copied to file GROUCHO:

File: GROUCHO Lines 1 Thru 6 Size 6

This is file GROUCHO.

It also is an example file.

It can also contain any type of file information you want.

File: HARPO Lines 1 Thru 10 Size 10

This is file HARPO.

Line 1

Line 2

Line 3

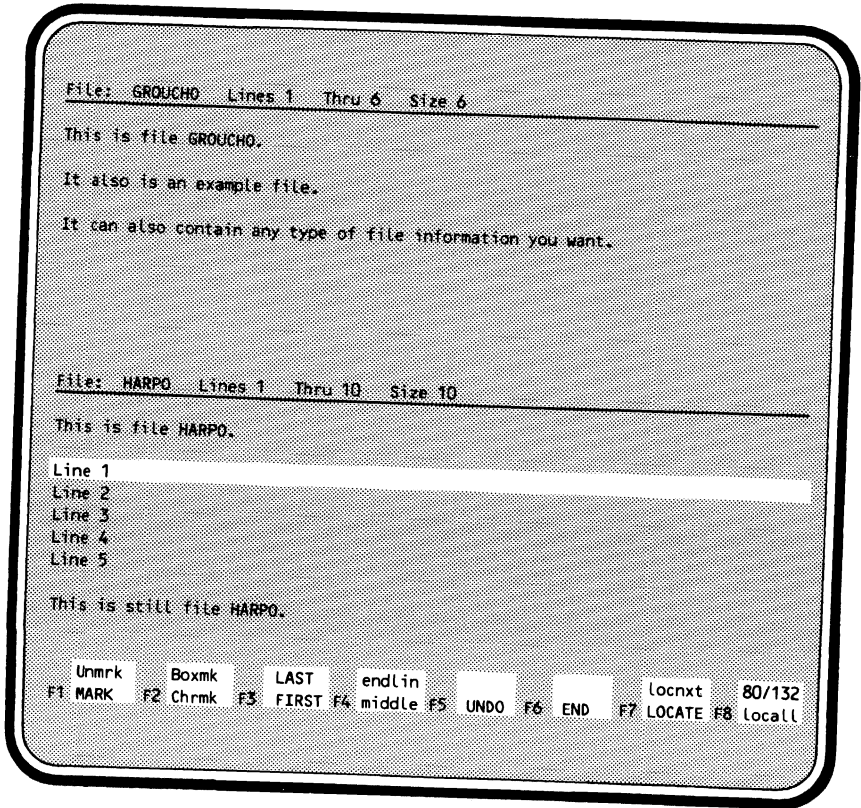
Line 4

Line 5

This is still file HARPO.

Unmrk	Boxmk	LAST	endlin			Locnxt	80/132
F1 MARK	F2 Chrmk	F3 FIRST	F4 middle	F5 UNDO	F6 END	F7 LOCATE	F8 locall

Press F1 **MARK**. The line is highlighted.



Press F1 **MARK** again. The text from the first marked line through the last marked line are highlighted:

File: GROUCHO Lines 1 Thru 6 Size 6

This is file GROUCHO.
It also is an example file.
It can also contain any type of file information you want.

File: HARPO Lines 1 Thru 10 Size 10

This is file HARPO.

Line 1
Line 2
Line 3
Line 4
Line 5

This is still file HARPO.

Unmrk	Boxmk	LAST	endlin		locnxt	80/132	
F1 MARK	F2 Chrmk	F3 FIRST	F4 middle	F5 UNDO	F6 END	F7 LOCATE	F8 LocalL

Move the cursor to the location in file GROUCHO at which you want the copied text to appear:

```

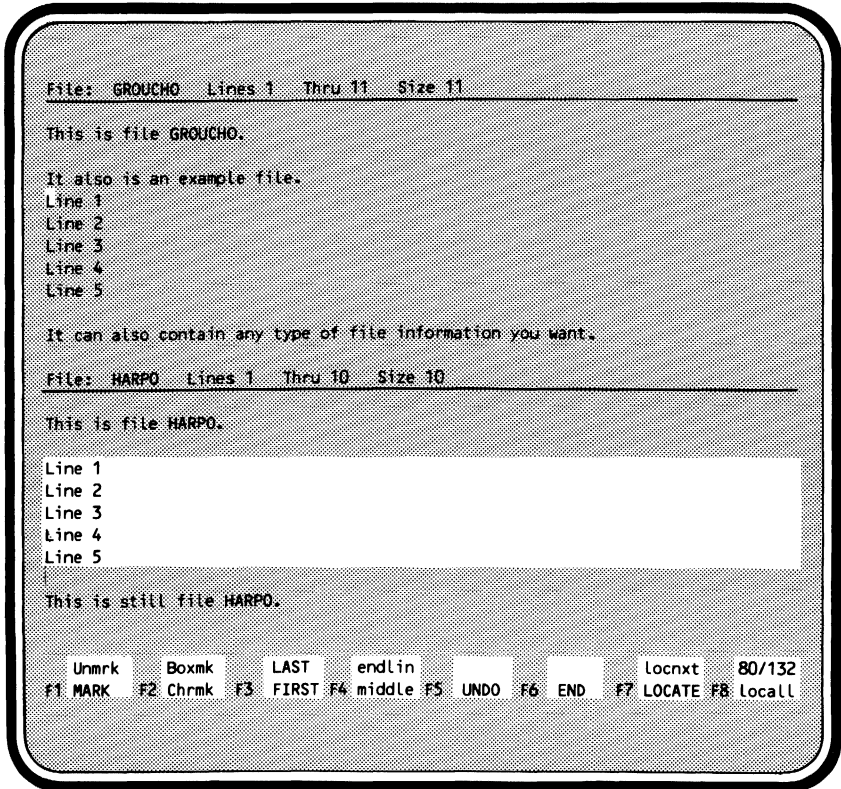
File: GROUCHO Lines 1 Thru 6 Size 6
-----
This is file GROUCHO.
It also is an example file.
It can also contain any type of file information you want.

File: HARPO Lines 1 Thru 10 Size 10
-----
This is file HARPO.
Line 1
Line 2
Line 3
Line 4
Line 5

This is still file HARPO.

Unmk   Boxmk  LAST   endlin  locnxt  80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO  F6 END   F7 LOCATE F8 locall
    
```

Press **COPY**. The text is copied to file GROUCHO:



The text remains marked until you UNDO, set new marks, or unmark them.

To return to editing just file GROUCHO, press:

EDIT

Copying Blocks of Text with Subcommands

To copy a block of text from one place to another within your working files use the COPY_TEXT subcommand. Format of the subcommand is:

COPY_TEXT (COPT or C)

TEXT=range of string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

COLUMN=range of integer or keyword value

INSERTION_LOCATION=integer or keyword value

INSERTION_COLUMN=integer or keyword value

PLACEMENT=keyword

BOUNDARY=keyword value

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

STATUS=status variable

The TEXT (T) parameter specifies strings of text in the first and last lines of a block of text to be copied. If you enter only one string, the block of text to be copied will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found. If you omit this parameter, the lines to be copied will be determined by the NUMBER and LINE parameters. If TEXT is omitted, and you specify the REPEAT_SEARCH parameter as TRUE, this parameter assumes the value last specified for the TEXT parameter on any subcommand.

The NUMBER (N) parameter specifies the number of blocks of text to be copied. Values for this parameter can be numbers or the keyword ALL (A). If you specify a range for the LINE parameter, this parameter assumes a value of ALL, otherwise a value of 1 is assumed.

The LINE (LINES or L) parameter specifies a range of lines to be searched for the text to be copied. Values can be an integer or any of the LINE keyword values described in the Common Parameters section of chapter 4. If a single value is specified, only that line is searched. If LINE is omitted, CURRENT..LAST is assumed.

The COLUMN (COLUMNS or C) parameter specifies the range of columns to be searched for the text to be copied. The integers can be from 1 through 256 or any of the keyword values listed in the Common Parameters section of chapter 4. If COLUMNS is omitted, CURRENT is assumed.

The `INSERTION_LOCATION` (`IL`) parameter specifies the line before which or after which the line is to be copied (depending on the value of the `PLACEMENT` parameter). Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `INSERTION_COLUMN` (`INSERTION_COLUMNS` or `IC`) parameter specifies the column before which or after which the text is to be copied (depending on the value of the `PLACEMENT` parameter). Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `INSERTION_COLUMN` is omitted, `CURRENT` is assumed.

The `PLACEMENT` (`P`) parameter specifies if the copied lines are to appear before or after the location specified by the `INSERTION_LOCATION` parameter. Values can be `BEFORE` (`B`) or `AFTER` (`A`). If `PLACEMENT` is omitted, `AFTER` is assumed.

The `BOUNDARY` (`B`) parameter specifies the type of boundary that will limit the search. Values can be `LINE` or `STREAM` as described in the Common Parameters section of chapter 4. If `BOUNDARY` and `COLUMNS` are omitted, `LINE` is assumed. If a value for `COLUMNS` is specified and `BOUNDARY` is omitted, `STREAM` is assumed.

The `UPPER_CASE` (`UC`) parameter determines the significance of capitalization in a search. When the value is `TRUE`, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, `B` matches to both `B` and `b`. If the value is `FALSE`, the editor searches for the string exactly as you entered it. If you do not specify a value, `FALSE` is assumed.

The `WORD` (`W`) parameter, when the value is `TRUE`, instructs the editor to search for the text to be copied as a word. That is, the text you want to move is surrounded by nonalphanumeric characters. The most common use of the `WORD` parameter is to search for a string to be copied that is surrounded by blanks or punctuation characters (just as each word on this page is surrounded by blanks or punctuation characters). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. When `WORD` is omitted, `FALSE` is assumed.

The `REPEAT_SEARCH` (`RS`) parameter instructs the editor on how to use the values for the last `TEXT`, `UPPER_CASE`, and `WORD` parameters. `TRUE` instructs the editor to use the same `TEXT`, `UPPER_CASE`, and `WORD` parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify `TRUE` for `REPEAT_SEARCH` and specify values for `TEXT`, `UPPER_CASE`, or `WORD`, the new values for `TEXT`, `UPPER_CASE`, and `WORD` are used. `FALSE` instructs the editor to use the parameters entered with the current `COPY_TEXT` subcommand. If `REPEAT_SEARCH` is omitted, `FALSE` is assumed.

Following are examples of how you might use the COPY_TEXT subcommand.

```
copy_text line=30..40
```

Copies lines 30 through 40 to immediately after the current line.

```
copy_text='one'..'five' insertion_location=71 placement=before
```

Copies the next occurrence of a block of text beginning with the line containing one and ending with the line containing five to immediately before line 71.

Copying Text Between Working Files and External Files

To copy text from the current working file to the external copy of a file, use the WRITE_FILE subcommand. (To copy text from another file to the current file, use the READ_FILE subcommand described later in this chapter.) The format is:

WRITE_FILE (WRIF)

TEXT=range of string

NUMBER=integer or keyword value

LINE=range of integer or keyword value

FILE=file reference

UPPER_CASE=boolean

WORD=boolean

REPEAT_SEARCH=boolean

MULTI_PARTITION=boolean

STATUS=status variable

The TEXT (T) parameter specifies the strings of text which identify the first and last lines of a block of text to be written. If you enter only one string, the block of text to be written will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found and the cursor is positioned at the beginning of the first string. If you omit the TEXT parameter, the lines to be written will be determined by the NUMBER, LINE, and DIRECTION parameters. If you omit the TEXT parameter and specify the REPEAT_SEARCH parameter as TRUE, this parameter assumes the value last specified on a subcommand with a TEXT parameter.

The NUMBER (N) parameter specifies the number of blocks of text to be copied. Values for this parameter can be an integer or the keyword value ALL (A). If NUMBER is omitted, ALL is assumed.

The LINE (LINES or L) parameter specifies a range of lines to be searched to locate the text to be copied. Values can be an integer or any of the LINE keyword values described in the Common Parameters section of chapter 4. If a single value is specified, only that line is searched. If LINE is omitted, ALL is assumed.

The **FILE (F)** parameter specifies the file to which the text is to be copied. The text from the current file is written to the external copy of the file specified, not a working copy. The specified file may or may not be overwritten depending on the file position specified on the **FILE** parameter and on how you open your external files (refer to chapter 4, Before You Continue, for information on external files). If you have not changed how your external files are opened, you can avoid overwriting a file by adding **.\$EOI** to the file name. For example, to add text from the current file to the end of file **ZAP**, specify the **FILE** parameter as **ZAP.\$EOI**. If **FILE** is omitted, the external copy of the current file is assumed.

The **UPPER_CASE (UC)** parameter determines the significance of capitalization in a search. When the value is **TRUE**, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, **B** matches to both **B** and **b**. If the value is **FALSE**, the editor searches for the string exactly as you entered it. If you do not specify a value, **FALSE** is assumed.

The **WORD (W)** parameter, when the value is **TRUE**, instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the **WORD** parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters **@**, **#**, **\$**, and **_** are allowed as characters in words. When **WORD** is omitted, **FALSE** is assumed.

The **REPEAT_SEARCH (RS)** parameter instructs the editor on how to use the values for the last **TEXT**, **UPPER_CASE**, and **WORD** parameters. **TRUE** instructs the editor to use the same **TEXT**, **UPPER_CASE**, and **WORD** parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify **TRUE** for **REPEAT_SEARCH** and specify values for **TEXT**, **UPPER_CASE**, or **WORD**, the new values for **TEXT**, **UPPER_CASE**, and **WORD** are used. **FALSE** instructs the editor to use the parameters entered with the current **WRITE_FILE** subcommand. If **REPEAT_SEARCH** is omitted, **FALSE** is assumed.

The **MULTI_PARTITION (MP)** parameter specifies whether the editor is to change **WEOP** directives to end-of-partition delimiters when the current working file is copied to an external file. When the value is **TRUE**, the editor changes **WEOP** directives to end-of-partition delimiters. When the value is **FALSE**, no substitution takes place. If **MULTI_PARTITION** is omitted, **FALSE** is assumed. Refer to **Creating Multipartition Files** later in this chapter for more information.

The following subcommands show how you might use the `WRITE_FILE` subcommand.

```
write_file line=20..last file=zap
```

Copies lines 20 through the last line of the current file to file ZAP.

```
write_file text='int'..'end' number=3 file=nertz
```

Copies three blocks of text beginning with the line containing `int` and ending with the line containing `end` to file NERTZ.

```
write_file line=all file=splat
```

Copies all lines from the current file to the external copy of file SPLAT.

```
wrif file=ZAP.$EOI
```

Copies all of the current working file to the end of file ZAP.

```
wrif
```

Copies the working copy of the current file to the external copy. In other words, it makes your changes permanent without closing the current file and leaving the editor.

To copy lines from an external file into the current working file, use the `READ_FILE` subcommand. This subcommand copies all text from another file into the current file. Format of the subcommand is:

READ_FILE (REAF)

FILE=file

INSERTION_LOCATION=integer or keyword value

PLACEMENT=keyword value

MULTI_PARTITION=boolean

STATUS=status variable

The `FILE (F)` parameter specifies the name of the file from which the text is to be copied. The entire file will be copied. The `READ_FILE` subcommand reads text from the external copy of the specified file, not a working copy. This parameter is required.

The `INSERTION_LOCATION` (IL) parameter specifies the line before which or after which the line is to be copied (depending on the value of the `PLACEMENT` parameter). Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `PLACEMENT` (P) parameter specifies if the copied lines are to appear before or after the location specified by the `INSERTION_LOCATION` parameter. Values can be `BEFORE` (B) or `AFTER` (A). If `PLACEMENT` is omitted, `AFTER` is assumed.

The `MULTI_PARTITION` (MP) parameter specifies whether the editor is to change the end-of-partition delimiters in the file to `WEOP` directives. When the value is `TRUE`, the editor changes the delimiters to `WEOP` directives. When the value is `FALSE`, the editor stops reading the file at the first end-of-partition boundary it encounters. If `MULTI_PARTITION` is omitted, `FALSE` is assumed. Refer to *Creating Multipartition Files* later in this chapter for more information.

The following examples show how you might use the `READ_FILE` subcommand.

```
read_file file=ernie insertion_location=320
```

Copies the contents of file `ERNIE` into the current file immediately after line 320.


```
read_file file=bert insertion_location=last_mark placement=before
```




Copies the contents of file `BERT` into the current file immediately before the last marked line.

Moving

The editor enables you to move text to different locations in the same file or to a different file. Moving text using several function keys in screen mode is easy. There is also a subcommand which enables you to perform the same functions in line mode.

Moving with Function Keys

To move text in screen mode, use the F1 **MARK** (or F2 **Chrmk**) key along with the  F14 **Move** keys. The keys perform the following functions.

Key	Function
F1 MARK	Marks the current line as a boundary for text to be copied. Move the cursor up or down to the second boundary and press F1 MARK again to create a block of lines to be copied. If only one line is to be copied, press F1 MARK only once.
F2 Chrmk	Marks the current character as a boundary of a series of characters to be copied. Move the cursor to the second boundary and press F2 Chrmk again to create the range of characters to be copied. If only one character is to be copied, press F2 Chrmk only once.
 F14 Move	Moves any marked text to immediately before the current line or character.
 F1 MARK	Cancels marks.
 F2 Chrmk	Cancels marks.

Moving with Subcommands

To move a block of text from one place to another in the same file, use the `MOVE_TEXT` subcommand. Format of the subcommand is:

MOVE_TEXT (MOV`T` or `M`)

TEXT=range of string
NUMBER=integer or keyword value
LINE=range of integer or keyword value
COLUMN=range of integer or keyword value
INSERTION_LOCATION=integer or keyword value
INSERTION_COLUMN=integer or keyword value
PLACEMENT=keyword value
BOUNDARY=keyword value
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

The `TEXT` (`T`) parameter specifies strings of text in the first and last lines of a block of text to be moved. If you enter only one string, the block of text to be moved will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found. If `TEXT` is omitted, the lines to be moved will be determined by the `NUMBER`, and `LINE` parameters.

The `NUMBER` (`N`) parameter specifies the number of blocks of text to be moved. Values for this parameter can be numbers or the keyword `ALL` (`A`). If you specify a range for the `LINE` parameter, this parameter assumes a value of `ALL`; otherwise, a value of 1 is assumed.

The `LINE` (`LINES` or `L`) parameter specifies a range of lines to be searched for the text to be moved. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If a single value is specified, only that line is searched. If `LINE` is omitted, `CURRENT..LAST` is assumed.

The `COLUMN` (`COLUMNS` or `C`) parameter specifies the range of columns to be searched for the text to be moved. The integers can be from 1 through 256 or any of the `COLUMN` keyword values listed in the Common Parameters section of chapter 4. If `COLUMN` is omitted, `CURRENT` is assumed.

The `INSERTION_LOCATION` (IL) parameter specifies the line before which or after which the line is to be moved (depending on the value of the `PLACEMENT` parameter). Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `INSERTION_LOCATION` is omitted, `CURRENT` is assumed.

The `INSERTION_COLUMN` (`INSERTION_COLUMNS` or `IC`) parameter specifies the column before which or after which the text is to be moved (depending on the value of the `PLACEMENT` parameter). Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values listed in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `INSERTION_COLUMN` is omitted, `CURRENT` is assumed.

The `PLACEMENT` (P) parameter specifies if the moved lines are to appear before or after the location specified by the `INSERTION_LOCATION` parameter. Values can be `BEFORE` (B) or `AFTER` (A). If `PLACEMENT` is omitted, `AFTER` is assumed.

The `BOUNDARY` (B) parameter specifies the type of boundary that will limit the search. Values can be `LINE` or `STREAM` as described in the Common Parameters section of chapter 4. If `BOUNDARY` and `COLUMNS` are omitted, `LINE` is assumed. If a value for `COLUMNS` is specified and `BOUNDARY` is omitted, `STREAM` is assumed.

The `UPPER_CASE` (UC) parameter determines the significance of capitalization in a search. When the value is `TRUE`, the editor matches strings assuming there is no distinction between uppercase and lowercase letters. For example, B matches to both B and b. If the value is `FALSE`, the editor searches for the string exactly as you entered it. If you do not specify a value, `FALSE` is assumed.

The `WORD` (W) parameter, when the value is `TRUE`, instructs the editor to search for the specified string as a word. That is, the text you want to find is surrounded by nonalphanumeric characters. The most common use of the `WORD` parameter is to search for a string surrounded by blanks (just as each word on this page is surrounded by blanks). The first and last columns on the screen are also considered to be nonalphanumeric characters and are interpreted as boundaries. The characters `@`, `#`, `$`, and `_` are allowed as characters in words. When `WORD` is omitted, `FALSE` is assumed.

The REPEAT_SEARCH (RS) parameter instructs the editor on how to use the values for the last TEXT, UPPER_CASE, and WORD parameters. TRUE instructs the editor to use the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand. In other words, if you specify TRUE for REPEAT_SEARCH and specify values for TEXT, UPPER_CASE, or WORD, the new values for TEXT, UPPER_CASE, and WORD are used. FALSE instructs the editor to use the parameters entered with the current MOVE_TEXT subcommand. If REPEAT_RESEARCH is omitted, FALSE is assumed.

Following are examples of how you might use the MOVE_TEXT subcommand.

```
move_text line=30..40
```

Moves lines 30 through 40 to immediately after the current line.

```
movt text='one'..'five' insertion_location=71 placement=before
```

Moves the next occurrence of a block of text beginning with the line containing one and ending with the line containing five to immediately before line 71.

Undoing


The editor provides you with the capability of undoing changes you have made to your files. There is a function key for use in screen mode and subcommands for use in either screen or line mode.

Undoing with a Function Key

The F5 **UNDO** key undoes any changes you have made since the last time you pressed **(NEXT)** or pressed a function key that supplies a **(NEXT)**. (All function keys on a CDC 721 supply a **(NEXT)**.) If you entered a subcommand or pressed **(NEXT)** to move the cursor, but made no changes to the file, the editor goes back until it encounters a change and that change is undone.

Each time you press F5 **UNDO**, changes that occurred between **(NEXT)** key entries are undone in reverse order.

For example, the following changes were made to a file in the order shown:

1. All abc's were changed to xyz's using the REPLACE_TEXT subcommand. **(NEXT)** was pressed.
2. The word water was changed to wine by typing wine over water.
3. The first line of the file was deleted using the  **(DELETE)** keys.

Each time you enter F5 **UNDO**, the changes are undone as shown:

The first time F5 **UNDO** is entered. The first line of the file is returned and the word wine is changed back to water.

The second time The xyz's are changed back to abc's.

Undoing with Subcommands

There are two subcommands which you can use to undo changes you have made to your file. The UNDO subcommand undoes changes one at a time in the reverse order they were made. The RESET_FILE subcommand cancels all changes you have made to your file since it was opened.

To undo changes one at a time, use the UNDO subcommand. For each UNDO, all changes made since the last time you pressed **(NEXT)** are canceled. As you continue to enter UNDO, changes that occurred between **(NEXT)** key entries are undone in reverse order.

The format of the subcommand is:

UNDO

STATUS=status variable

For example, the following changes were made to a file in the order shown.

1. Five lines in the file were deleted using the `DELETE_LINES` subcommand.
2. The next three lines are displayed using the `LOCATE_TEXT` subcommand.
3. A new line is entered using the `INSERT_LINES` subcommand.

Each time `UNDO` is entered, the following changes are undone:

The first time The new line inserted is deleted.
`UNDO` is entered

The second time The five lines deleted are returned.

You can undo only changes made to the current file. You can, however, make any file that was edited during this session the current file if it has not been closed with `END_FILE`, `END_DECK`, or a select deck subcommand. You can do this by entering the `EDIT_FILE` or `EDIT_DECK` subcommand, or, if your screen is split, by positioning the cursor in the file you want to be the current file.

To undo all changes you have made since opening the current file, use the `RESET_FILE` subcommand. This subcommand cancels all the changes you have made to your file since you last accessed the file using the `EDIT_FILE` command. Format of the subcommand is:

RESET_FILE (RESF)

STATUS=status variable

Refer to *Discarding Deck Changes*, in chapter 8, for more information about undoing changes in decks.

Creating Multipartition Files

You can add or delete end-of-partition delimiters to any file you can edit [the file has a record type file attribute of variable (V)].

The SCU text-embedded directive WEOP represents the end-of-partition delimiter within the editor. The format of the directive is:

***WEOP**

The asterisk (*) is the default key character. If you are editing decks, this character was defined when the library was created. For files, the asterisk is always used. The directive starts in column 1 and is the only thing on the line.

You can put this directive in your file or deck where you want end-of-partition delimiters to appear. When you start editing, end-of-partition delimiters become WEOP directives. When you manipulate a file using WRITE_FILE or READ_FILE, the MULTI_PARTITION parameter determines what happens to the WEOP directives.

When you rewrite files or decks using the END, END_FILE, END_DECK, SELECT_DECK, SELECT_FIRST_DECK, SELECT_NEXT_DECK, SELECT_LAST_DECK, or QUIT subcommands, WEOP directives are always converted to end-of-partition delimiters.

For example, the following lines show a WEOP directive separating two blocks of text.



```
Text for the first partition.
*WEOP
Text for the second partition.
```

Text Formatting

There are several function keys and subcommands that enable you to format text. These function keys and subcommands help you make documents of almost any type including memos and reports.

Text Formatting with Function Keys

In addition to the basic editing function keys which enable you to insert, delete, and so on, the editor also provides function keys which apply specifically to text formatting. These keys are:

Key	Function
F10 inswrđ	Inserts 32 spaces on the current line, enabling you to type in a word or phrase.
 F10 delwrđ	Deletes the word on which the cursor is positioned. If the cursor is positioned on a blank, all blanks are deleted until a nonblank character is encountered.
F11 Break	Breaks the current line into two lines. The break occurs right before the current cursor position. The cursor is positioned at the end of the first line of the two resulting lines.
F12 Join	Joins the current line with the next line (does the opposite of the F11 Break key).
F16 FORMAT	Formats the current paragraph.
 F16 center	Centers the current line.

Text Formatting with Subcommands

There are also equivalent subcommands that perform the same functions. The subcommands you can use to break or join text are described here. In addition, there are subcommands you can use to set paragraph margins, format paragraphs, and center lines. These subcommands are also described here. For information on subcommands for inserting or deleting words, refer to the Inserting and Deleting sections earlier in this chapter.

Breaking Text

To break a line at a specific point in the line to make one line into two lines, use the `BREAK_TEXT` subcommand. The format of the subcommand is:

BREAK_TEXT (BRET or B)

LINE=integer or keyword value

COLUMN=integer or keyword value

STATUS=status variable

The `LINE` (`LINES` or `L`) parameter identifies the line to be broken. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `ALL`, `MARK`, and `SCREEN`. Ranges are not allowed. If `LINE` is omitted, `CURRENT` is assumed.

The `COLUMN` (`COLUMNS` or `C`) parameter specifies the column before which the break is to occur. In other words, the break occurs just before the column specified. Values can be an integer from 1 through 256, or any of the `COLUMN` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `COLUMN` is omitted, `CURRENT` is assumed.

Joining Text

To join two lines, use the `JOIN_TEXT` subcommand. This subcommand joins a line with the next line by appending the second to the first. Format of the subcommand is:

JOIN_TEXT (JOIT or J)

LINE=integer or keyword value

COLUMN=integer or keyword value

STATUS=status variable

The `LINE` (`LINES` or `L`) parameter specifies the first of the two lines to be joined. The line following the specified line is the line to which the first is joined. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If `LINE` is omitted, `CURRENT` is assumed.

The `COLUMN` (`COLUMNS` or `C`) parameter specifies the starting column to which the second line is moved. The second line is always added to the end of the first line. The columns parameter determines how far after the end of the first line the second will be added. Values can be an integer from 1 through 256, or any of the `COLUMN` keyword parameters described in the Common Parameters section of chapter 4 with the exception of `MARK`. Ranges are not allowed. If the value you specify is less than or equal to the length of the first line, the line is added to the end of the first line. If the value you specify is greater than the length of the first line, the editor fills the columns in between with blank characters. If `COLUMN` is omitted, `CURRENT` is assumed.

To indent text, use the `INDENT_TEXT` subcommand described in Inserting earlier in this chapter.

Setting Paragraph Margins

To set or change the paragraph margins, use the `SET_PARAGRAPH_MARGINS` subcommand. In any subsequent `FORMAT_PARAGRAPH` or `CENTER_LINE` subcommands, the margins set with `SET_PARAGRAPH_MARGINS` are used. The format is:

SET_PARAGRAPH_MARGINS (SETPM)

MARGIN_COLUMN=range of integer

OFFSET=integer

The `MARGIN_COLUMN (MC)` parameter specifies the right and left margins. If just one column number is specified, the left margin is set to that number. If `MARGIN_COLUMN` is omitted and you have not specified this subcommand previously in your terminal session, columns 1 and 65 are used. If you have specified the subcommand previously, any parameter not specified is not changed.

The `OFFSET (O)` parameter specifies the number of columns the first line in the paragraph is to be offset from the rest of the lines in the paragraph. If the number specified is a positive number, the first line of the paragraph is indented the number of columns specified. If zero is specified, the first line is not indented. If a negative value is given, the first line begins to the left of the rest of the paragraph. If `OFFSET` is omitted and you have specified this subcommand during this terminal session, the previous value is used. If you have not entered this subcommand previously and omit the `OFFSET` parameter, a value of 4 is assumed.

The following examples show how you might want to use the `SET_PARAGRAPH_MARGINS` subcommand.

```
set_paragraph_margins mc=7..72
```

Sets the paragraph margins to column 7 and 72.

```
set_paragraph_margins mc=10..70 o=5
```

Sets the margins to 10 and 70 and also specifies that you want the first line of the paragraph indented 5 columns.

Formatting Paragraphs

To adjust words or sentences in a paragraph of text to bring line lengths as close as possible to preset margins (see the `SET_PARAGRAPH_MARGINS` subcommand), use the `FORMAT_PARAGRAPHS` subcommand. The format of the subcommand is:

FORMAT_PARAGRAPHS (FORP)

LINE=integer or keyword value

NUMBER=integer

The `LINE` (`LINES` or `L`) parameter specifies a range of lines to format. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4. If omitted, the current paragraph is assumed.

The `NUMBER` (`N`) parameter specifies the number of lines to format starting with the current line and moving forward. If `LINE` is omitted and `NUMBER` is specified, the number of lines in the current paragraph specified by the `NUMBER` parameter are formatted. If both `LINE` and `NUMBER` are omitted, the current paragraph is assumed.

The following example shows how you might want to use the `FORMAT_PARAGRAPHS` subcommand.

```
forp n=6
```

Adjusts the current line and the five subsequent lines to conform to previously set margins.

Using the `FORMAT_PARAGRAPHS` subcommand adds 2 blanks after '.', '!', and '?'.

Centering Lines

To center a line or lines between margins that have been previously set using the `SET_PARAGRAPH_MARGINS` subcommand, use the `CENTER_LINES` subcommand. The format of this subcommand is:

`CENTER_LINES (CENL)`

LINE=range of integer or keyword value

NUMBER=integer

The `LINE` (`LINES` or `L`) parameter specifies a line or range of lines to be centered. If you specify only one line, the centering is limited to that line. Values can be an integer or any of the `LINE` keyword values described in the Common Parameters section of chapter 4.

If `LINE` is omitted, the lines to be centered are determined by the `NUMBER` parameter. If `LINE` and `NUMBER` are both omitted, `CURRENT` is assumed.

The `NUMBER` (`N`) parameter specifies the number of lines to be centered. If `NUMBER` is omitted, the lines to be centered are determined by the `LINE` parameter. If `NUMBER` and `LINE` are both omitted, `CURRENT..LAST` is assumed.

The following examples show how you might want to use the `CENTER_LINES` subcommand.

```
center_lines number=5
```

Centers the next five lines.

```
cenl line=15..23
```

Centers all lines between lines 15 and 23.

```
cenl
```

Centers all lines between the current line and the last line.



In screen mode, you can use function keys to perform many editing tasks that would otherwise require you to enter subcommands. This chapter describes those function keys.

Editing Keys	6-1
CDC Standard Function Keys	6-2
Programmable Function Keys	6-3



In addition to the standard ASCII character keys, like **(A)** and **(5)**, the editor supports three types of function keys: editing keys, CDC standard keys, and programmable keys. This chapter describes these keys and how you can redefine the programmable keys.

Editing Keys

The terminal does the processing of the editing keys. Since the editor is not involved, the operations are instantaneous. The editing keys and their functions are:

Key	Function
(INSRT)	Inserts a blank character, allowing you to type a new character over the blank. Some terminals put you in insertion mode, which enables you to enter the character itself rather than a space to type in the new character.
(↑) (INSRT)	Inserts a blank line.
(DELETE)	Deletes the current character.
(↑) (DELETE)	Deletes the current line. Press: (NEXT) to fill in the lines at the bottom of the screen with text from the file.
(ERASE)	Backspaces a single character and deletes it.
(↑) (ERASE)	Blanks the current line and positions the cursor in column 1 of the blank line.
(→)	Moves the cursor forward to the next tab. Refer to the SET_TAB_OPTIONS in chapter 7, Selecting Editor Options, for information on setting tabs.
(←)	Moves the cursor back to the immediately preceding tab.
(CLEAR)	Deletes all characters from the cursor to the end of the line.
(↑) (CLEAR)	Clears the entire screen; the editor completely rewrites the screen if you clear it. If you suspect the screen does not look right, rewrite it by entering: (↑) (CLEAR) + (NEXT)

CDC Standard Function Keys

The CDC standard function keys for the Viking 721 terminal perform operations that apply to nearly all applications. These operations are assigned to a key or to key combinations on most supported terminals. The keys and their functions are:











Key	Function
(FWD)	Displays the next screen in the file. The last line of the previous screen becomes the first line in the displayed screen.
(BKW)	Moves screen backward one screen. The top line then becomes the bottom line of the displayed screen.
(UP)	Positions the file so that the current line is at the top of the screen.
(DOWN)	Positions the file so that the current line is at the bottom of the screen and the cursor is centered vertically on the screen.
(HELP)	Displays the editor HELP file. The HELP file has brief descriptions of all editor subcommands.
(EDIT)	Returns you to editing one file (the file at the top of the screen) when you are in split screen mode. (EDIT) key can be used to leave the HELP file, returning you to the file you were editing.
(STOP)	Stops a search or replacement in progress.
(NEXT)	Terminates an input line.
(HOME)	Moves the cursor to the subcommand line where you can then enter subcommands.
(DATA)	Stores the current position for later use with the (BACK) key.
(BACK)	Returns you to the position stored with the (DATA) key and stores the current position for later use with the (BACK) key. Repeated entry of the (BACK) key toggles you between two locations.
(⏪) (FWD)	Positions the file so the cursor is at the last line (same as (⏪) F3 (LAST)).
(⏩) (BKW)	Positions the file so the cursor is at the first line (same as F3 (FIRST)).




Programmable Function Keys





You can define the programmable function keys to execute any of the editor subcommands. The labels of the programmable function keys are displayed at the bottom of your screen.

Usually, only the F1 through F8 function key prompts are displayed. The F9 through F16 prompts can be displayed using the `SET_SCREEN_OPTIONS` subcommand described in chapter 7. The lower line of the prompt indicates the unshifted key function; the upper line indicates the shifted key function. On the Viking 721 keyboard, the function key numbers are in raised letters adjacent to the keys. Some programmable function keys also have labels on the key itself. These labels are used by other applications and have no significance within the editor.

The original settings for the Viking 721 programmable function keys follow. (Settings for other terminals are included in chapter 11, Using Other Terminals in Screen Mode.)

Key	Function
F1 	Marks a line or lines to be used by another editor function or subcommand. The marked lines are shown in inverse video on terminals with that capability. When shifted,  F1  unmarks any marked text.
F2 	Marks a character or range of characters to be used by another editor function or subcommand. Marked lines or characters are displayed in inverse video on terminals with that capability. The shifted F2 function, <code>Boxmk</code> , is for a future release.
F3 	Moves the cursor to the first line in the file. When shifted,  F3  moves the cursor to the last line of the file and vertically centers the line on the screen.
F4 	Centers the current line vertically on the screen. When shifted,  F4  positions the cursor to the end of the current line.

Key	Function
F5 UNDO	Cancels any marks you have set and undoes all changes you made to your file since the last time the text was changed.
F6 END	Stops the current editing session. Changes made to any open files are made permanent.
F7 locnxt LOCATE	Prompts you to enter the text you want to locate. When you enter the text and press: (NEXT) the editor locates the text, positioning the cursor at the first character of the text string. When shifted,  F7 locnxt locates the next occurrence of the last text located.
F8 80/132 Locall	Prompts you for a string to find, locates all lines containing the string, and displays them in a directory-type display. To locate all occurrences of the last specified string, press (NEXT) when prompted to enter a search string. When shifted,  F8 80/132 changes the screen format from 80 columns to 132 columns or from 132 columns to 80 columns.
F9 delel insel	Inserts a block of blank lines just before the current line. The number of lines inserted depends on the number of lines displayed on the current file. Two lines of old text are left at the top and bottom of the screen and blanks inserted between. When shifted,  F9 delel deletes empty lines, starting with the current line, until a nonempty line is encountered.

Key	Function
10 delwrd inswrd	<p>Inserts 32 blank spaces in front of the current character. When shifted,</p> <p> F10 deLwrd</p> <p>deletes the current word. If the cursor is positioned on a blank character, blank characters on that line are deleted until a nonblank character is encountered.</p>
11 Break	<p>Breaks the current line into two lines. The line is split just in front of the current cursor position. The cursor is then positioned at the beginning of the second line. If the cursor is positioned at the beginning of a line, a blank line is inserted before the current line.</p>
12 Join	<p>Appends the next line to the end of the current line.</p>
13	<p>Undefined.</p>
14 Move Copy	<p>Copies the lines or characters marked by the MARK or Chrmk function in front of the current cursor position. When shifted,</p> <p> F14 Move</p> <p>moves all marked text in front of the current cursor position, deleting it from its former location.</p>
15 dedent INDENT	<p>Inserts two blank characters in front of all marked lines. When shifted,</p> <p> F15 dedent</p> <p>deletes the first 2 characters of all marked lines.</p>
16	<p>Formats the current paragraph. When shifted,</p> <p> F16 Center</p> <p>centers the current line.</p>

For information on initial programmable function key settings for other supported terminals, refer to chapter 11, Using Other Terminals in Screen Mode.

To redefine the settings of these keys, use the `SET_FUNCTION_KEY` subcommand. This subcommand enables you to create your own set, or sets, of function keys. Format of the subcommand is:

SET_FUNCTION_KEY (SETFK)

NUMBER=integer

COMMAND_STRING=string

SHIFT=boolean

LABEL=string

STATUS=status variable

The **NUMBER** (**NUMBERS** or **N**) parameter specifies the number of the key to be defined. Values can be an integer from 1 through 16. This parameter is required.

The **COMMAND_STRING** (**CS**) parameter specifies a string that contains the subcommand(s) or command(s) to be executed when the specified key is pressed. Values can be any editor subcommand or SCL command. When more than one is specified, separate them with semicolons. This parameter is required.

The **SHIFT** (**S**) parameter specifies whether the key is to be used with the **SHIFT** key. If **SHIFT** is omitted, **FALSE** is assumed.

The **LABEL** (**L**) parameter specifies a string that is to appear as the label on the screen for the specified key. If **LABEL** is omitted, the label becomes the first six characters of **COMMAND_STRING**.

The following are examples of how you might use the `SET_FUNCTION_KEY` subcommand.

```
set_function_key number=4 command_string='dises' label='Status'
```

Defines the F4 key to execute the `DISPLAY_EDITOR_STATUS` subcommand with a screen label of F4 **Status**

```
setfk number=5 command_string='help' shift=true label='help'
```

Defines the shifted F5 key to execute the `HELP` subcommand. The key has a screen label of F5 **help**.

Include a `SET_FUNCTION_KEY` subcommand in your editor prologue file to redefine a function key each time you start the editor. The editor prologue file is described in chapter 10, Prologue and Epilogue Files.

This chapter describes the subcommands that can change many of the system-supplied settings for the editor and check the status of those settings.

Changing the Screen Display	7-1
Setting Screen Options	7-1
Exchanging Screen Widths	7-8
Aligning the Screen	7-9
Changing Tab Settings	7-10
Changing Line Width	7-12
Changing the Verify Option	7-13
Changing the Characters Allowed in a Word	7-14
Changing How Lines Are Listed in Line Mode	7-15
Displaying Status Information	7-16



Changing the Screen Display

There are three subcommands enabling you to change the way the screen appears. These are the SET_SCREEN_OPTIONS, EXCHANGE_SCREEN_WIDTH, and ALIGN_SCREEN subcommands.

Setting Screen Options

The SET_SCREEN_OPTIONS subcommand enables you to change things like the number of lines that are listed on your screen, the number of files you can display at one time, and the number of columns displayed. Format for the subcommand is:

SET_SCREEN_OPTIONS (SETSO)

MODEL=name
COLUMN=integer
MENU_ROW=integer
SPLIT=integer
STATUS=status variable

For all omitted parameters, the editor assumes you want to use the same values you used the last time you entered the SET_SCREEN_OPTIONS subcommand.

The MODEL (M) parameter specifies the terminal type you are using. Valid entries are:

Entry	Terminal
CDC721	CDC Viking 721
CDC722	CDC 722
CDC722_30	CDC 722-30
VT100	DEC VT100
PC_CONNECT	IBM PC (or equivalent)
Z19	Zenith Z19 or Heathkit H19
Z29	Zenith Z29

If the MODEL parameter has not been specified on an earlier subcommand of the editing session, or by a SET_TERMINAL_ATTRIBUTES TRM=name command previous to the editing session, it is required.

The COLUMN (COLUMNS or C) parameter specifies the number of columns to be displayed. Values can range from 1 to the maximum number allowed on your terminal. When first entering the editor, it assumes a value of 80 columns. If COLUMN is omitted, the number of columns displayed remains the same.

The MENU_ROW (MENU_ROWS or MR) parameter specifies the number of rows of function key prompts to display. Values can be:

Value	Meaning
0	Displays no function key prompts.
1	Displays one line of prompts (F1 through F8).
2	Displays two lines of prompts (F1 through F16).

If MENU_ROW is omitted, the number of rows displayed remains the same. When starting the editor, a value of 1 is assumed.

The SPLIT (SPLITS or S) parameter specifies the number of areas of text (splits) you want displayed on the screen. (The screen is divided horizontally to show more than one file.) This number determines how many files you can display at the same time. Values can be numbers from 1 through 16. When entering the editor, the assumed value is 1. The size of the splits is determined by the integer you specify, each split using an equal number of lines. If SPLIT is omitted, the number of splits remains the same.

Pressing the **(EDIT)** key returns you to editing one file (SPLIT=1) when you are in split screen mode.

The following example shows how you might use the SET_SCREEN_OPTIONS subcommand.

In this example, you've already used the SET_SCREEN_OPTIONS and EDIT_FILE subcommands to display files GROUCHO and HARPO and you are editing them on a Viking 721 terminal:

```

File: GROUCHO Lines 1 Thru 6 Size 6
-----
This is file GROUCHO.

It also is an example file.

It can also contain any type of file information you want.

File: HARPO Lines 1 Thru 10 Size 10
-----
This is file HARPO.
█
Line 1
Line 2
Line 3
Line 4
Line 5

This is still file HARPO.

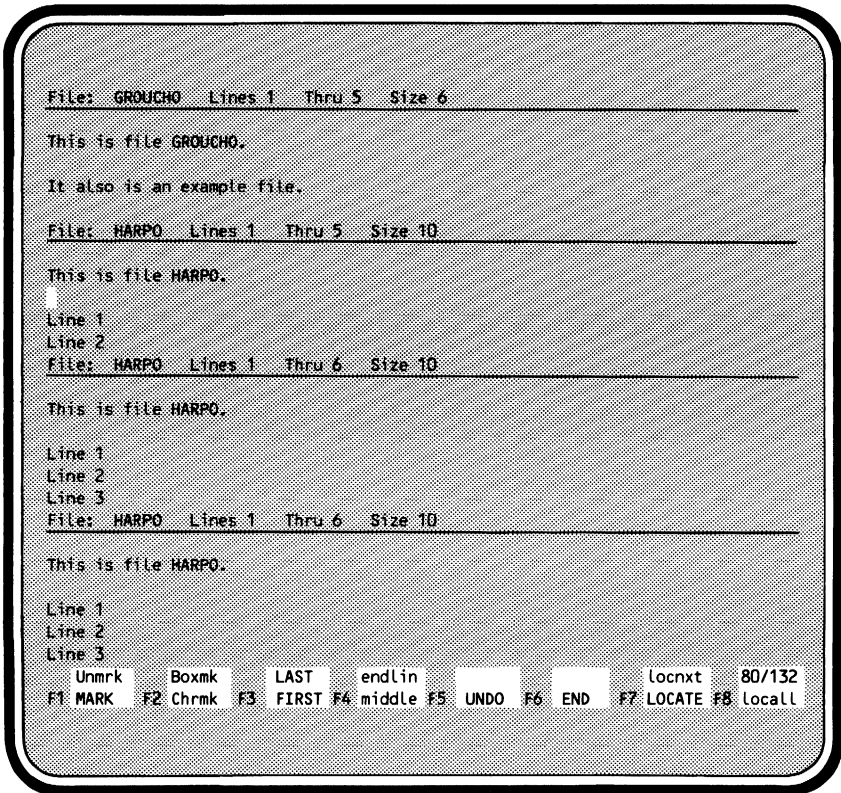
Unmrk   Boxmk   LAST   endlin   █   locnxt   80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO  F6 END   F7 LOCATE F8 LocalL

```

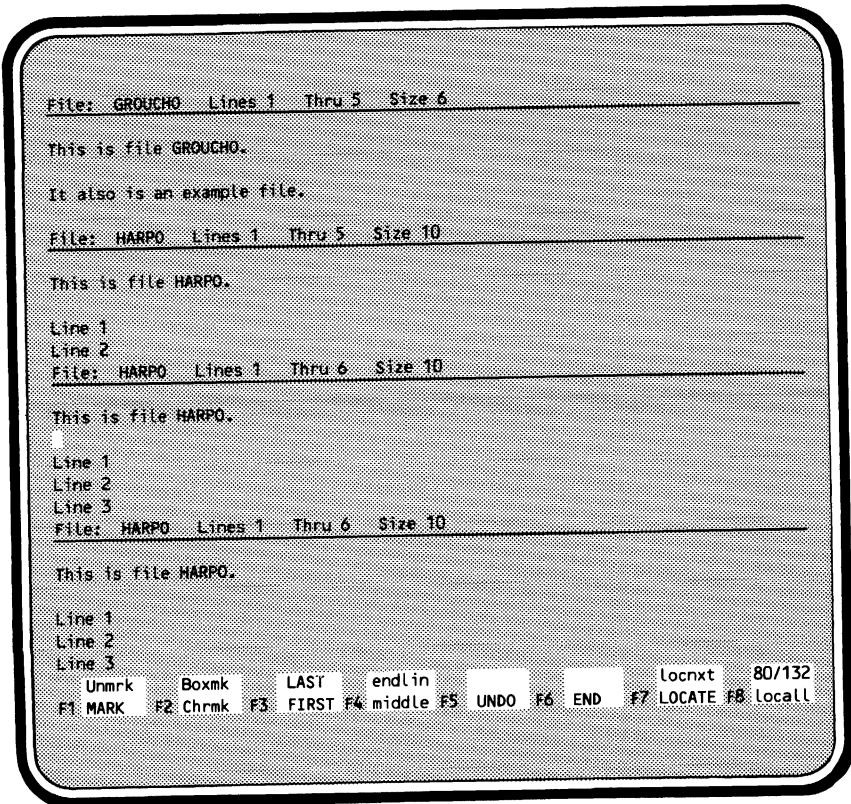
You then want to display files CHICO and ZEPP0 at the same time. To do this, press **(HOME)** and enter the following SET_SCREEN_OPTIONS subcommand.

```
set_screen_options splits=4
```

This adds two more text areas (splits) to the screen. The text from the current file, in this example HARPO, is displayed in the two new split areas:



You then move the cursor to the split in which you want file CHICO to appear. In this example, the third split:

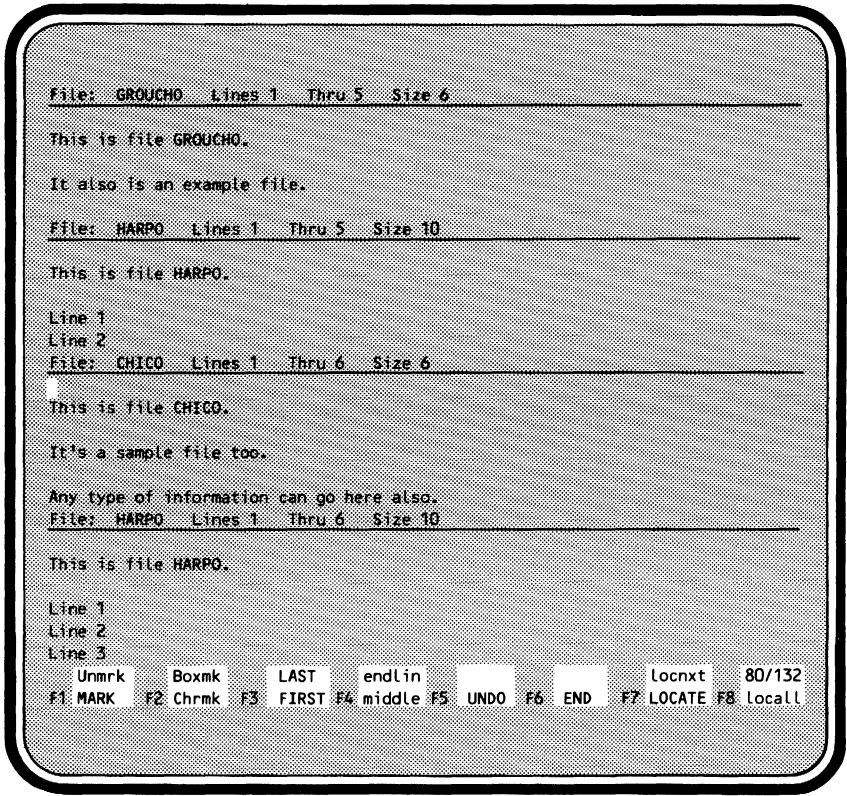


CHANGING THE SCREEN DISPLAY

Then, press **HOME** and enter:

edif chico

File CHICO appears in the third split area:



Then position the cursor in the fourth split, press **HOME** and enter:

```
edif zeppo
```

File ZEPPO appears:

```
File: GROUCHO Lines 1 Thru 5 Size 6
-----
This is file GROUCHO.

It also is an example file.

File: HARPO Lines 1 Thru 5 Size 10
-----
This is file HARPO.

Line 1
Line 2
File: CHICO Lines 1 Thru 6 Size 6
-----
This is file CHICO.

It's a sample file too.

Any type of information can go here also.
File: ZEPPO Lines 1 Thru 6 Size 6
-----
This is file ZEPPO.

It is the fourth example file on this screen.

Use the EDIT key to remove this file from the screen.
  Unmrk  Boxmk  LAST  endlin  locnxt  80/132
F1 MARK F2 Chrmk F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 locall
```

Exchanging Screen Widths

The `EXCHANGE_SCREEN_WIDTH` subcommand allows you to change the width of the screen, alternating between the 80- and 132- column screen displays, for those terminals that support them. Format of the subcommand is:

EXCHANGE_SCREEN_WIDTH (EXCSW)
STATUS=status variable

When entered, `EXCHANGE_SCREEN_WIDTH` changes the screen width to whichever column screen is not being used. For example, if you are using an 80-column screen, entering `EXCSW` changes it to a 132-column screen.

Aligning the Screen

The `ALIGN_SCREEN` subcommand enables you to change the alignment of your screen. Generally, the screen is not realigned unless the current position of the cursor is no longer within the bounds of what is on the screen. When this happens, the screen is realigned and repainted with the current line at the middle of the screen. Format of the subcommand is:

ALIGN_SCREEN (ALIS or A)
MIDDLE=integer or keyword value
TOP=integer or keyword value
BOTTOM=integer or keyword value
OFFSET=integer
STATUS=status variable

The **MIDDLE (M)** parameter specifies a line to be centered vertically on the screen. Values can be an integer or any of the **LINE** keyword values described in the Common Parameters section of chapter 4 with the exception of **ALL**, **MARK**, and **SCREEN**. You cannot use this parameter with the **TOP** or **BOTTOM** parameters. If **MIDDLE** is omitted, **CURRENT** is assumed.

The **TOP (T)** parameter specifies a line to be positioned at the top of the screen. The resulting middle line of the screen becomes the current line. Values can be an integer or any of the **LINE** keyword values described in the Common Parameters section of chapter 4 with the exception of **ALL**, **MARK**, and **SCREEN**. You cannot use this parameter if you have specified a **MIDDLE** or **BOTTOM** parameter. If **TOP** is omitted, a value is not supplied.

The **BOTTOM (B)** parameter specifies a line to appear at the bottom of the screen. The resulting middle line of the screen becomes the current line. Values can be an integer or any of the **LINE** keyword values described in the Common Parameters section of chapter 4 with the exception of **ALL**, **MARK**, and **SCREEN**. You cannot use this parameter with the **TOP** or **MIDDLE** parameters. If **BOTTOM** is omitted, no value is supplied.



The **OFFSET (O)** parameter specifies the number of columns to offset your view of the file on the screen. Values can be integers from 0 through 255. When an offset is specified, the number you specify is added to column 1 and the last column displayed. For example, if the rightmost column is 80, and you specify an **OFFSET** value of 20, the leftmost column becomes 21 and the rightmost column becomes 100. The following examples show how you might use the **ALIGN_SCREEN** subcommand.

```
align_screen top=current
```

Moves the current line to the top of the screen (same as the **UP** key).

```
align_screen bottom=current
```

Move the current line to the bottom of the screen (same as the **DOWN** key).

```
alis offset=50
```

Displays column 51 as the leftmost column.

Changing Tab Settings

There are two subcommands designed specifically to change tab settings (to view which tab columns are currently set, use the `DISPLAY_EDITOR_STATUS` subcommand described later). These are the `SET_TAB_OPTIONS` and `CLEAR_TAB` subcommands. When editing SCU libraries, tab settings can be saved for each deck on the individual deck headers.

The `SET_TAB_OPTIONS` subcommand sets a tab character and the columns in which you want tabs set. Format of the subcommand is:

```
SET_TAB_OPTIONS (SETTO)
  CHARACTER=string
  TAB_COLUMN=list of integer
  STATUS=status variable
```

The `CHARACTER (C)` parameter specifies the tab character. When you enter a tab character within text typed from your terminal, the tab character moves any text from the current position to the next tab setting. If you enter a tab character after the last tab column, the tab character is included as part of the file text. Values can be any printable character. The horizontal tab character, `chr$(9)`, works well as a value. When you start editing a file, the tab character is set to the reverse slant (`\`). When you start editing a deck, the tab character is set as specified in the deck header (refer to the `CREATE_DECK` SCU subcommand in the SCL Source Code Management manual). If `CHARACTER` is omitted, the tab character is not changed.

The `CHARACTER` parameter is useful for line-mode terminals or for screen-mode terminals that do not have a dedicated tab key. The CDC 722 is such a terminal. For terminals that do have a tab key, you will find it more useful than `SET_TAB_OPTIONS C=chr$(9)` since a dedicated tab key moves the cursor instantly, while the interpretation of `SETTO C=chr$(9)` cannot be seen until later.

The `TAB_COLUMN (TAB_COLUMNS or TC)` parameter specifies tab columns to be added to those already selected. A maximum of 256 columns can be specified as tab columns. Values can be any integer from 1 through 256 and must be enclosed in parentheses. To specify more than one column, enclose the values in parentheses, and separate the values with commas (,) or spaces. When you start editing a file, the tab columns selected are 1, 7, and 72. When you start editing a deck, the tab columns selected are those specified in the deck header (refer to the `CREATE_DECK` SCU subcommand in the SCL Source Code Management manual). If `TAB_COLUMN` is omitted, the tab settings are not changed.

The following examples show how you might use the `SET_TAB_OPTIONS` subcommand.

```
set_tab_options character=']' tab_column=(11,18,41,53)
```

Sets the tab character to] and adds columns 11, 18, 41, and 53 as tab columns.

```
set_tab_options character='!' tab_column=(3)
```

Sets the tab character to ! and adds column 3 as a tab column.

To delete all or some of the tab columns, use the `CLEAR_TAB` subcommand. Format of the subcommand is:

```
CLEAR_TAB (CLEAR_TABS or CLET)  
TAB_COLUMN=keyword value or list of range of integer  
STATUS=status variable
```

The `TAB_COLUMN` (`TAB_COLUMNS` or `TC`) parameter specifies the columns to delete as tab columns. Values can be the keyword `ALL` or a list of integers from 1 through 256. Values can also be specified as a range, like 10..60. If `TAB_COLUMN` is omitted, all tabs are cleared.

For example, the following `CLEAR_TAB` subcommand clears columns 7 and 65 as tab columns:

```
clear_tab tab_column=(7,65)
```

Changing Line Width

You can have the editor send you a message when lines in your file exceed a specified limit. Doing this also causes the editor to add trailing spaces to lines with a character count less than the limit when you do string comparisons. Use the `SET_LINE_WIDTH` subcommand to set the limit. Format of the subcommand is:

```
SET_LINE_WIDTH (SETLW)
  WIDTH=integer
  STATUS=status variable
```

The `WIDTH (W)` parameter specifies the number of characters you can have on a line before a message is sent. Values can be integers from 0 through 256. Specifying a value of 0 eliminates the message and adds no trailing blanks to lines. When you create a file, an initial value of 0 is assumed; for decks, the value is taken from the deck header information. This parameter is required.

For example, if you press

```
(HOME)
```

and enter:

```
set_line_width width=80
```

Then you add a line that is 81 characters long. The message

```
--WARNING ES 510530--Line longer than current width.
```

appears. You can then choose to change the line or leave it. Once you have set the line width, you can locate long lines by using the `LOCATE_WIDE_LINES` subcommand. The line width marker only remains in effect throughout an editing session. Each time you edit the file, you must enter the `SET_LINE_WIDTH` subcommand to change the line width and be warned when lines are too long.

Changing the Verify Option

The verify option displays lines that have been changed via the REPLACE_TEXT subcommand and also displays the first and last lines of a block of text located with the LOCATE_TEXT subcommand. Format of the subcommand is:

```
SET_VERIFY_OPTION (SETVO)  
ECHO=boolean  
STATUS=status variable
```

The ECHO (E) parameter specifies if you want the verify option on or off. When you start the editor, the system sets the verify option to TRUE. Therefore in line mode, the verify option is on unless you specify ECHO=FALSE on a SET_VERIFY_OPTION subcommand. In screen mode, the verify option is always off. This parameter is required.

Changing the Characters Allowed in a Word

If you want to use the WORD parameters on subcommands which allow it and need to use characters within those words other than alphanumeric characters, use the SET_WORD_CHARACTERS subcommand to add or delete allowable characters. When starting the editor, allowable characters are any alphanumeric character plus the special characters @, #, \$, and _ . The format of the subcommand is:

SET_WORD_CHARACTERS (SETWC)

ADD=list of string

DELETE=list of string

STATUS=status variable

The ADD (A) parameter specifies the characters to add as allowable characters. Values can be any printable character. If you specify more than one character, separate them with commas (,) or spaces. If you add the comma as an allowable character, separate the characters with spaces. Enclose the values in parentheses and quotes. The space character cannot be specified as an allowable character. If ADD is omitted, no characters are added.

The DELETE (D) parameter specifies the characters to delete as allowable characters in a word. In other words, characters specified by this parameter will be treated as punctuation marks. If you specify more than one character, separate them with commas (,) or spaces. Enclose the values in parentheses and quotes. Values can be any printable character. The space character is not allowed. If DELETE is omitted, no characters are deleted.

The following examples show how you might use the SET_WORD_CHARACTERS subcommand.

```
set_word_characters add=('&', '|', '%')
```

Adds the characters &, |, and % as characters allowed in words.

```
set_word_characters add=('%' ) delete=('x')
```

Adds % as an allowable word character and deletes x as an allowable word character.

```
set_word_characters add('-') delete=(' '$' '#' '_' 'a')
```

Changes the characters allowed in words to those used in the NOS/VE COBOL compiler.

Changing How Lines Are Listed in Line Mode

The `SET_LIST_OPTIONS` subcommand determines whether line identifiers are displayed in line mode. You would usually use this subcommand when you are editing decks and want to see the line identifiers. The editor does, however, add line identifiers to all files. The format of the subcommand is:

```
SET_LIST_OPTIONS (SETLO)  
LINE_IDENTIFIER=keyword value  
STATE=boolean  
STATUS=status variable
```

The `LINE_IDENTIFIER` (LI) parameter specifies where or if the identifier is to be displayed. Values can be `LEFT` (L), `SEPARATE` (S), or `NONE` (N). If you select `LEFT`, the line identifier appears on the same line as the text itself. If you specify `SEPARATE`, the identifier is displayed before the line of text as a separate line. If `NONE` is specified, the identifier is not displayed at all. If `LINE_IDENTIFIER` is omitted, `NONE` is assumed.

The `STATE` (S) parameter specifies whether the state of the modification associated with the line's introduction is to be displayed. Modification states are described in the SCL Source Code Management manual. When the value is `TRUE`, the state is displayed. When the value is `FALSE`, the state is not displayed. If `STATE` is omitted, `FALSE` is assumed.

Displaying Status Information

Three subcommands enable you to display information about the current status of the editor. The subcommands are `DISPLAY_POSITION`, `DISPLAY_EDITOR_STATUS`, and `DISPLAY_COLUMN_NUMBERS`.

The `DISPLAY_POSITION` subcommand displays the current line number, current column number, size of the file, and the line number of the top and bottom line of the screen on the message line. The format is:

DISPLAY_POSITION (DISP)
STATUS=status variable

For example, if you enter

```
display_position
```

you might get the following display:

```
Current Line: 12 Column: 10 Size: 109
```

The `DISPLAY_EDITOR_STATUS` subcommand enables you to check the status of a number of editor variables including the current tab character, tab columns, function key definitions, and so on. Format of the subcommand is:

DISPLAY_EDITOR_STATUS (DISES)
STATUS=status variable

Following is the type of display you get when you enter `DISPLAY_EDITOR_STATUS` in screen mode.

```
Press Next/Return for more
Displaying Editor Status
```

```
SDU Editor version is 85037 †
Modification name is EDIT_FILE
Current file is: $LOCAL.ZAP
Line width is 0. Search margins are 1 to 256
Set verify option FALSE. State FALSE. No mask character. Tab character is J.
Tab columns are: 1 5 10 15 50 72
```

Function Keys:

Key	Label	Commands
F1	MARK	mark_lines
Shift F1	Unmk	unmark
F2	Chrmk	mark_characters
Shift F2	Boxmk	mark_boxes
F3	FIRST	align_screen top=first
Shift F3	LAST	align_screen middle=last
F4	middle	align_screen middle=current
Shift F4	endlin	position_cursor (=c c\$strlen(\$it)+1
F5	UNDO	undo
F6	END	end
F7	LOCATE	locate_text t=\$screen_input('Enter search string')
Shift F7	locnxt	position_cursor; position_cursor rs=true
F8	locall	esv\$text=\$screen_input('Enter search string'); if esv\$text
Shift F8	80/132	exchange_screen_width
F9	insel	insert_empty_lines p=b n=\$split_size-4; position_cursor d
Unmk	Boxmk	LAST endlin locnxt 80/132
F1 MARK	F2 Chrmk	F3 FIRST F4 middle F5 UNDO F6 END F7 LOCATE F8 locall

To list the column numbers, use the `DISPLAY_COLUMN_NUMBERS` subcommand. The column numbers appear over the current line. The format of the subcommand is:

`DISPLAY_COLUMN_NUMBERS (DISCN)`

ROW=integer

STATUS=status variable

The `ROW (R)` parameter specifies which row on the screen is to show the column numbers. Values can be an integer from 1 through 255. In line mode, this parameter is ignored. If `ROW` is omitted, the column numbers are displayed over the current line.

† If you submit a Programming System Report (PSR) on the editor, please include the editor's version number.

DISPLAYING STATUS INFORMATION

For example, to list the column numbers of the third line

```
FIRST LINE
SECOND LINE
THIRD LINE
FOURTH LINE
```

position the cursor on the line:

```
FIRST LINE
SECOND LINE
|THIRD LINE
FOURTH LINE
```

Press:

HOME

and enter

```
display_column_numbers
```

The following appears:

```
FIRST LINE
SECOND LINE
123456789A123456789B123456789C123456789D123456789E123456789F123456789G123456789H
FOURTH LINE
```

The column numbers shown correspond to columns in the file and not column numbers on the screen.

Starting the Editor	8-2
Opening Decks	8-4
Opening a Deck While Maintaining Other Decks	8-5
Editing a Specific Deck	8-5
Editing the First Deck on the Library	8-5
Editing the Last Deck on the Library	8-6
Editing the Next Deck on the Library	8-6
Opening a Deck While Closing the Previous Deck	8-6
Editing a Specific Deck	8-7
Editing the First Deck on the Library	8-7
Editing the Last Deck on the Library	8-7
Editing the Next Deck on the Library	8-7
Discarding Deck Changes	8-8
Closing a Deck	8-10
Stopping the Editor	8-11



Within the Source Code Utility (SCU) you can use the Full Screen Editor to make changes to decks. Besides using the full range of features available to you during file editing sessions, while editing decks you have an extra set of subcommands designed specifically for editing decks:

EDIT_DECK	SELECT_FIRST_DECK
EDIT_FIRST_DECK	SELECT_LAST_DECK
EDIT_LAST_DECK	SELECT_NEXT_DECK
EDIT_NEXT_DECK	RESET_DECK
SELECT_DECK	END_DECK

As with files, you can work on more than one deck during an EDIT_LIBRARY session. Also if your terminal has screen handling capabilities, you can activate the screen and edit decks in screen mode.

NOTE

When editing, you can modify only one deck at a time. For example, a REPLACE_TEXT subcommand can affect only the current deck; if you want to make the same change in other decks, you must switch to each deck and repeat the REPLACE_TEXT subcommand.

Once you have entered the editor using the SCU subcommand EDIT_LIBRARY, SCU restricts your use of other SCU capabilities to those that do not make changes to the library. The following SCU subcommands are allowed:

CREATE_MODIFICATION	DISPLAY_GROUP
DISPLAY_DECK	DISPLAY_GROUP_LIST
DISPLAY_DECK_LIST	DISPLAY_LIBRARY
DISPLAY_DECK_REFERENCES	DISPLAY_MODIFICATION
DISPLAY_FEATURE	DISPLAY_MODIFICATION_LIST
DISPLAY_FEATURE_LIST	EXTRACT_MODIFICATION

When you have completed editing (by entering END or QUIT), all SCU subcommands are again accessible.

Starting the Editor

You must be in SCU to edit SCU decks. Enter the utility using the SCL command described in the SCL Source Code Management manual. Its format is:

SOURCE_CODE_UTILITY

BASE=file
RESULT=file
LIST=file
STATUS=status variable

You then enter the SCU subcommand `EDIT_LIBRARY` to begin an editing session. The format of the subcommand is:

EDIT_LIBRARY

MODIFICATION=name
INPUT=file
OUTPUT=file
DECK=name
CONTINUE=boolean
PROLOG=file
STATUS=status variable

The **MODIFICATION (M)** parameter specifies the name of the modification to which changes made during the editor session belong. If the modification does not already exist, you must also specify `CONTINUE=FALSE`. This parameter is required.

The **INPUT (I)** parameter specifies the file used as input to the editor. This file can be positioned. If omitted, file `$COMMAND` is assumed. File `$COMMAND` is usually connected to terminal input.

The **OUTPUT (O)** parameter specifies the file to which the edit display is written. This file can be positioned. If `OUTPUT` is omitted, file `$OUTPUT` is assumed.

The **DECK (D)** parameter specifies the name of the deck to be edited first. If you do not specify the first deck, you must enter an editor deck selection subcommand before entering commands to change text.

The **CONTINUE (C)** parameter indicates whether the editor session continues an existing modification or begins a new modification. When you specify `TRUE`, an existing modification is continued. When you specify `FALSE`, the editor begins a new modification. If `CONTINUE` is omitted, `TRUE` is assumed.

The PROLOG (P) parameter specifies the name of the file the system executes when you start an editor session. This file can be positioned. If PROLOG is omitted, file \$USER.SCU_EDITOR_PROLOG is assumed. Chapter 10, Prologue and Epilogue Files, describes the prologue file in more detail.

For example, to edit deck LOG_CHANGES on the permanent source file named OLDPL using the existing modification LOG_MOD1, and write the resulting library on the next cycle of OLDPL, enter:

```
/scu base=$user.oldpl result=$user.oldpl.$next
sc/edit_library modification=log_mod1 deck=log_changes
sce/
```

You can now use any of the capabilities of the editor including the deck editor subcommands listed earlier in this chapter; the EDIT_LIBRARY SCU subcommand adds an entry containing the editor subcommands to the NOS/VE subcommand list. The name of the entry is SCU_EDIT.

You can edit decks in either screen or line mode. Whenever you are in line mode, the prompt

```
sce/
```

is displayed when the editor is ready for input.

The following is the header written on the output file if the EDIT_LIBRARY SCU subcommand is entered in batch mode.

```
NOS/VE SOURCE CODE UTILITY V1.1 84151 1985-03-22 13:21:53 PAGE 1
EDITOR
BASE=$USER.OLDPL
```

Opening Decks

You can specify the first deck you want to edit on either the SCU subcommand `EDIT_LIBRARY` or on one of the editor subcommands that edit or select decks. The `EDIT_LIBRARY` subcommand is described in the previous section, *Starting the Editor*; a discussion of using the edit and select deck subcommands follows.

Once you are in the editor, you can choose to either edit or select a deck. Either method allows you to make changes to a deck. When editing additional decks, you can either keep the first deck open while editing a new deck or you can close the first deck when you start editing a new deck. If you choose to keep the first deck open, that deck is put in the background. There it is not affected by editing changes you make to the new deck, but it is maintained in the same state you left it when you changed decks.

The deck subcommands that start with `EDIT`, open the deck chosen and leave other decks open and in the background. The decks in the background can be edited again by entering another deck subcommand. You can use the edit deck subcommands in combination with other subcommands like `SET_SCREEN_OPTIONS` and `COPY_TEXT` to transfer lines from one deck to another and to do other editing functions. The edit deck subcommands allow you to have many decks easily accessible during one editing session. Although this is a nice feature, you may want to consider the amount of system resources you are using when you do this. With each additional deck, the resources you are using increases.

The deck subcommands that start with `SELECT`, open the deck chosen, but at the same time, close the deck you were editing right before you entered the subcommand (if any). The changes you made to the previous deck are written to the working library. Refer to the *SCL Source Code Management manual* if you would like more information about working libraries. You can open the previous deck again later in the editing session by entering another edit or select deck subcommand.

While you are editing decks (having entered the `EDIT_LIBRARY` subcommand), you can also edit files. Use the `EDIT_FILE` subcommand to edit files. Any deck that is open when you enter `EDIT_FILE` remains open while you are editing a file. Entering `END` or `QUIT` closes all files as well as all decks. To edit a deck after editing a file enter one of the edit or select deck subcommands.

Opening a Deck While Maintaining Other Decks

The following subcommands start the editing process on a deck while other decks you had previously edited in this session remain available in the background:

```
EDIT_DECK          EDIT_LAST_DECK
EDIT_FIRST_DECK   EDIT_NEXT_DECK
```

Their individual descriptions follow.

Editing a Specific Deck

The `EDIT_DECK` subcommand opens the specified deck on the working library for editing while maintaining your current position in other decks. The format of the subcommand is:

```
EDIT_DECK (EDID)
  DECK=name
  STATUS=status variable
```

The `DECK (D)` parameter specifies the name of the deck to be edited. This parameter is required.

For example, to edit a deck named `MY_DECK` enter:

```
edit_deck deck=my_deck
```

Editing the First Deck on the Library

The `EDIT_FIRST_DECK` subcommand opens the first deck on the working library for editing while maintaining your current position in other decks. The format of the subcommand is:

```
EDIT_FIRST_DECK (EDIFD)
  STATUS=status variable
```

Editing the Last Deck on the Library

The `EDIT_LAST_DECK` subcommand opens the last deck on the working library for editing while maintaining your current position in other decks. The format of the subcommand is:

EDIT_LAST_DECK (EDILD)
STATUS=status variable

Editing the Next Deck on the Library

The `EDIT_NEXT_DECK` subcommand opens the next deck on the working library for editing while maintaining your current position in other decks. The format of the subcommand is:

EDIT_NEXT_DECK (EDIND)
STATUS=status variable

Opening a Deck While Closing the Previous Deck

The following subcommands start the editing process on a deck and at the same time close the last deck you were editing, if any (changes made to the previous deck are written to the working library):

`SELECT_DECK` `SELECT_LAST_DECK`
`SELECT_FIRST_DECK` `SELECT_NEXT_DECK`

Their individual descriptions follow.

Editing a Specific Deck

The `SELECT_DECK` subcommand opens the specified deck on the working library for editing and closes the previous deck (if any). The format of the subcommand is:

```
SELECT_DECK (SELD)  
DECK=name  
STATUS=status variable
```

The `DECK (D)` parameter specifies the name of the deck to be edited. This parameter is required.

For example, to edit a deck named `YOUR_DECK` enter:

```
select_deck deck=your_deck
```

Editing the First Deck on the Library

The `SELECT_FIRST_DECK` subcommand opens the first deck on the working library for editing and closes the previous deck (if any). The format of the subcommand is:

```
SELECT_FIRST_DECK (SELFD)  
STATUS=status variable
```

Editing the Last Deck on the Library

The `SELECT_LAST_DECK` subcommand opens the last deck on the working library for editing and closes the previous deck (if any). The format of the subcommand is:

```
SELECT_LAST_DECK (SELLD)  
STATUS=status variable
```

Editing the Next Deck on the Library

The `SELECT_NEXT_DECK` subcommand opens the next deck on the working library for editing and closes the previous deck (if any). The format of the subcommand is:

```
SELECT_NEXT_DECK (SELND)  
STATUS=status variable
```

Discarding Deck Changes

To discard changes made to the current deck being edited, use the `RESET_DECK` subcommand. All changes made since the last time the deck was opened for editing are discarded; the editor obtains a new copy of the deck from the working library. The format of the subcommand is:

RESET_DECK (RESD)
STATUS=status variable

For example, consider the following sequence of line mode entries:

Entries	Description
<code>sc/edit_library m=mod3 c=false</code>	Begin an editor session within SCU.
<code>sce/edit_deck deck=deck_three</code>	Edit <code>DECK_THREE</code> (open first time).
Begin editing deck <code>DECK_THREE</code> .	
<code>sce/</code>	
.	Enter first set of changes for <code>DECK_THREE</code> .
.	
.	
<code>sce/select_deck deck=deck_five</code>	Select <code>DECK_FIVE</code> , closing <code>DECK_THREE</code>
Begin editing deck <code>DECK_FIVE</code> .	(<code>DECK_THREE</code> is rewritten on the
<code>sce/</code>	working library).
.	
.	Enter set of changes for <code>DECK_FIVE</code> .
.	
<code>sce/edit_deck deck=deck_three</code>	Edit <code>DECK_THREE</code> (open second time).
Begin editing deck <code>DECK_THREE</code> .	<code>DECK_FIVE</code> remains open.
<code>sce/</code>	
.	Enter second set of changes for <code>DECK_THREE</code> .
.	
.	
<code>sce/reset_deck</code>	Delete the second set of changes for
Begin editing deck <code>DECK_THREE</code> .	<code>DECK_THREE</code> . The first set of changes
	remains since <code>DECK_THREE</code> was closed
	after they were made. <code>DECK_FIVE</code> is unchanged.

You can also undo changes in decks as you would do in files (refer to Undoing in chapter 5).

Once you enter `END_DECK` or a select deck subcommand, a deck is closed. Changes made to the closed deck are copied to the working library and therefore cannot be discarded or undone.

The `RESET_DECK` subcommand performs a function similar to the `RESET_FILE` subcommand. `RESET_DECK` nullifies changes as does `RESET_FILE`. It nullifies them, however, by deleting the deck and replacing it with a copy from the working library. When you enter `RESET_FILE`, the editor actually retraces its steps and undoes changes in reverse order.

Closing a Deck

To close editing on the current deck, use the `END_DECK` subcommand. Whether changes you have made are written to the working library is determined by the `WRITE_DECK` parameter. The `END_DECK` and `END_FILE` subcommands perform the same function. Either can be used to close a deck or file.

The format of the `END_DECK` subcommand is:

```
END_DECK (ENDD)  
WRITE_DECK=boolean  
STATUS=status variable
```

The `WRITE_DECK` (`WD`, `WRITE_FILE`, `WF`) parameter specifies whether the changes made to the deck since it was opened for editing are to be written to the working library. A value of `TRUE` indicates that the deck is to be rewritten; a value of `FALSE` indicates that the deck remains unchanged (the edited copy is discarded). If `WRITE_DECK` is omitted, `TRUE` is assumed and the results are written to the working library.

Stopping the Editor

All decks are automatically closed when you leave the editor. To leave the editor you can either use a function key (screen mode only), or use the END or QUIT subcommand.

The function key that stops the editor is:

F6 **END**

Changes made to all open decks are written to the working library and changes made to all files are made permanent.

The END and QUIT subcommands stop the editor. They do exactly the same thing and they have the same format. The format is:

END or **QUIT (QUI)**

WRITE_DECK=boolean
STATUS=status variable

The WRITE_DECK (WD, WRITE_FILE, WF) parameter specifies if you want changes to all open decks or files made permanent. When you specify TRUE, changes made to all open decks are written to the working library and changes made to all files are made permanent. When you specify FALSE, changes to all open decks and files are canceled. If WRITE_DECK is omitted, TRUE is assumed.

For example, if you have edited decks SUB1, SUB2, and SUB3, are now editing deck MAIN, and enter:

```
end_deck write_deck=true
```

deck MAIN is rewritten on the working library (preserving any changes made).

You then end the editing session by entering:

```
quit write_deck=false
```

The changes you made to decks SUB1, SUB2, and SUB3 do not become part of the working library.

When you end the editing session you are still in SCU. You end the SCU session by entering:

```
sc/quit write_library=true
```

SCU generates a result library. The changes made to the working library (those made to deck MAIN) become part of the result library. Other changes made to decks SUB1, SUB2, and SUB3 do not become part of the result library.



To further enhance the power of the editor, you can create procedures which execute editor subcommands, SCL commands, and so on. This chapter describes how to create and run procedures and how to call the editor from within a procedure.

- Structure 9-1
 - Procedure Header 9-2
 - Statement List 9-2
 - Procedure End 9-3
- Subcommands and Functions 9-4
 - Subcommand 9-4
 - Functions 9-5
- Examples 9-11
- Calling the Editor from Within a Procedure 9-13



This chapter describes how to create procedures to enhance the power and usability of the editor. In a procedure you can combine editor subcommands, SCL commands, and special SCL functions to perform sophisticated editing functions. This chapter describes:

- The structure of procedures.
- Special editor subcommands and SCL functions designed to be used within editor procedures.
- Examples of procedures.
- How to call the editor from within a procedure.

Structure

Procedures can contain any editor subcommand, SCL command, or SCL function; you are not limited to only editor subcommands. This allows you to have access to the power and versatility of SCL from within the editor.

The structure of a procedure for the editor is the same as an SCL procedure:

```
procedure header  
  statement list  
procedure end
```

This section briefly describes the three components of a procedure. For more detailed information, refer to the SCL Language Definition manual.

Procedure Header

The procedure header is a line giving the procedure a name. The format of the procedure header is:

PROC procedure names

For example, a procedure named `TERMINAL_SETUP` could have the following procedure header:

```
proc terminal_setup
```

The name of the procedure can be any valid SCL name. On the procedure header you can also add any other names by which you may want to reference the procedure. For example:

```
proc terminal_setup,terms,t
```

With this procedure header, you could call the procedure using any of the following:

```
terminal_setup
terms
t
```

Statement List

The statement list is a list of the commands and subcommands that you want the procedure to execute. For example, the `TERMINAL_SETUP` procedure could contain the following editor subcommands:

```
setso mr=2
setfk n=5 s=true cs='setso mr=1'
```

These subcommands tell the editor to display two rows of function key prompts and set the shifted function key 5 to display only one row.

Procedure End

The procedure end statement tells the editor that the procedure is over. The procedure end statement can appear as:

```
PROCEND
```

or

```
PROCEND procedure name
```

For example, `TERMINAL_SETUP` might look like:

```
PROC terminal_setup,terms,t
setso mr=2
setfk n=5 s=true cs='setso mr=1'
PROCEND terminal_setup
```

To make your procedure more readable, you probably want to use indentation. For example,

```
PROC terminal_setup,terms,t
  setso mr=2
  setfk n=5 s=true cs='setso mr=1'
PROCEND terminal_setup
```

Another way to make your procedure more readable is to add comments. You can add comments with quotation marks. For example, to document procedure `TERMINAL_SETUP`, you might add comments like:

```
PROC terminal_setup,terms,t
  setso mr=2           "Display two function key rows."
  setfk n=5 s=true cs='setso mr=1' "Set F5 to display one row."
PROCEND terminal_setup
```

Subcommands and Functions

There is a subcommand and several functions that are designed to be used within editor procedures. This subcommand and the functions are described in this section. The Examples section at the end of this chapter shows how you can use editor subcommands and functions with SCL commands to create very useful procedures.

Subcommand

The `PUT_ROW` subcommand is designed to be used within procedures. Its function is to enable you to print text on any row on the screen. When this subcommand is in procedures, this enables you to display messages on different lines on the screen. Format of the subcommand is:

```
PUT_ROW (PUTR)
  TEXT=string
  ROW=integer
  STATUS=status variable
```

The `TEXT (T)` parameter specifies the text to be printed. This is a text string from 1 through 256 characters. This parameter is required.

The `ROW (ROWS or R)` parameter indicates the row in which the text will be written. Values can be an integer from 1 through the number of rows available on your screen or any of the functions described in the next section that specify a row. For example, to specify the message row, enter `$MESSAGE_ROW`. If `ROW` is omitted, the current line number is assumed.

For example, in a procedure which defines an alternative set of function key definitions for the CDC 722 terminal, you might want to write the message

```
722 keys are set.
```

in the message row. To do so, include the following subcommand in the procedure:

```
put_row text='722 keys are set.' row=$message_row
```


Functions

Functions are phrases that are replaced by values and must be used in a subcommand, an SCL command, or a procedure. If the value returned is a string, you can use the function wherever a string parameter is used. If the value returned is an integer, you can use the function wherever an integer parameter is used, and so on. The functions, listed alphabetically (with the abbreviation shown in parentheses), are:

Function Name	Value that Replaces the Function Name
<code>\$ACTIVE_IDENTIFIER</code> ('line id')† (<code>\$AI</code>)	A line identifier string (for editing decks only). If the line is active, it returns the string entered. If the line is not active, it returns a string, representing the line identifier for the nearest active line. If no lines are active, <code>FIRST</code> is returned.
<code>\$CURRENT_COLUMN</code> (<code>\$CC</code>)	An integer specifying the current column number. If the <code>POSITION_CURSOR</code> subcommand is used to specify a column on a row which is not part of the file text, the value returned is the column at which the cursor was positioned before the <code>POSITION_CURSOR</code> subcommand was entered.
<code>\$CURRENT_DECK</code> (<code>\$CD</code>)	A string specifying the current deck's name (for editing decks only). All letters in the string are uppercase, even if the name was originally entered using lowercase letters.
<code>\$CURRENT_LINE</code> (<code>\$CL</code>)	An integer specifying the current line number.
<code>\$CURRENT_OBJECT</code> (<code>\$CO</code>)	A string identifying the current file name.

† Entries shown in lowercase characters require that you supply a value.

Function Name	Value that Replaces the Function Name
<code>\$CURRENT_OBJECT_TYPE</code> (<code>\$COT</code>)	A string identifying the current object being edited. Possible values are FILE, DECK, or NULL.
<code>\$CURRENT_ROW</code> (<code>\$CR</code>)	An integer identifying the current row on the screen (as opposed to the current line number of a file). Zero is returned if the current row is not within screen boundaries or if you are in line mode.
<code>\$CURRENT_SPLIT</code> (<code>\$CS</code>)	An integer specifying the current split of the screen. Values returned can be from 1 through 16. The top split of the screen is 1, the next lower split is 2, and so on.
<code>\$CURRENT_WORD</code> (<code>\$CW</code>)	The current word as a string.
<code>\$CURRENT_WORD_COLUMN</code> (<code>\$CWC</code>)	An integer specifying the column in which the current word begins.
<code>\$FUNCTION_ROW</code> (<code>\$FR</code>)	An integer specifying the top row in which the function key prompts are displayed.
<code>\$FUNCTION_SIZE</code> (<code>\$FS</code>)	An integer specifying the number of rows used by the function key prompts.
<code>\$HOME_ROW</code> (<code>\$HR</code>)	An integer identifying the row used for entering subcommands.
<code>\$LINE_IDENTIFIER</code> (<code>\$LI</code>)	A string specifying the line identifier of the current line (for decks only).
<code>\$LINE_TEXT</code> (<code>\$LT</code>)	The text of the current line as a string.

Function Name	Value that Replaces the Function Name
\$MARK_FIRST_COLUMN (\$MFC)	An integer specifying the column number of the first marked column.
\$MARK_FIRST_LINE (\$MFL)	An integer specifying the line number of the first marked line.
\$MARK_LAST_COLUMN (\$MLC)	An integer specifying the column number of the last marked column.
\$MARK_LAST_LINE (\$MLL)	An integer specifying the line number of the last marked line.
\$MARK_OBJECT (\$MO)	A string specifying the name of the current file containing the marked text.
\$MARK_OBJECT_TYPE (\$MOT)	A string specifying if the marked text is in a file or deck. Values returned can be FILE, DECK, or NULL.
\$MARK_TYPE (\$MT)	Specifies if the marked region is bounded by lines or characters. Values returned can be LINES (line boundary) or STREAM (character boundary).
\$MESSAGE_ROW (\$MR)	An integer specifying the number of the row on the screen used to display messages.
\$NEW_TEXT (\$NT)	The last string entered as a NEW_TEXT parameter.
\$NUMBER_OF_COLUMNS (\$NOC)	An integer specifying the number of columns currently being used to display text on the screen.
\$NUMBER_OF_ROWS (\$NOR)	An integer specifying the number of rows that now have text on them.
\$NUMBER_OF_SPLITS (\$NOS)	An integer specifying the number of splits on the screen.

Function Name	Value that Replaces the Function Name
\$OFFSET (\$O)	An integer specifying the number specified by the OFFSET parameter of the ALIGN_SCREEN subcommand. If you have not specified the OFFSET parameter, 0 is assumed.
\$PARAGRAPH_MARGINS (keyword value)† (\$PM)	An integer specifying the current left margin setting if you include the keyword value LEFT ; an integer specifying the current right margin setting if you include the keyword value RIGHT ; an integer specifying the current margin offset if you include the keyword value OFFSET .
\$SCREEN_ACTIVE (\$SA)	A boolean value. It is TRUE if screen mode is active; FALSE if it is not.
\$SCREEN_INPUT ('text')† (\$SI)	This function allows an SCL procedure to pause and ask the user for some input. For example, the following command performs a search for whatever text the user provides in response to \$SI .
	<pre>locate_text text=\$SI ('What do you want to search for?')</pre>
	The value that replaces the function is whatever you type on the subcommand line.

† Entries shown in lowercase characters require that you supply a value.

Function Name	Value that Replaces the Function Name
---------------	---------------------------------------

NOTE

When typing your input, type only the text you want. Do not put apostrophes around it or double up any apostrophes within your input.

The text in the function is displayed on the message row as a prompt for your input. If ('text') is omitted, ENTER TEXT is used as the prompt.

`$SEARCH_MARGINS (keyword value)†`
(`$SM`)

An integer specifying the left margin is returned if you include the keyword value **LOW**. An integer specifying the right margin is returned if you include the keyword value **HIGH**.

† Entries shown in lowercase characters require that you supply a value.

Function Name	Value that Replaces the Function Name
\$SPLIT_SIZE (split)† (\$SS)	An integer specifying the number of available text lines for the specified split of the screen. If you do not specify a split number, the current split is assumed.
\$TERMINAL_MODEL (\$TM)	A string specifying the current terminal model. Values returned can be any of the terminal models defined at your site including CDC721, CDC722, CDC722_30, PC_CONNECT, Z19, or Z29.
\$TEXT (\$T)	A string specifying the last text you specified on a TEXT parameter.
\$TITLE_ROW (split)† (\$TR)	An integer specifying the row number of the title row used for the specified split of the screen. If you don't specify a split number, the current split is assumed.
\$UPPER_CASE (\$UC)	A boolean value. It is TRUE if you specified TRUE on the last UPPER_CASE parameter; FALSE if you specified FALSE.
\$VERIFY_OPTION (\$VO)	A boolean value. It is TRUE if you have activated the VERIFY option; FALSE if it is not activated.
\$WORD (\$W)	A boolean value. It is TRUE if the word search feature is active; FALSE if it is not.

† Entries shown in lowercase characters require that you supply a value.

Examples

The procedures in this section are provided to show you how you might use procedures to perform editing functions.

Example 1:

This procedure deletes characters.

```
PROC delete_characters,delete_character,delc,dc (
  number, n : ANY
  lines, line, l : RANGE of ANY
  columns, column, c: RANGE OF ANY
  status)

  WHEN any_fault DO
    EXIT_PROC WITH osv$status
  WHENEND

  create_variable local_status kind=status

  command='delete_text boundary=stream'
  IF $specified(lines) THEN
    command=command//' l='//$parameter(lines)
  ELSE
    command=command//' l=current'
  IFEND
  IF $specified(columns) THEN
    command=command//' c='//$parameter(columns)
  ELSE
    command=command//' c=c'
  IFEND
  IF $specified(number) THEN
    command=command//' n='//$parameter(number)
  IFEND

  include_line command

PROCEND delete_characters
```

EXAMPLES

Example 2:

This procedure deletes empty lines.

```
PROC delete_empty_lines, delete_empty_line, deletel,(
  lines, line, l : RANGE OF ANY
  status)

  WHEN any_fault DO
    set_verify_option verify status=local_status
    EXIT_PROC WITH osv$status
  WHENEND

  create_variable local_status kind=status

  verify=$verify_option
  set_verify_option false status=local_status

  IF $specified(lines) THEN
    position_cursor l=$parameter(lines) c=1
  ELSE
    position_cursor c=1 l=c
  IFEND

  start=$line_identifier

  position_cursor l=last
  id_two=$line_identifier

  position_cursor l=start

  set_verify_option verify

  REPEAT
    id_one=$line_identifier
    IF $line_text='' THEN
      delete_line l=current
    ELSE
      id_one=id_two
    IFEND
  UNTIL id_one=id_two

  position_cursor l=current

PROCEND delete_empty_lines
```


Calling the Editor from Within a Procedure

In order to call the editor from within a procedure, you must first be aware of how the default command stream works. The `EDIT_FILE` command offers an `INPUT` parameter, which defaults to `$COMMAND`. This parameter indicates a file from which editor commands will be read. The special file name `$COMMAND` indicates the current command stream.

When the `EDIT_FILE` command is entered at the terminal, the current command stream is the terminal. Screen mode occurs if it has been set using `ACTIVATE_SCREEN`. When the `EDIT_FILE` is embedded in a procedure or included file, the current command stream consists of the remainder of that procedure or included file; that is, everything starting just after the `EDIT_FILE`, until or unless an editor `QUIT` command is reached. If no editor `QUIT` is provided, `YES` will be assumed at the end of the procedure. In this case, the body of the procedure or included file is considered to be editor commands. The editor reads the entire procedure or included file without reading from the terminal in either line or screen mode.

The `INPUT` parameter allows you to call a file that contains editor subcommands (but not including the `EDIT_FILE` command that starts the editor), and the editor executes those subcommands instead of reading them from the current command stream in either line or screen mode.

For example, the file `EDITOR_SCRIPT` contains the commands:

```
delete_text lines=first
delete_text lines=last
```

This file can be used to delete the first and last lines of any file when entered as follows:

```
edit_file file=file_to_be_shortened input=editor_script
```

You can also refer to the command stream that calls a procedure by using the `$COMMAND_OF_CALLER` function. For example, suppose the procedure, `PROC_A`, contains:

```
PROC proc_a (status)
  do_something thing=whatever_must_precede_editing
  edit_file file=file_to_be_edited input=$command_of_caller
  do_something thing=make_use_file_that_is_fully_edited_now
PROCEND proc_a
```

Once `PROC_A` is prepared, you can execute it in various ways. If you call `PROC_A` from the terminal, after the initial `DO_SOMETHING`, the procedure pauses while the editor talks to you at the terminal (in either line or screen mode according to what you have set for your terminal using `ACTIVATE_SCREEN`). When you tell the editor to `QUIT`, `PROC_A` continues with the second `DO_SOMETHING`, and then finishes.

Procedure PROC_B shows another way to use the editor:

```
PROC proc_b (status)
  do_something thing=whatever_this_proc_needs_to_do_first
  proc_c
    list_lines all
    display_value 'this is a sample proc'
    quit
  do_something thing=final_thing_to_do
PROCEND proc_b
```

```
PROC proc_c (status)
  do_something thing=just_before_editor
  edit_file file=file_c input=$command_of_caller
  do_something thing=just_after_editor
PROCEND proc_c
```

In this example, you call PROC_B at the terminal. PROC_B does something and then calls PROC_C. PROC_C does something to initialize and then calls the editor. This time, \$COMMAND_OF_CALLER refers to the middle of PROC_B, so the editor does the LIST_LINES, DISPLAY_VALUE, and the QUIT subcommands. The last part of PROC_C then executes and you return to PROC_B after QUIT. PROC_B executes its second DO_SOMETHING and is finished.

There are two special files within the editor that are executed each time you start and leave the editor. These files, the prologue and epilogue files, are described in this chapter.

Prologue File	10-1
Epilogue File	10-1



Prologue File

The prologue file is a file containing commands that are executed every time you start the editor. In this file, you can put any subcommands and SCL commands that you want executed each time you start the editor. In effect, this enables you to permanently change settings of function keys, format of the screen, and so on, from the initial settings provided by the system. For example, if you know you are going to use the editor in screen mode on a Viking 721 terminal most of the time, you might want to include an `ACTIVATE_SCREEN CDC721` subcommand in your editor prologue file.

When you start the editor with the `EDIT_FILE` command, you can specify a file to be used as the prologue file with the `PROLOG` parameter. Otherwise, the editor assumes the prologue file is `SCU_EDITOR_PROLOG` in catalog `$USER`.

Epilogue File

You also have the option of setting an epilogue file that is executed when you stop the editor. To specify a file containing `EDIT_FILE` subcommands that you want executed when you leave the editor, use the `SET_EPILOG` subcommand. The format is:

```
SET_EPILOG  
FILE=file
```

The `FILE (F)` parameter specifies the file to contain the `FSE` subcommands. If `FILE` is omitted, `$USER.SCU_EDITOR_EPILOG` is assumed.

The `SET_EPILOG` subcommand may be used anytime within an edit session. If you want epilog file processing to occur automatically, put the `SET_EPILOG` subcommand into your prologue file.

If you do not use the `SET_EPILOG` subcommand, no epilogue file is executed.



Using Other Terminals in Screen Mode

Full Screen Editor supports terminals other than the Viking 721. To use these terminals, you will need to know information specific to these terminals and how it affects your use of the editor. This chapter shows how to do functions performed by Viking 721 function keys on the other supported terminals.

CDC 722	11-1
CDC 722-30	11-5
DEC VT100	11-9
IBM PC or Equivalent	11-13
Zenith Z19 or Heathkit H19	11-17
Zenith Z29	11-22



Using Other Terminals in Screen Mode

11

This chapter provides you with the information you will need to run the editor in screen mode on the following terminals.

- CDC 722
- CDC 722-30
- DEC VT100
- IBM PC or equivalent
- Zenith Z19 or Heathkit H19
- Zenith Z29

Information on these terminals includes:

- Equivalent CDC standard function keys.
- Programmable function key initial settings.

For information on creating a terminal-definition file so that you can use the editor in screen mode with other terminals, refer to the Screen Formatting manual.

CDC 722



Equivalent CDC Standard Function Keys:

Viking 721 Key	CDC 722 Key	Function
(FWD)	(F1) + (NEW LINE)	Moves forward to the next screen of the file.
(BKW)	(SHIFT) (F1) + (NEW LINE)	Moves backward to the previous screen of the file.
(UP)	(SHIFT) (F2) + (NEW LINE)	Positions the file so that the current line is at the top of the screen.
(DOWN)	(F2) + (NEW LINE)	Positions the file so that the current line is at the bottom of the screen.
(HELP)	(F7) + (NEW LINE)	Displays the editor HELP file.
(EDIT)	Move the cursor to the upper file text area, press (HOME) and enter setso s=1	Returns you to the previously edited file.
(INSRT)	(F3) + (NEW LINE)	Inserts a blank character over which you type the new character.
(↑) (INSRT)	(F4) + (NEW LINE)	Inserts an empty line over which you can type a new line of text.
(DELETE)	(SHIFT) (F3) + (NEW LINE)	Deletes the current character.
(↑) (DELETE)	(SHIFT) (F4) + (NEW LINE)	Deletes the current line.
(↑) (CLEAR)	(SHIFT) (CLEAR)	Rewrites the entire screen.
(HOME)	(SHIFT) (HOME)	Moves the cursor to the subcommand line, allowing you to enter subcommands and SCL commands.
(NEXT)	(NEW LINE) or (CR)	Ends an input line.

Initial Programmable Function Key Settings for the CDC 722:

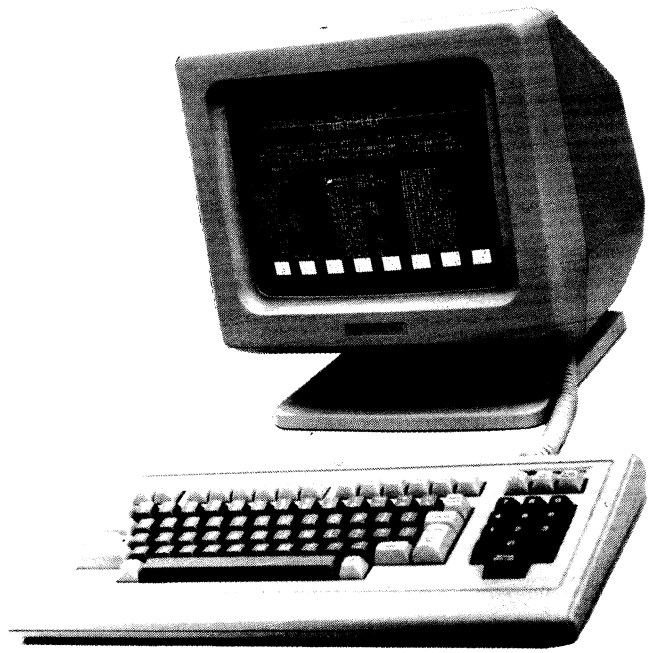
CDC 722 Key	Description
F1 BKW FWD	F1 FWD moves forward one page in the file. When shifted, F1 BKW moves backward one page in the file.
F2 Linedn Lineup	F2 Lineup moves the current line to the top of the screen. When shifted, F2 Linedn positions the current line to the bottom of the screen.
F3 DELC INSC	F3 INSC inserts a blank character at the current character over which you can type a new character. When shifted, the F3 DELC key deletes the current character. You can press the F3 key several times before pressing (NEW LINE) to delete or insert more than one character. It is not until you press (NEW LINE) that the results are shown.
F4 DELL INSL	F4 INSL inserts a blank line over which you can type new text. When shifted, F4 DELL deletes the current line. You can press the F4 key several times before pressing (NEW LINE) to insert or delete more than one line. It is not until you press (NEW LINE) that the results are shown.
F5 Undo MARK	F5 MARK marks a line or lines for later use with another subcommand. These marked lines are not displayed in inverse video as on the Viking 721. When shifted, F5 Undo undoes the previous change to your file.

CDC 722 Key	Description
F6 Move Copy	F6 Copy copies any marked lines or characters to the current line or character. When shifted, F6 Move moves any marked lines or characters.
F7 Left HELP	F7 HELP displays the editor HELP file. When shifted, F7 Left moves your view of the file to the left.
F8 Right END	F8 END stops the editor, making any changes to all open files permanent. When shifted, F8 Right moves your view of the file to the right.
F9 Unmrk EndLin	F9 EndLin moves the cursor to the end of the current line. When shifted, F9 Unmrk cancels any marks you may have set.
F10 [REDACTED]	Undefined.
F11 [REDACTED]	Undefined.

NOTES

- You must press **(NEW LINE)** or **(CR)** after pressing a programmable function key.
- To use the cursor positioning keys or the **(HOME)** key, you must first press **(SHIFT)**.
- The **(EOL)** key clears a line from the position of the cursor to the end of the line. The shifted **(EOL)** key clears the entire screen.
- Do not use the **(TAB)** key to insert tabs. Instead, use the tab character defined by the SET_TAB_OPTIONS subcommand. The horizontal tab character works well (SETTO c=chr\$(9)).

CDC 722-30



Equivalent CDC Standard Function Keys:

Viking 721 Key	CDC 722-30 Key	Function
FWD	F3	Moves forward to the next screen of the file.
BKW	F4	Moves backward to the previous screen of the file.
UP	SHIFT F3	Positions the file so that the current line is at the top of the screen.
DOWN	SHIFT F4	Positions the file so that the current line is at the bottom of the screen.
HELP	SHIFT F5	Displays the editor HELP file.
INSRT	SHIFT IC †	Begins insert mode at cursor. To end, press [RETURN].
⏠ INSRT	SHIFT IL †	Inserts a blank line over which you can type a new line of text.
DLETE	SHIFT DC †	Deletes the current character.
⏠ DLETE	SHIFT DL †	Deletes the current line.
HOME	SHIFT 5 †	Positions the cursor at the editor subcommand line, allowing you to enter editor subcommands.
NEXT	RETURN	Ends an input line.

† Located on the numeric keypad to the right of the main keyboard.

Initial Programmable Function Key Settings for the CDC 722-30:

CDC 722-30 Key	Description
F1 Unmrk MARK	F1 MARK marks a line or lines for later use with another subcommand. When shifted, F1 Unmrk cancels any marks you may have set.
F2 Boxmk Chrmk	F2 Chrmk sets the mark at character boundaries. When shifted, F2 Boxmk sets the mark at a box corner.
F3 LinUp FWD	F3 FWD moves forward one page in the file. When shifted, F3 LinUp moves the current line to the top of the screen.
F4 LinDn BKW	F4 BKW moves backward one page in the file. When shifted, F4 LinDn positions the current line to the bottom of the screen.
F5 Help UNDO	F5 UNDO undoes the previous change to your file. When shifted, F5 Help displays the editor HELP file.
F6 Offset END	F6 END stops the editor, making changes to all open files permanent. When shifted, F6 Offset switches the display of columns 1 through 80 to the display of columns 54 through 133 or the reverse.
F7 locnxt Locat	F7 Locat moves the cursor to the subcommand line and prompts you for the text you want to enter. When shifted, F7 locnxt locates the next occurrence of a previously specified string.
F8 Move Copy	F8 Copy copies any marked lines or characters to the current line or character. When shifted, F8 Move moves any marked lines or character.
F9 Last First	F9 First aligns the screen to display the beginning of the file. When shifted, F9 Last aligns the screen centered around the end of the file.

Initial Programmable Function Key Settings for the CDC 722-30:

CDC 722-30 Key	Description
F10 Endln middl	F10 middl aligns the screen so that the current line is in the middle of the screen. When shifted, F10 Endln moves the cursor to just beyond the end of the current line.
F11 Break	F11 Break divides the current line into two smaller lines just in front of the current column. The cursor stays at the same place, which is just beyond the last character of the first line of the pair.
F12 Join	F12 Join adds the line after the current line onto the end of the current line.

NOTE

To enter INSERT CHARACTERS (IC), INSERT LINES (IL), DELETE CHARACTERS (DC), DELETE LINES (DL), or HOME, use the numbers on the keypad to the right of the main keyboard. For example, to insert characters, press:

(SHIFT) (1)

Then type in the desired characters and press **(RETURN)** to end.

The keys are on the keypad to the right of the main keyboard as follows:

- 1 - IC
- 3 - DC
- 5 - HOME
- 7 - IL
- 9 - DL

Equivalent CDC Standard Function Keys:

The following VT100 keys (except **RETURN**) are located on the keypad to the right of the main keyboard.

Viking

721 Key	DEC VT100 Key	Function
FWD	1 + RETURN	Moves forward to the next screen of the file.
BKW	PF1 + RETURN	Moves backward to the previous screen of the file.
UP	2 + RETURN	Positions the file so that the current line is at the top of the screen.
DOWN	PF2 + RETURN	Positions the file so that the current line is at the bottom of the screen.
HELP	7 + RETURN	Displays the editor HELP file.
EDIT	Move the cursor to the upper file text area, press HOME and enter setso s=1	Returns you to the previously edited file.
INSRT	4 + RETURN	Inserts a blank line, allowing you to type in a new line of text.
INSRT	3 + RETURN	Inserts a blank character over which you type the new character.
DLETE	PF4 + RETURN	Deletes the current line.
DLETE	PF3 + RETURN	Deletes the current character.
CLEAR	. + RETURN	Rewrites the entire screen.
HOME	ENTER + RETURN	Positions the cursor at the editor subcommand line, allowing you to enter editor subcommands.
NEXT	RETURN	Ends an input line.

Initial Programmable Function Key Settings for the DEC VT100:

DEC VT100 Key	Description
F1 BKW FWD	F1 FWD moves forward one page in the file. When shifted, F1 BKW moves backward one page in the file.
F2 Linedn Lineup	F2 Lineup moves the current line to the top of the screen. When shifted, F2 Linedn positions the current line to the bottom of the screen.
F3 DELC INSC	F3 INSC inserts a blank character at the current character over which you can type a new character. When shifted, F3 DELC deletes the current character. You can press the F3 key several times before pressing (NEW LINE) to delete or insert more than one character. It is not until you press (NEW LINE) that the results are shown.
F4 DELL INSL	F4 INSL inserts a blank line over which you can type new text. When shifted, F4 DELL deletes the current line. You can press the F4 key several times before pressing (NEW LINE) to insert or delete more than one line. It is not until you press (NEW LINE) that the results are shown.
F5 Undo MARK	F5 MARK marks a line or lines for later use with another subcommand. These marked lines are highlighted as on the Viking 721. When shifted, F5 Undo undoes the previous change to your file.
F6 Move Copy	F6 Copy copies any marked lines or characters to the current line or character. When shifted, F6 Move moves any marked lines or character.

DEC VT100 Key	Description
F7 HOME HELP	F7 HELP displays the editor HELP file. When shifted, F7 HOME positions the cursor on the editor subcommand line.
F8 Clear END	F8 END stops editor making changes to all open files permanent. When shifted, F8 Clear rewrites your screen.
F9 Unmrk ENDLIN	F9 ENDLIN moves the cursor to the end of the current line. When shifted, F9 Unmrk cancels any marks you have set.

NOTES

- You must press **(RETURN)** after pressing a programmable function key.
- Keypad keys **(1)** through **(9)** are function keys F1 through F9.
- The following shows the correspondence between the shifted function keys and the keypad keys.

Function Key	Associated Keypad Keys
(SHIFT) (F1)	(PF1)
(SHIFT) (F2)	(PF2)
(SHIFT) (F3)	(PF3)
(SHIFT) (F4)	(PF4)
(SHIFT) (F5)	(-)
(SHIFT) (F6)	(.)
(SHIFT) (F7)	(ENTER)
(SHIFT) (F8)	(.)
(SHIFT) (F9)	(0)

The keypad can never be used for numeric values within the editor.

- Turn on output flow control if it's not already on.

IBM PC (or Equivalent)



Equivalent CDC Standard Function Keys:

Viking

721 Key	IBM PC Key	Function
(FWD)	(PgDn)	Moves forward to the next screen of the file.
	(CTRL) (PgDn)	Moves to the end of file.
(BKW)	(PgUp)	Moves backward to the previous screen of the file.
	(CTRL) (PgUp)	Moves to the beginning of file.
(INSRT)	(INS)	Begins insertion mode. Text to the right of the cursor moves as you enter new characters. To end insertion mode, press END .
(↑) (INSRT)	(ALT-I)	Inserts a blank line over which you can type new text.
(DELETE)	(DEL)	Deletes the current character.
(↑) (DELETE)	(ALT-D)	Deletes the current line.
(HOME)	(HOME)	Positions the cursor at the editor subcommand line, allowing you to enter editor subcommands.
(NEXT)	(RETURN)	Ends an input line.

NOTE

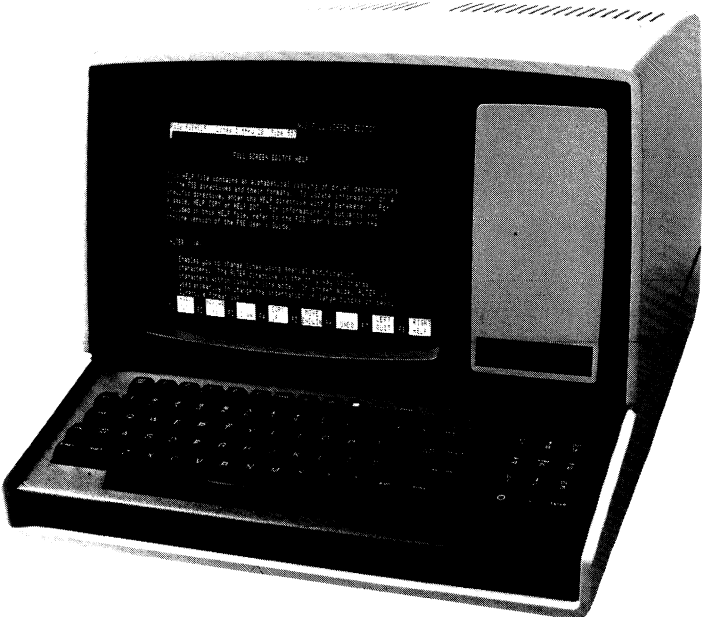
The IBM PC terminal definition assumes that the CDC terminal emulation package CONNECT is running in the PC.

Initial Programmable Function Key Settings for the IBM PC:

IBM PC Key	Description
F1 Unmrk MARK	F1 MARK marks a line or lines for later use with another subcommand. When shifted, F1 Unmrk cancels any marks you may have set.
F2 Trunc Chrmk	F2 Chrmk sets the mark at character boundaries. When shifted, F2 Trunc blanks the remainder of the current line from the cursor to the right margin.
F3 Move Copy	F3 Copy copies any marked lines or characters to the current line or character. When shifted, F3 Move moves any marked lines or characters.
F4 Join Break	F4 Break divides the current line into two smaller lines just in front of the current column. The cursor stays at the same place, which is just beyond the last character of the first line of the pair. When shifted, F4 Join adds the line after the current line to the end of the current line.
F5 Clear UNDO	F5 UNDO undoes the previous change to your file. When shifted, F5 Clear rewrites your screen.
F6 Quit	F6 Quit exits the editor, making changes to all open files permanent.
F7 Locnxt Locate	F7 Locate moves the cursor to the subcommand line and prompts you for the text you want to enter. When shifted, F7 Locnxt locates the next occurrence of a previously specified string.
F8 Offset Locall	F8 Locall locates all occurrences of a previously specified string. When shifted, F8 Offset switches the display of columns 1 through 80 to the display of columns 54 through 133 or from columns 54 through 133 to 1 through 80.

IBM PC Key	Description
F9 Format Middle	F9 Middle aligns the screen so that the current line is in the middle of the screen. When shifted, F9 Format moves words across line boundaries and standardizes spacing between words to fit the current paragraph to the margins. Standard spacing is single except for double spacing after any period, exclamation, question, or colon. Punctuation marks embedded in nonblank text are not double spaced.
F10 Center Endlin	F10 Endlin moves the cursor to the end of the current line. When shifted, F10 Center horizontally centers the current line within the paragraph formatting margins.

Zenith Z19 or Heathkit H19



Equivalent CDC Standard Keys:

Viking

721 Key	Z19 or H19 Key	Function
(FWD)	(F1) + (RETURN)	Moves forward to the next screen of the file.
(BKW)	(F2) + (RETURN)	Moves backward to the previous screen of the file.
(UP)	(F3) + (RETURN)	Positions the file so that the current line is at the top of the screen.
(DOWN)	(F4) + (RETURN)	Positions the file so that the current line is at the bottom of the screen.
(HELP)	(F8) + (RETURN)	Displays the editor HELP file. The (F8) key is the key with the white square on it.
(EDIT)	Move the cursor to the upper file text area, press (HOME) and enter set so s=1	Returns you to the previously edited file.
(↑) (INSRT)	(IL)†	Inserts a blank line over which you can type a new line of text. When you insert lines, the function key prompts move according to the number of lines inserted. To align the prompts properly, press (RETURN) . This key does not require you to press (RETURN) afterward.

† Located on the numeric keypad to the right of the main keyboard.

Viking**721 Key****Z19 or H19 Key****Function****INSRT****IC** †

When you press **IC**, it puts the terminal into insert mode. While insert mode is on, pressing any character moves the existing text to the right and inserts the new character. Insert mode is canceled when you either press **IC** a second time or press **RETURN**.

⏠ DLETE**DL** †

Deletes the current line. When you delete lines, the function key prompts move according to the number of lines deleted. To align the prompts properly, press **RETURN**. This key does not require you to press **RETURN** afterward.

DLETE**DC** †

Deletes the current character.

⏠ CLEAR**SHIFT ERASE** + **RETURN**+ **RETURN**

Rewrites the entire screen.

HOME**HOME**

Positions the cursor at the editor subcommand line, allowing you to enter editor subcommands.

NEXT**RETURN**

Ends an input line.

† Located on the numeric keypad to the right of the main keyboard.

Initial Programmable Function Key Settings for the Zenith Z19/Heathkit H19:

Z19 or H19 Key	Description
F1 MARK F1 FWD	F1 FWD moves forward one page in the file. When shifted, F1 MARK marks a line or lines to be used with another function or subcommand.
F2 Mrkchr F2 BKW	F2 BKW moves backward one page in the file. When shifted, F2 Mrkchr marks a character or characters for use with another function or subcommand.
F3 Unmark F3 Lineup	F3 Lineup moves the current line to the top of the screen. When shifted, F3 Unmark cancels any marks you have set.
F4 Copy F4 Linedn	F4 Linedn positions the current line to the bottom of the screen. When shifted, F4 Copy copies any marked lines or characters to the current line or character.
F5 Move F5 Middle	F5 Middle moves the cursor to the end of the current line. When shifted, F5 Move moves any marked lines or characters to the current line or character.
F6 Endlin F6 UNDO	F6 UNDO undoes the previous change to your file. When shifted, F6 Endlin moves the cursor to the end of the current line.
F7 Left F7 Quit	F7 Quit exits the editor, making changes to your file permanent. When shifted, F7 Left moves your view of the file to the left.
F8 Right F8 HELP	F8 HELP accesses the editor HELP file. When shifted, F8 Right moves your view of the file to the right 40 columns.
F9 	Undefined but available as a function key only when shifted.

Z19 or H19 Key	Description
F10 Locnxt	When shifted, F10 Locnxt locates the next occurrence of a previously specified string.
F11 LocalL	When shifted, F11 LocalL locates all occurrences of a previously specified string.
F12 LOCATE	When shifted, F12 LOCATE moves the cursor to the subcommand line and prompts you for the text you want to enter.

NOTES

- You must enter **(RETURN)** after pressing an editor function key.
- Function keys F1 through F5 are terminal keys f1 through f5.
- Function keys F6 through F8 are the following terminal keys:
 - F6 Blue square
 - F7 Red square
 - F8 White square
- To enter shifted function keys, use the numbers on the keypad to the right of the main keyboard. For example, to enter a shifted F3, press:

(SHIFT) **(3)** + **(RETURN)**

The **(3)** key is on the keypad to the right of the main keyboard.

- The F10 through F12 keys are on the keypad to the right of the main keyboard and are:

F10 **(0)**

F11 **(.)**

F12 **(ENTER)**

- The **(ERASE)** key, followed by **(RETURN)**, rewrites the entire screen.
- The Z19 hardware has tabs set every eighth column beginning with 1. These tabs are set at columns:

1 9 17 25 33 41 49 57 65 73

To specify tabs other than these, use the tab character as set by the SET_TAB_OPTIONS subcommand.

ZENITH Z:29

Zenith Z29



Revision C



Equivalent CDC Standard Function Keys:

Viking

Viking 721 Key	Zenith Z29 Key	Function
(FWD)	(F1) + (RETURN)	Moves forward to the next screen of the file.
(BKW)	(F2) + (RETURN)	Moves backward to the previous screen of the file.
(UP)	(F3) + (RETURN)	Positions the file so that the current line is at the top of the screen.
(DOWN)	(F4) + (RETURN)	Positions the file so that the current line is at the bottom of the screen.
(HELP)	(HELP) + (RETURN)	Displays the editor HELP file.
(EDIT)	(DEL) + (RETURN)	Returns you to the previously edited file.
(↵) (INSRT)	(IL)†	Inserts a blank line over which you can type a new line of text. When you insert lines, the function key prompts move according to the number of lines inserted. To align the prompts properly, press (RETURN) . This key does not require you to press (RETURN) afterward.
(INSRT)	(IC)†	When you press (IC) , it puts the terminal into insert mode. While insert mode is on, pressing any character moves the existing text to the right and inserts the new character. Insert mode is canceled when you either press (IC) a second time or press (RETURN) .

† Located on the numeric keypad to the right of the main keyboard.

Viking

721 Key	Zenith Z29 Key	Function
 DLETE	DL †	Deletes the current line. When you delete lines, the function key prompts move according to the number of lines deleted. To align the prompts properly, press RETURN . This key does not require you to press RETURN afterward.
DLETE	DC †	Deletes the current character.
 CLEAR	SHIFT ERASE + RETURN	Rewrites the entire screen.
HOME	HOME	Positions the cursor at the editor subcommand line, allowing you to enter editor subcommands.
NEXT	RETURN	Ends an input line.

† Located on the numeric keypad to the right of the main keyboard.

Initial Programmable Function Key Settings for the Zenith Z29:

**Zenith
Z29 Key****Description**

F1	MARK fwd	F1 fwd moves forward one page in the file. When shifted, F1 MARK marks a line or lines for later use with another subcommand.
F2	MRKCHR bkw	F2 bkw moves backward one page in the file. When shifted, F2 MRKCHR marks a character or characters for use with another function or subcommand.
F3	UNMARK lineup	F3 lineup moves the current line to the top of the screen. When shifted, F3 UNMARK cancels any marks you have set.
F4	COPY linedn	F4 linedn positions the current line to the bottom of the screen. When shifted, F4 COPY copies any marked lines or characters to the current line or character.
F5	Move Middle	F5 Middle centers the display vertically around the line the cursor is on. When shifted, F5 Move moves any marked lines or characters to the current line or character.
F6	Endlin UNDO	F6 UNDO undoes the previous change to your file. When shifted, F6 EndLin moves the cursor to the end of the current line.
F7	Left Quit	F7 Quit exits the editor, making changes to your file permanent. When shifted, F7 Left moves your view of the file to the left.

Zenith

Z29 Key

Description

F8	RIGHT top	F8 top aligns the screen to display the beginning of the file. When shifted, F8 RIGHT moves your view of the file to the right 40 columns.
F9	BOTTOM wrif	F9 wrif checkpoints the file being edited (WRITE_FILE). When shifted, F9 BOTTOM aligns the screen centered around the end of the file.
F10	Locnxt	When shifted, F10 Locnxt locates the next occurrence of a previously specified string.
F11	Locall	When shifted, F11 Locall locates all occurrences of a previously specified string.
F12	LOCATE	When shifted, F12 LOCATE moves the cursor to the subcommand line and prompts you for the text you want to enter.
F13	Join	When shifted, F13 JOIN adds the line after the current line onto the end of the current line.
F14	Break	When shifted, F14 BREAK divides the current line into two smaller lines just in front of the current column. The cursor stays at the same place, which is just beyond the last character of the first line of the pair.

NOTES

- You must enter **(RETURN)** after pressing an editor function key.
- Function keys F1 through F9 are terminal keys f1 through f9.
- To enter shifted function keys, use the numbers on the keypad to the right of the main keyboard. For example, to enter a shifted F3, press:

(SHIFT) **(3)** + **(RETURN)**

The **(3)** key is on the keypad to the right of the main keyboard.

- The F10 through F14 keys are on the keypad to the right of the main keyboard and are:

F10 **(0)** F13 **(,)**

F11 **(.)** F14 **(-)**

F12 **(ENTER)**

- The **(ERASE)** key, followed by **(RETURN)**, rewrites the entire screen.
- The Z29 hardware has variable tab settings. To specify tabs, use the tab character as set by the SET_TAB_OPTIONS subcommand.

● **Appendixes**

Glossary	A-1
Character Set	B-1
Command Strings That Define Function Keys	C-1
Viking 721 Terminal Settings	D-1





The following terms are used in this manual:

A

Alphabetic Character

One of the following letters:

A through Z

a through z

See **Character** and **Alphanumeric Character**.

Alphanumeric Character

An alphabetic character or a digit. See **Character**, **Alphabetic Character**, and **Digit**.

ASCII

American National Standard Code for Information Interchange. A 7-bit code representing a prescribed set of characters. The 7-bit ASCII code character is stored right-justified in an 8-bit byte.

B

Batch Mode

A mode of execution where a job is submitted and processed as a unit without intervention from the user. Contrast with **Interactive Mode**.

Block

A logical or physical grouping of data.

Boolean

A kind of value that can be either **TRUE** or **FALSE**.

Boolean Constant

A constant that represents a boolean (logical) value of **TRUE** or **FALSE**. One of the following names can be used to specify a boolean constant:

TRUE **FALSE**

YES **NO**

ON **OFF**

Byte

A group of bits. For **NOS/VE**, a byte is 8 bits. An **ASCII** character code uses the rightmost 7 bits of 1 byte.

C

Catalog

A directory of files and catalogs maintained by the system for a user. The catalog \$LOCAL contains only file entries.

Also, the part of a path that identifies a particular catalog in a catalog hierarchy. The format is as follows:

name.name.name

where each name is a catalog. See Catalog Name and Path.

Catalog Name

The name of a catalog in a catalog hierarchy (path). By convention, the name of the user's master catalog is the same as the user's user name.

CDC Standard Keys

Keys on CDC terminals that perform the same functions on all applications and can be performed on all supported terminals.

Character

A letter, digit, space, or symbol that is represented by a code in one or more of the standard character sets.

It is also referred to as a byte when used as a unit of measure to specify block length, record length, and so forth.

A character can be a graphic character or a control character. A graphic character is printable; a control character is nonprintable and is used to control an input or output operation.

Closed File

A file that is no longer open. Changes in a closed file cannot be undone. A closed file is not affected by the END FALSE or QUIT FALSE subcommand.

Command

An instruction to NOS/VE. Commands that can only be entered from within the editor are editor subcommands.

Common Parameter

A parameter used by several subcommands for which the same values can be entered.

Current Line

The line on which the cursor is positioned. If the cursor is on the subcommand line, the current line is the line on which the cursor was positioned when you pressed **HOME**.

Current Paragraph

A block of lines including the current line separated from the rest of the file by empty lines.

Current Position

The location of the cursor in the file at this time. The editor determines the current position by the line and column number. If the cursor is on the subcommand line, the current position is the position at which the cursor was positioned when you pressed **(HOME)**.

Cursor

The pointer used by your terminal to indicate where you are positioned in the file.

D

Deck

A sequence of lines in a source library that can be manipulated as a unit by the Source Code Utility (SCU).

Delimiter

A character or sequence of contiguous characters that identify the end of a string of characters and separate that string of characters from the following string of characters. A delimiter is not part of the string of characters that it delimits.

Delimiter String

A string that marks the end of text input.

Digit

One of the following characters:

0 1 2 3 4 5 6 7 8 9

E

Editing Keys

Keys such as **(INSRT)**, **(DELETE)**, and **(ERASE)**, whose functions are predetermined and usually performed by software at your terminal. Contrast with Programmable Function Keys.

Editing Session

The time from when you start the editor (by entering the `EDIT_FILE` or `EDIT_LIBRARY` subcommand) to the time you stop the editor.

Ellipsis

1. Two or more consecutive periods at the end of a physical line to indicate command line continuation. The ellipsis can be optionally preceded and/or followed by a space.
2. Two consecutive periods separating two values to indicate a range of values in a parameter list.

End-of-Partition (EOP)

A special delimiter in a file with variable record type.

Epilogue

The procedure file that is executed when you stop the editor.

F

Family

A logical grouping of NOS/VE users that determines the location of their permanent files. A family can be subdivided into accounts and projects.

Family Name

A name that identifies a NOS/VE family. See Family.

File

An SCL element specifying a temporary or permanent file, including its path and, optionally, a cycle reference (for permanent files). A file is identified by specifying a path and, optionally, a cycle reference (for permanent files) as follows:

path.cycle reference

A collection of information referenced by a name.

File Header

The line containing the file name. It is displayed during a screen editing session.

File Name

The name of a NOS/VE file. It is used in a file reference to identify the file. See Name.

File Position

The location in the file at which the next read or write operation will begin. The file position designators are:

\$ASIS Leave the file in its current position.

\$BOI Position the file at the beginning-of-information.

\$EOI Position the file at the end-of-information.

File Reference

An SCL element that identifies a file and, optionally, the file position to be established prior to use. The format of a file reference is as follows:

file.file position

See File and File Position.

FSE

See Full Screen Editor.

Full Screen Editor

An editor enabling you to edit files either page by page or line by line.

Function Key

A key on the terminal that, when pressed, performs a specified operation. The operation can be either defined by the software or built into the terminal.

Function Key Prompts

Labels displayed on your screen which describe the function of a programmable function key prompt.

G

Graphic Character

A character that can be printed or displayed.

I

Integer

A value representing one of the numbers 0, +1, -1, +2, -2 ...

Interactive Mode

A mode of execution where a user enters commands at a terminal and each command elicits a response from the computer. Contrast with Batch Mode.

J

Job

A set of tasks executed for a user name. NOS/VE accepts interactive and batch jobs. In interactive mode, a job is usually the same as a terminal session.

L

Line Identifier

The unique identifier of a line in a deck. The line identifier consists of a modification name followed by a sequence number. The modification name identifies the modification to which the line belongs.

Local File

A file that is accessed via the local catalog named \$LOCAL. See also File, Path, and Local Path.

Local File Name

The name used by an executing job to reference a file while the file is assigned to the job's \$LOCAL catalog. Only one file can be associated with a given name in one job; however, in one job a file can have more than one instance of the file open by that name.

Local Path

Identifies a local file as follows:

\$LOCAL.file name

Login

The process used to gain access to the system.

Logout

The process used to end a job.

M

Mask Character

A character the editor considers a match to any character in a string comparison. In other words, it is a wild card character.

Message Row

The row on the screen where messages are displayed.

N

Name

A combination of from 1 through 31 characters chosen from:

Alphabetic characters (A through Z and a through z)

Digits (0 through 9)

Special characters (#, @, \$, or _)

The first character of a name cannot be a digit.

Numeric Character

Any digit 0 through 9.

O

Object

The thing being edited. In this manual, either a file or a deck.

Open File

A file prepared for data access. In this manual, an open file is a file that is being edited or a file that has been edited but has not been closed with the `END_FILE` subcommand. When you enter the `END` or `QUIT` subcommand all open files are closed.

P

Path

Identifies a file. It may include the family name, user name, subcatalog name or names, and file name.

Permanent Catalog

A catalog of permanent files.

Permanent File

A file that does not go away when you log off or when the system is deadstarted. A permanent file has an entry in a permanent catalog. See `File`.

Programmable Function Keys

Keys whose function you can redefine. Contrast with `Editing Keys`.

Prologue

The file of commands that is executed when you start the editor.

S

Search Margins

An editing mode in which a file is edited with a page of text as the basic unit of operation. Also, an SCU editor string-search option that restricts a string search to a range of columns in each line.

Subcommand

A command that can only be entered and recognized from within the editor.

T

Text-Embedded Directive

A text line that SCU processes as a directive when expanding a deck or a file.

U

User Name

A name that identifies a NOS/VE user and the location of a user's permanent files in the user's family.

User Path

Identifies a file or catalog via a user name and optionally a relative path as follows:

.user name.relative path

or

\$USER.relative path

V

V-Type Record

Variable-sized records; system default record type. Each V-type record has a record header. The header contains the record length and the length of the preceding record.

W

Word

A string of alphanumeric characters (plus the special characters \$, #, @, and _) delimited by nonalphanumeric characters, blank characters, and beginning or end of lines.

Table B-1 lists the ASCII character set used by the NOS/VE system.

NOS/VE supports the American National Standards Institute (ANSI) standard ASCII character set (ANSI X3.4-1977). NOS/VE represents each 7-bit ASCII code in an 8-bit byte. The 7 bits are right-justified in each byte. For ASCII characters, the leftmost bit is always zero.

In addition to the 128 ASCII characters, NOS/VE allows use of the leftmost bit in an 8-bit byte for 256 characters. The use and interpretation of the additional 128 characters is user defined.

Table B-1. ASCII Character Set

ASCII Code			Graphic or Mnemonic	Name or Meaning
Decimal	Hexadecimal	Octal		
000	00	000	NUL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign
036	24	044	\$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe
040	28	050	(Opening parenthesis
041	29	051)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus
044	2C	054	,	Comma
045	2D	055	-	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant

(Continued)

Table B-1. ASCII Character Set (Continued)

ASCII Code			Graphic or Mnemonic	Name or Meaning
Decimal	Hexadecimal	Octal		
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equals
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	{	Opening bracket

(Continued)

CHARACTER SET

Table B-1. ASCII Character Set (Continued)

ASCII Code			Graphic or Mnemonic	Name or Meaning
Decimal	Hexadecimal	Octal		
092	5C	134	\	Reverse slant
093	5D	135]	Closing bracket
094	5E	136	˘	Circumflex
095	5F	137	—	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174		Vertical line
125	7D	175	}	Closing brace
126	7E	176	˘	Tilde
127	7F	177	DEL	Delete

Subcommand Strings That Define Function Keys

C

The appendix lists the initial settings of the programmable function keys for all the supported terminals. The string is the command string which defines the particular key and the key prompt is the label that appears on the screen.

Viking 721 Subcommand String Settings

The subcommand strings associated with the initial settings for the Viking 721 terminal are:

<u>String</u>	<u>Key Prompt</u>
1	
mark_lines	MARK
unmark	Unmrk
2	
mark_characters	Chrmk
mark_boxes	Boxmk
3	
align_screen top=first	FIRST
align_screen middle=last	LAST
4	
align_screen middle=current	middle
position_cursor l=c c=\$strlen(\$lt)+1	endlin
5	
undo	UNDO
6	
end	END
7	
locate_text t=\$screen_input('Enter search string')	LOCATE
position_cursor; position_cursor rs=true	locnxt
8	
esv\$text=\$screen_input('Enter search string');	locall
if esv\$text='' then; esv\$text=\$text;	
ifend: locate_text rs=true l=a v=true t=esv\$text	
exchange_screen_width	80/132

CDC 722 Subcommand String Settings

The subcommand strings associated with the default settings for the CDC 722 terminal are:

String	Key Prompt
1 align_screen bottom=first_screen align_screen top=last_screen	BKW FWD
2 align_screen bottom=current align_screen top=current	Linedn Lineup
3 delete_characters c=c insert_characters nt=' '	DEL ^C IN ^{SC}
4 delete_lines l=c insert_lines p=b nt=''; position_cursor d=b	DELL INSL
5 undo mark_lines	Undo MARK
6 move_text l=m p=b; unmark copy_text l=m p=b	Move Copy
7 align_screen o=0 help	Left HELP
8 esv\$off=30; if \$current_column<>1 then; esv\$off=\$current_column-1; ifend; align_screen offset=esv\$off; position_cursor l=c c=\$current_column+30	Right
end	END
9 unmark position_cursor l=c c=\$strlen(\$lt)+1)	Unmrk Endlin

CDC 722-30 Subcommand String Settings

The subcommand strings associated with the default settings for the CDC 722-30 terminal are:

String	Key Prompt
1 unmark mark_lines	Unmrk MARK
2 mark_boxes mark_characters	Boxmk Chrmk
3 align_screen top=current align_screen top=last_screen	LinUp FWD
4 align_screen bottom=current align_screen bottom=first_screen	LinDn BKW
5 help undo	Help UNDO
6 if\$offset=0 then; alis o=53; else; alis o=0; ifend end	Offset END
7 position_cursor; position_cursor rs=true; locate_text t=\$screen_input('Enter search string')	locnxt Locat
8 move_text l=m p=b; unmark copy_text l=m p=b	Move COPY
9 align_screen middle=last align_screen top=first	Last First
10 position_cursor l=c c=1+\$strlen(\$lt) align_screen middle=current	Endln middl
11 break_text	Break
12 join_text	Join

DEC VT100 Subcommand String Settings

The subcommand strings associated with the default settings for the DEC VT100 terminal are:

String	Key Prompt
1 align_screen bottom=first_screen align_screen top=last_screen	BKW FWD
2 align_screen bottom=current align_screen top=current	Linedn Lineup
3 delete_characters c=c insert_characters nt=' '	DELC INSC
4 delete_lines l=c insert_lines p=b nt=''	DELL INSL
5 undo mark_lines	Undo MARK
6 move_text l=m p=b; unmark copy_text l=m p=b	Move Copy
7 position_cursor r=\$home_row c=1 help	HOME HELP
8 activate_screen end	Clear END
9 unmark position_cursor l=c c=\$strlen(\$lt)+1)	Unmrk Endlin

IBM PC (or Equivalent) Subcommand String Settings

The subcommand strings associated with the default settings for the IBM PC (or equivalent) terminal are:

<u>String</u>	<u>Key Prompt</u>
1 unmark mark_lines	Unmrk MARK
2 delete_text l=c b=s c=\$cc..max mark_characters	Trunc Chrmk
3 move_text l=m p=b; unmark copy_text l=m p=b	Move COPY
4 join_text break_text	Join Break
5 acts undo	Clear UNDO
6 quit	QUIT
7 locate_next locate_string t=\$screen_input('Enter search string')	Locnxt Locate
8 if\$offset=0 then; alis o=53; else; alis o=0; ifend esv\$text=\$screen_input('Enter search string'); if esv\$text='' then locate_all; else; locate_all t=esv\$text; ifend	Offset Locall
9 format_paragraphs align_screen middle=current	Format Middle
10 center_lines position_cursor l=c c=\$strlen(\$lt)+1	Center Endlin

Zenith Z19/Heathkit H19 Subcommand String Settings

The subcommand strings associated with the default settings for the Zenith Z19 and Heathkit H19 terminal are:

String	Key Prompt
1 mark_lines align_screen top=last_screen	MARK FWD
2 mark_characters align_screen bottom=first_screen	Mrkchr BKW
3 unmark align_screen top=current	Unmark Lineup
4 copy_text l=m p=b align_screen bottom=current	Copy Linedn
5 move_text l=m p=b;unmark align_screen middle=current	Move Middle
6 position_cursor l=c c=\$strlen(\$lt)+1 undo	Endlin UNDO
7 align_screen offset=0 end	Left Quit
8 esv\$off=30; if \$current_column<>1 then; esv\$off=\$current_column-1; ifend; align_screen offset=esv\$off; position_cursor l=c c=\$current_column+30 help	Right HELP

String	Key Prompt
10 position_cursor; position_cursor rs=true	Locnxt
11 esv\$text=\$screen_input('Enter search string'); if esv\$text='' then; esv\$text=\$text; ifend; locate_text rs=true l=a v=true t=esv\$text	Locall
12 locate_text t=\$screen_input('Enter search string')	LOCATE

Zenith Z29 Subcommand String Settings

The subcommand strings associated with the default settings for the Zenith Z29 terminal are:

String	Key Prompt
1 mark_lines align_screen top=last_screen	MARK fwd
2 mark_characters align_screen bottom=first_screen	MRKCHR bkw
3 unmark align_screen top=current	UNMARK lineup
4 copy_text l=m p=b align_screen bottom=current	COPY linedn
5 move_text l=m p=b; unmark align_screen middle=current	MOVE Middle
6 position_cursor l=c c=\$strlen(\$lt)+1 undo	Endlin UNDO
7 align_screen offset=0 end	Left Quit
8 esv\$off=30; if \$current_column < > 1 then; esv\$off=\$current_column ifend; align_screen offset=esv\$off; position_cursor l=c c=\$current_column+30 align_screen top=first	RIGHT top
9 align_screen middle=last write_file	BOTTOM wrf
10 position_cursor; position_cursor rs=true;	Locnxt

String	Key Prompt
11 esv\$text=\$screen_input('Enter search string'); if esv\$text='' then; esv\$text=\$text; ifend; locate_text rs=true l=a v=true t=esv\$text	Locall
12 locate_text t=\$screen_input('Enter search string')	LOCATE
13 join_text	JOIN
14 break_text	BREAK

Viking 721 Terminal Settings

D

The Viking 721 terminal has three sets of parameters that must be set correctly to ensure correct operation under the editor. The first set, the terminal installation parameters, tells the terminal what additional items are installed with the terminal. The second set, the mode installation parameters, determines how each mode will operate. The third set, the operator parameters, allows the terminal user to change certain guidelines.

Once the terminal is installed the first two sets of parameters need not be changed. The settings are stored in the terminal's nonvolatile memory and do not need to be reentered unless the battery loses its power. If you want detailed information about these parameters refer to the Control Data 721-21/31 Owner Manual. The CDC 721-10/20/30 Hardware Reference Manual also contains the needed information (ordering information is in About This Manual).

The operator parameters allow you to temporarily change some of the guidelines set up when the terminal was installed. When you reset the terminal or turn it off, the settings go back to their defaults. To change the default setting you must change some of the installation parameters. For use with the editor the operator parameters should be set as shown here. These parameters are software toggle switches. That is, you have a limited number of options (usually two) from which to choose. To change a setting, press the corresponding programmable function key. The first set of prompts appears when you turn on your terminal.

```
F MODE 1 F MODE 2 F MODE 3 F MODE 4 F MODE 5 F MODE 6 F MODE 7 F TRMNL
1 CYBER 2 PLATO 3 CP/M 4 DISK 5 4014 6 7 PACK 8 TEST
```

Press:

```
F MODE 1
1 CYBER
```

to select CYBER mode. At this point, the screen becomes blank except for the cursor.

Press:

```
(SETUP)
```

to display the operator parameters and change them, if necessary, to the following settings:

```
F Return F LINE F PRNTR F MARGIN F ALERT F LOCK F N PAD F SCREEN F CYBER F MORE
1 2 ON 3 OFF 4 ON 5 SOFT 6 ALPHA 7 NORMAL 8 ROLL 9 LARGE 10 SELECT
```

VIKING 721 TERMINAL SETTINGS

Then, press:

F MORE
10 SELECT

to set more operator parameters to the following:

F return F BACKGD F CURSOR F CURSOR F BAUD F DUPLEX F CHR/LN F LINES F XPARNT F mode
1 2 DARK 3 BLOCK 4 SOLID 5 1200 6 HALF 7 80 8 30 9 OFF 10 SELECT

When you have set all parameters, press

F return
1

to remove the prompts from the screen. You are now ready to log in to NOS/VE.

NOTE

You must also ensure that parity is set to EVEN or NONE.

Index





Index

A

About this manual 7
ACTIVATE_SCREEN 3-2; 5-2
ACTS 3-2; 5-2
Adding
 Blank lines 3-30; 5-25
 Characters 3-27; 5-23
 Lines 3-28; 5-24
 Text to end of line 3-11
 Words 3-29; 5-26
Adding trailing blanks 7-12
Additional related manuals 12
ALIGN_SCREEN 7-8
Alignment of screen 7-8
ALIS 7-8
ALL value 4-4
Allowable characters
 in a word 7-14
Arrow keys 3-8; 5-10
ASCII character set B-1
Audience 7

B

BACK 6-2
Backspace 2-2; 6-1
BACKTAB 5-12; 6-1
Backward 3-16
Basic editing functions 3-1
Basics 3-1
Before you continue 4-1
Beginning editor 3-1
BKW 3-16; 5-14; 6-2
Blank lines
 Delete 3-26; 5-30
 Deleting 5-36
 Insert 3-30; 5-22,25
Blanks 1-2; 7-12
Blocks of empty lines 3-26
BOUNDARY parameter 4-2
Boxmk 6-3
Break 5-68; 6-5
BREAK_TEXT 5-69

Breaking lines 5-68
BRET 5-69

C

Cancelling tab settings 7-11
Capabilities 1-1
Carriage return 2-2
CDC standard function keys 6-2
CDC 722 function keys 11-1
CDC 722 terminal 11-1
CDC 722 2-3
CDC721 7-1
CDC722-30 2-3; 7-1; 11-5
CDC722 2-3; 7-1; 11-1
CENL 5-73
CENTER_LINES 5-73
Changing
 Characters allowed
 in a word 7-14
 Columns displayed 7-1
 Files displayed 7-1
 How lines are listed in
 line mode 7-15
 Line width 7-12
 Lines 5-38
 Lines displayed 7-1
 Rows of prompts 7-1
 Screen alignment 7-8
 Screen content 3-9
 Screen display 7-1
 Tab character 7-10
 Tab setting 7-10
 Tab settings 7-10
 Text 5-40
 Verify option 7-13
Character
 Copying 3-42
 Mark 5-42
 Marking 3-40; 6-3
 Mask 4-9
 Moving 3-45
 Tab 7-10
Character set B-1

Characters

Delete 3-23; 5-30,31; 6-1

Deleting 5-37

Insert 3-27; 5-22,23; 6-1

Replace 5-40

Chrmk 3-40; 5-42; 6-3**CLEAR** 6-1**CLEAR_TAB** 7-11

Clearing screen 6-1

Clearing tab 7-11

CLET 7-11

Close files 4-6

Closing

All edited files 5-9

Current file 5-10

Closing a deck 8-10

Column numbers list 7-17

COLUMN parameter 4-3**Columns**Changing number
displayed 7-1

Tab 7-10

132 6-4

80 6-4

\$COMMAND 5-1

Common editing functions 5-1

Common parameters 4-2

Control Data Viking 721

terminal 2-1

Control Data 722 7-1

Conventions 10

COPT 5-55

Copy 6-5

COPY key 5-46**COPY_TEXT** 5-55**COPY** 3-41,43**Copying**

Between files 5-46

Between working and external
files 5-57

Characters 3-42

Function keys 5-46; 6-5

General description 3-41

Lines 3-41

Subcommands 5-55

Correcting errors in

screen mode 3-9

Correcting mistakes 3-46

Creating a file 3-6; 5-4

Creating multipartition files 5-67

Current status 3-5

CURRENT value 4-3,4

Cursor 3-5

Cursor movement 2-2; 3-8

Cursor positioning 5-12

D**DATA** 6-2**DC** 5-31**DEACTIVATE_SCREEN** 5-10**DEAS** 5-10**DEC VT100** 2-3; 7-1; 11-9**Deck**

Close 8-10

Deleting changes 8-8

Editing a specific 8-5,7

Editing the first 8-5,7

Editing the last 8-6,7

Editing the next 8-6,7

Opening 8-5,6

Deck listings 7-15

Decks 8-1

dedent 5-30; 6-5

Define tab character 7-10

Defining function keys 6-6

Definitions A-1

DELC 5-31

delel 3-26; 5-30; 6-4

DELEL 5-36**DELETE_CHARACTERS** 5-31**DELETE_EMPTY_LINES** 5-36**DELETE_LINES** 5-32**DELETE_TEXT** 5-34**DELETE_WORD** 5-36**Deleting**

Blanks 1-2

Blocks of empty lines 3-26

Blocks of text 5-34

Characters 3-23; 5-30,31; 6-1

Characters from the beginning
of lines 5-37

Current character 6-1

Current line 6-1

- Deck changes 8-8
 - Empty line 3-26
 - Empty lines 5-30,36; 6-4
 - End of line 6-1
 - General description 3-23
 - Line 3-24
 - Lines 5-30,32; 6-1
 - Subcommands 5-30
 - Tab columns 7-11
 - Words 3-24; 5-30,36
 - Deleting function keys 5-30
 - DELL 5-32
 - DELT 5-34
 - DELW 5-36
 - delwrd 3-24; 5-30,68; 6-5
 - Discarding deck changes 8-8
 - DISCN 7-17
 - DISES 7-16
 - DISP 7-16
 - DISPLAY_COLUMN_
 - NUMBERS 7-17
 - DISPLAY_EDITOR_
 - STATUS 7-16
 - DISPLAY_POSITION 7-16
 - Displaying
 - Status information 7-16
 - Displaying column numbers 7-17
 - Displaying status
 - information 7-16
 - Displays next screen 6-2
 - DLETE 3-23; 5-30; 6-1
 - DOWN 6-2
 - DW 5-36
- E**
- EDID 8-5
 - EDIF 3-1; 5-1
 - EDIFD 8-5
 - EDIL 8-2
 - EDILD 8-6
 - EDIND 8-6
 - EDIT_DECK 8-5
 - EDIT_FILE 3-1; 5-1
 - EDIT_FIRST_DECK 8-5
 - EDIT_LAST_DECK 8-6
 - EDIT_LIBRARY 8-2
 - EDIT_NEXT_DECK 8-6
 - Edit SCU decks 8-2
 - Edit top file 6-2
 - EDIT 3-21; 6-2; 7-2
 - Editing
 - First deck on library 8-5,7
 - Last deck on library 8-6,7
 - Next deck on library 8-6,7
 - Specific deck 8-5,7
 - Editing keys 6-1
 - Editing more than one file at the
 - same time 7-2
 - Editing SCU decks 8-1
 - Editor concepts 4-1
 - Editor options 7-1
 - Editor status 6-4; 7-16
 - Empty lines
 - Delete 3-26; 5-30
 - Deleting 5-36
 - Insert 3-30; 5-22,25
 - END_DECK 8-10
 - END_FILE 5-10
 - End-of-partition delimiter 5-1,67
 - End split screen 6-2
 - END 3-47; 5-9; 6-4; 8-11
 - ENDD 8-10
 - ENDF 5-10
 - endlin 5-12; 6-3
 - ENDLIN 3-11
 - Entering subcommands
 - Line mode 5-7
 - Screen mode 3-19; 5-6
 - Entering text
 - Line mode 5-5
 - Screen mode 5-4
 - Entering, text
 - Screen mode 3-7
 - Epilogue file 10-1
 - ERASE 6-1
 - Erasing 6-1
 - Example procedures 9-10
 - EXCHANGE_POSITION 5-45
 - EXCHANGE_SCREEN_
 - WIDTH 7-8
 - EXCP 5-45
 - EXCSW 7-8
 - External files 4-5

F

File

Creating 3-6; 5-4

Header 3-4

Help 5-8; 6-2

Multipartition 5-67

Reposition 6-2

Size 7-16

Text 3-5

Variable record type 5-67

FILE 3-1

Files

Changing number
displayed 7-1

Close 4-6

Epilogue 10-1

External 4-5

Input 5-1

Inserting 5-28

Multipartition 5-1

Open 4-6

Output 5-1

Prologue 10-1

Specification 5-1

Working 4-5

FIRST 6-3

First line 3-15

FIRST_MARK value 4-3,4

FIRST_SCREEN value 4-4

FIRST value 4-4

Format of the screen 3-4

FORMAT_PARAGRAPHS 5-72

Formatting text 5-68

FORP 5-72

Forward 3-16

Full Screen Editor 8

Capabilities 1-1

Stopping 3-47; 6-4

Function key

Definitions 7-16

Function key prompts 7-2

Function keys 6-1

CDC standard 6-2

CDC 722 11-1

CDC 722 definitions C-3

DEC VT100 11-6

DEC VT100 definitions C-4

Delete 5-30

Editing 6-1

Heathkit H19 11-10

Heathkit H19 definitions C-5

Insert 5-22

Programmable 3-5; 6-3

Viking 721 definitions C-1

Zenith Z19 11-10; C-5

Functions 9-5

FWD 3-16; 5-14; 6-2

F1 MARK 3-39; 5-42,46,61; 6-3

F10 delwrd 3-23,24

F10 inswrd 3-29; 5-22,68; 6-5

F11 Break 5-68; 6-5

F12 Join 5-68; 6-5

F14 Copy 6-5

F15 INDENT 5-22; 6-5

F2 Chrmk 3-40; 5-42,46,61; 6-3

F3 FIRST 3-15; 6-3

F4 ENDLIN 3-11

F4 middle 6-3

F5 UNDO 3-46; 5-65; 6-4

F6 END 3-47; 5-9; 6-4

F7 LOCATE 3-32; 5-15; 6-4

F7 locnxt 3-35

F8 locall 3-36; 6-4

F9 delel 3-26

F9 insel 3-30; 5-22; 6-4

G

General Description 6-4

Function key 5-65

Subcommand 5-66

Getting help 5-8; 6-2

Getting Help 3-19

Getting started 3-1

Glossary A-1

Going to the first line 3-15

Going to the last line 3-15

H

Heathkit H19 2-3; 7-1; 11-17

Help 3-19; 5-8; 6-2

HOME 3-19; 5-12

How to enter subcommands 5-6

I

IBM PC 2-3; 7-1; 11-13
 IC 5-23
 Increasing the file length 3-12
 INDENT_TEXT 5-27,37
 INDENT 5-22; 6-5
 Indenting text 5-22,27
 Index
 INDТ 5-27,37
 INSC 5-23
 inset 3-30; 6-4
 INSEL 5-25
 INSERT_CHARACTERS 5-23
 INSERT_EMPTY_LINES 5-25
 INSERT_LINES 5-24
 INSERT_WORD 5-26
 Inserting
 Blank characters 5-22
 Blank lines 3-30; 5-22
 Blocks of empty lines 3-30
 Characters 3-27; 5-23; 6-1
 Empty lines 3-30; 5-22,25; 6-5
 Files 5-28
 Function Keys 5-22
 General description 3-27
 Lines 3-28,30; 6-1
 Subcommands 5-22
 Text from another file 5-28
 Words 3-29; 5-22,26; 6-5
 inset 5-22
 INSL 5-24
 INSRT 3-27; 5-22; 6-1
 INSW 5-26
 inswrд 3-29; 5-22,68; 6-5

J

Join 5-68; 6-5
 JOIN_TEXT 5-70
 Joining lines 5-68,70; 6-5
 JOIT 5-70

K

Keys
 CDC standard 6-2

Cursor positioning 2-2
 Editing 6-1
 Function 6-1
 Prompts displayed 7-2
 Redefine setting 6-6

L

Last line 3-15
 LAST_MARK value 4-3,4
 LAST_SCREEN value 4-4
 LAST value 4-4
 LAST 6-3
 Leaving FSE 3-47
 Lengthing the file 3-12
 Limit search columns 5-21
 Limiting line width 7-12
 Line
 Break 5-68
 Changing width 7-12
 Copying 3-41
 Enter 3-7; 5-4
 Join 5-70
 Marking 3-39; 6-3
 Moving 3-44
 Width 7-12
 Line editing 1-1
 Line identifiers 7-15
 LINE parameter 4-4
 Lines
 Changing number
 displayed 7-1
 Delete 3-24; 5-30,32; 6-1
 Insert 3-28,30; 5-22; 6-1
 Listing 5-71
 Mark 5-42
 Replacing 5-38
 LISB 5-14.1
 LISF 5-14.2
 LISL 5-14.3
 LIST_BACKWARD 5-14.1
 List column numbers 7-17
 LIST_FORWARD 5-14.2
 LIST_LINES 5-14.3
 Listing decks 7-15
 Listing lines 5-71
 LOCA 5-18.1

- locall 3-36; 5-15; 6-4
 - Locate
 - Line 5-12
 - Text 5-12
 - LOCATE_ALL 5-18.1
 - LOCATE_EMPTY_LINES 5-19
 - LOCATE_NEXT 5-18.2
 - LOCATE_STRING 5-18.2
 - LOCATE_TEXT 5-16
 - LOCATE_WIDE_LINES 5-20
 - LOCATE 3-32; 5-15; 6-4
 - Locating
 - All occurrences 3-36; 5-15
 - Empty lines 5-19
 - Examples 5-18
 - General description 3-32
 - Next occurrence 3-35; 5-15
 - String 3-32; 5-15
 - Text 3-32; 5-15,16
 - Wide lines 5-20
 - LOCEL 5-19
 - LOCN 5-18.2
 - locnxt 3-35; 5-15; 6-4
 - LOCS 5-18.2
 - LOCT 5-16,71
 - LOCWL 5-20
- M**
- Manual conventions 10
 - Manual organization 7
 - Manual set 7
 - Manuals
 - Ordering 12
 - Related 12
 - MARB 5-43
 - MARC 5-44
 - Mark 3-39
 - MARK_BOX 5-43
 - MARK_CHARACTER 5-44
 - MARK_LINES 5-43
 - Mark value 4-3
 - MARK value 4-4
 - MARK 5-42; 6-3
 - Marking 3-39
 - Characters 5-42; 6-3
 - Function keys 5-42
 - General description 5-42
 - Lines 5-42; 6-3
 - Subcommands 5-43
 - MARL 5-43
 - Mask character 4-8
 - Maximum value 4-3
 - MC 5-43
 - Menu row 3-24
 - MENU_ROWS parameter 7-2
 - Message line 3-4
 - middle 6-3
 - ML 5-43
 - Model 3-2; 5-2
 - MODEL parameter 7-1
 - Move 5-61; 6-5
 - Move cursor
 - Down 5-12
 - Keys 5-12
 - Left 5-12
 - Right 5-12
 - Subcommands 5-12
 - To next tab setting 5-12
 - To previous tab setting 5-12
 - To subcommand line 5-12
 - Up 5-12
 - Move screen backward one screen 6-2
 - MOVE_TEXT 5-62
 - Move to next tab 6-1
 - Moving
 - Backward 3-16
 - Cursor 2-2
 - Forward 3-16
 - From screen to screen 3-16
 - Function keys 5-61
 - General description 3-44
 - Screen by screen 5-14
 - Subcommands 5-62
 - Text 5-61
 - The cursor 5-12
 - To first line 6-3
 - To last line 6-3
 - To the end of a line 3-11
 - To the first line 3-15
 - To the last line 3-15
 - Within the file 3-12
 - Moving around the screen 3-8

Moving to the end of a line 3-11
 MOV_T 5-62
 Multipartition file 5-1,67

N

New line 2-2
 NEXT 6-2

O

Online help 3-19; 5-8; 6-2
 Open files 4-6
 Opening decks
 Closing previous deck 8-6
 General description 8-4
 Maintaining other decks 8-5
 Ordering manuals 12
 Organization 7

P

Paging 5-14; 6-2
 Painting the screen 5-6; 6-1
 POSB 5-14
 POSC 5-12
 POSF 5-14
 Position
 Saving 5-45
 POSITION_BACKWARD 5-14
 POSITION_CURSOR 5-12
 POSITION_FORWARD 5-14
 Positioning the cursor 2-2
 Printing lines in line mode 5-16
 Procedure end 9-3
 Procedure header 9-2
 Procedures 9-1
 Examples 9-10
 Functions used in 9-5
 Structure 9-1
 PROCEND 9-3
 Programmable function keys
 Defining 6-6
 Description 6-3
 Prompts 3-5,24
 Prologue 5-1

Prologue file 10-1
 Prompts
 Changing number
 displayed 7-1
 PUT_ROW 9-4
 PUTR 9-4

Q

quit 3-47
 QUIT 5-9; 8-11

R

READ_FILE 5-28,59
 REAF 5-28,59
 Realign screen 7-8
 Record type 5-1
 Record type of variable 5-67
 Redefine key settings 6-6
 REFR 5-6
 REFRESH_ROW 5-6
 Refreshing the screen 5-6; 6-1
 Related manuals 12
 REPL 5-38
 REPLACE_LINES 5-38
 REPLACE_TEXT 5-40
 Replacing
 Blocks of text 5-40
 Lines 5-38
 Reposition file 6-2
 REPT 5-40
 RESD 8-8
 RESET_DECK 8-8
 RESET_FILE 5-66
 RESF 5-66
 RESP 5-45
 RESTORE_POSITION 5-45
 Restoring the screen 5-6; 6-1
 Restoring your position 5-45; 6-2
 Return 2-2
 Reversing changes 5-65
 Rewrite screen 6-1

S

SAVE_POSITION 5-45

- Saving positions 5-45
- SAVP 5-45
- SCL manual set 7
- SCL procedures 9-1
- Screen
 - Alignment 7-8
 - Clear 6-1
 - Display 7-1
 - Display next 6-2
 - Format 3-4
 - Painting 5-6; 6-1
 - Refreshing 5-6; 6-1
 - Splitting 7-1
 - Width 5-20; 7-1,12
- Screen editing 1-1
- Screen mode 5-4
- SCREEN value 4-4
- SCU decks 7-15; 8-1
- SCU_EDITOR_EPILOG 10-1
- SCU_EDITOR_PROLOG 10-1
- SCU, stopping 8-11
- Search margins 5-21
- Searching
 - All occurrences 3-36; 5-15
 - Empty lines 5-19
 - For all occurrences 3-36
 - For text strings 3-32
 - For the next occurrence 3-35
 - Function keys 5-15
 - General description 3-32
 - Margins 5-21
 - Next occurrence 3-35; 5-15
 - String 5-15
 - Subcommands 5-15
 - Text 3-32; 5-15
 - Wides lines 5-20
- SELD 8-7
- SELECT_DECK 8-7
- SELECT_FIRST_DECK 8-7
- SELECT_LAST_DECK 8-7
- SELECT_NEXT_DECK 8-7
- Selecting editor options 7-1
- SELFD 8-7
- SELLD 8-7
- SELND 8-7
- SET_EPILOG 10-1
- SET_FUNCTION_KEY 6-6
- SET_LINE_WIDTH 7-12
- SET_LIST_OPTIONS 7-15
- SET_MASK 4-9
- SET_PARAGRAPH_
 - MARGINS 5-71
- SET_SCREEN_OPTIONS 7-1
- SET_SEARCH_MARGINS 5-21
- SET_TAB_OPTIONS 7-10
- SET_VERIFY_OPTION 7-13
- SET_WORD_
 - CHARACTERS 7-14
- SETE 10-1
- SETFK 6-6
- SETLO 7-15
- SETLW 7-12
- SETM 4-9
- SETPM 5-71
- SETSM 5-21
- SETSO 7-1
- Setting
 - Line width 7-12
 - Search margins 5-21
 - Tab character 7-10
 - Tabs 7-10
 - Your Viking 721 terminal D-1
- SETTO 7-10
- SETUP D-1
- SETVO 7-13
- SETWC 7-14
- Shift 2-2
- SHIFT CLEAR 6-1
- SHIFT DELETE 5-30; 6-1
- SHIFT ERASE 6-1
- SHIFT F1 Unmrk 3-40; 5-42; 6-3
- SHIFT F10 delwrld 5-30,68; 6-5
- SHIFT F14 Move 3-44; 6-5
- SHIFT F14 MOVE 5-61
- SHIFT F15 dedent 5-30; 6-5
- SHIFT F2 Boxmk 6-3
- SHIFT F3 LAST 6-3
- SHIFT F4 endlin 5-12; 6-3
- SHIFT F7 locnxt 5-15; 6-4
- SHIFT F8 132col 6-4
- SHIFT F9 delel 5-30; 6-4
- SHIFT INSRT 3-28; 5-22; 6-1
- Shift lock 2-2
- Size of file 7-16

- Source Code Utility 8-1
- SOURCE_CODE_UTILITY 8-2
- Split-screen editing 7-2
- SPLITS parameter 7-2
- Splitting lines 5-68; 6-5
- Splitting screen 7-1
- Standard function keys 6-2
- Starting the editor 3-1; 5-1
- Statement list 9-2
- Status 3-5; 6-4
- Status information 7-16
- STATUS parameter 4-4
- Stop a search 6-2
- STOP 5-9; 6-2
- Stopping
 - An editing session 3-47
 - Editor 8-11
 - FSE 6-4
 - Function keys 3-47; 5-9
 - Screen mode 5-10
 - SCU 8-11
 - Subcommands 3-47; 5-9
- Store current position 6-2
- Storing positions 5-45
- Storing your position 6-2
- String
 - Delete 5-36
 - Mark 5-42
 - Replace 5-40
- Strings
 - Insert 5-22
- Subcommand
 - Abbreviations 4-1
 - Line 3-4
 - Strings C-1
 - Syntax 4-1
- Subcommand syntax 4-1
- Subcommands 3-19; 5-6
 - Entering 3-19
- Submitting comments 12
- Supported terminals 2-1; 3-2

T

- Tab
 - Clear 7-11
 - Settings 7-10
- Tab backward 2-2
- Tab character 7-16
- Tab column 7-16
- Tab forward 2-2
- Tab movement 6-1
- TAB 5-12; 6-1
- Tabs 7-10
- Terminal
 - CDC 722 2-3; 11-1
 - CDC 722-30 2-3; 11-5
 - DEC VT100 2-3; 11-9
 - Heathkit H19 2-3; 11-17
 - IBM PC 2-3; 11-13
 - Viking 721 2-3; D-1
 - Zenith Z19 2-3; 11-17
 - Zenith Z29 2-3; 11-22
- Terminal model 3-2; 5-2
- Terminate input line 6-2
- Text
 - Delete 5-34
 - Enter 3-7; 5-4
 - Indenting 5-27
 - Replace 5-40
- Text formatting
 - Function keys 5-68
 - Subcommands 5-68
- Text string searching 3-32
- Trailing blanks 7-12
- Typing over 3-9

U

UNDO 5-66; 6-4
 Undoing 3-46
 UNM 5-44
 Unmark 5-42
 UNMARK 5-44
 Unmarking 3-40; 6-3
 Subcommands 5-44
 Unmrk 3-40; 5-42; 6-3
 UNTIL character 5-5
 UP 6-2
 Using other terminals in screen
 mode 11-1

V

Variable record type file 5-67
 Verify option 7-13
 VETO parameter 4-7
 Viking 721 subcommand
 strings C-1
 Viking 721 terminal 2-1; 7-1
 Viking 721 terminal settings D-1
 VT100 7-1

W

WEOP directives 5-1
 WEOP 5-67
 Width 5-20; 7-1,12
 Word
 Processing 5-68
 Words
 Allowable characters 7-14
 Delete 3-24; 5-30,36
 Insert 3-29; 5-22,26; 6-4
 Working files 4-5
 WRIF 5-57
 WRITE_FILE 5-57

Z

Zenith Z19 2-3; 7-1; 11-17
 Zenith Z29 2-3; 7-1; 11-22
 Z19 7-1; 11-17
 Z29 7-1; 11-22
 132 columns 6-4
 721 2-1; 7-1
 722 7-1; 11-1
 80 columns 6-4
 80/132 6-4

Full Screen Editor for NOS/VE 60464015 C

We would like your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who Are You?	How Do You Use This Manual?	Which Do You Also Have?
<input type="checkbox"/> Manager	<input type="checkbox"/> As an Overview	<input type="checkbox"/> SCL Language Definition
<input type="checkbox"/> Systems Analyst or Programmer	<input type="checkbox"/> To Learn the Product/System	<input type="checkbox"/> SCL System Interface
<input type="checkbox"/> Applications Programmer	<input type="checkbox"/> For Comprehensive Reference	<input type="checkbox"/> SCL Quick Reference
<input type="checkbox"/> Operator	<input type="checkbox"/> For Quick Look-up	<input type="checkbox"/> Full Screen Editor
<input type="checkbox"/> Other _____		

What programming languages do you use? _____

Which are helpful to you? Common Parameters (inside front cover) Subcommand Index (inside back cover)

Related Manuals page Character Set Glossary Other: _____

How Do You Like This Manual? Check those that apply.

Yes	Somewhat	No	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the manual easy to read (print size, page layout, and so on)?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is it easy to understand?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the order of topics logical?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Are there enough examples?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Are the examples helpful? (<input type="checkbox"/> Too simple <input type="checkbox"/> Too complex)
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the technical information accurate?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Can you easily find what you want?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Do the illustrations help you?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Does the manual tell you what you need to know about the topic?

Comments? If applicable, note page number and paragraph.

Would you like a reply? Yes No

Continue on other side

From:

Name _____	Company _____
Address _____	Date _____
_____	Phone No. _____

Please send program listing and output if applicable to your comment.



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

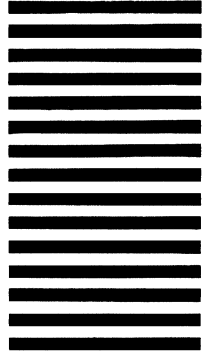
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY ADDRESSEE

GD CONTROL DATA

Publications and Graphics Division
ARH219
4201 Lexington Avenue North
Saint Paul, Minnesota 55112



FOLD
Comments (continued from other side)

Subcommand Index

A

ACTIVATE_SCREEN 3-2; 5-2
ACTS 3-2; 5-2
ALIGN_SCREEN 7-8
ALIS 7-8

B

BREAK_TEXT 5-69
BRET 5-69

C

CENL 5-73
CENTER_LINES 5-73
CLEAR_TAB 7-11
CLET 7-11
COPT 5-55
COPY_TEXT 5-55

D

DEACTIVATE_SCREEN 5-10
DEAS 5-10
DELC 5-31
DELEL 5-36
DELETE_CHARACTERS ... 5-31
DELETE_EMPTY_LINES ... 5-36
DELETE_LINES 5-32
DELETE_TEXT 5-34
DELETE_WORD 5-36
DELL 5-32
DELT 5-34
DELW 5-36
DISCN 7-17
DISES 7-16
DISP 7-16
DISPLAY_COLUMN_
 NUMBERS 7-17
DISPLAY_EDITOR_
 STATUS 7-16
DISPLAY_POSITION 7-16

E

EDID 8-5
EDIF 3-1; 5-1
EDIFD 8-5
EDIL 8-2
EDILD 8-6
EDIND 8-6
EDIT_DECK 8-5
EDIT_FILE 3-1; 5-1
EDIT_FIRST_DECK 8-5
EDIT_LAST_DECK 8-6
EDIT_LIBRARY 8-2
EDIT_NEXT_DECK 8-6
END 5-9; 8-11
ENDD 8-10
END_DECK 8-10
ENDF 5-10
END_FILE 5-10
EXCHANGE_POSITION ... 5-45
EXCHANGE_SCREEN_
 WIDTH 7-8
EXCP 5-45
EXCSW 7-8

F

FORMAT_PARAGRAPHS ... 5-72
FORP 5-72

H

HEL 5-8
HELP 5-8

I

INDENT_TEXT 5-27,37
INDT 5-27,37
INSC 5-23
INSEL 5-25
INSERT_CHARACTERS ... 5-23
INSERT_EMPTY_LINES ... 5-25

(Continued)

INSERT_LINES 5-24
INSERT_WORD 5-26
INSL 5-24
INSW 5-26

J

JOIN_TEXT 5-70
JOIT 5-70

L

LISB 5-14.1
LISF 5-14.2
LISL 5-14.3
LIST_BACKWARD 5-14.1
LIST_FORWARD 5-14.2
LIST_LINES 5-14.3
LOCA 5-18.1
LOCATE_ALL 5-18.1
LOCATE_EMPTY_LINES ... 5-19
LOCATE_NEXT 5-18.2
LOCATE STRING 5-18.2
LOCATE_TEXT 5-16
LOCATE_WIDE_LINES 5-20
LOCEL 5-19
LOCN 5-18.2
LOCS 5-18.2
LOCT 5-16
LOCWL 5-20

M

MARB 5-43
MARC 5-44
MARK_BOX 5-43
MARK_CHARACTER 5-43
MARK_LINES 5-44
MARL 5-43
MOVE_TEXT 5-62
MOVT 5-62

P

POSB 5-14
POSC 5-11
POSF 5-14

POSITION_BACKWARD 5-14
POSITION_CURSOR 5-11
POSITION_FORWARD 5-14
PUTR 9-4
PUT_ROW 9-4

Q

QUI 5-9; 8-11
QUIT 5-9; 8-11

R

READ_FILE 5-28,59
REAF 5-28,59
REFR 5-6
REFRESH_ROW 5-6
REPL 5-38
REPLACE_LINES 5-38
REPLACE_TEXT 5-40
REPT 5-40
RESD 8-8
RESET_DECK 8-8
RESET_FILE 5-66
RESF 5-66
RESP 5-45
RESTORE_POSITION 5-45

S

SAVE_POSITION 5-45
SAVP 5-45
SELD 8-7
SELECT_DECK 8-7
SELECT_FIRST_DECK 8-7
SELECT_LAST_DECK 8-7
SELECT_NEXT_DECK 8-7
SELFD 8-7
SELLD 8-7
SELND 8-7
SETE 10-1
SET_EPILOG 10-1
SETFK 6-6
SET_FUNCTION_KEY 6-6
SET_LINE_WIDTH 7-12
SET_LIST_OPTIONS 7-15

(Continued)

SETLO	7-15
SETLW	7-12
SETM	4-9
SET_MASK	4-9
SET_PARAGRAPH_	
MARGINS.....	5-71
SETPM	5-71
SET_SCREEN_OPTIONS	7-1
SET_SEARCH_MARGINS.....	5-21
SETSM	5-21
SETSO	7-1
SET_TAB_OPTIONS	7-10
SETTO	7-10
SET_VERIFY_OPTION	7-13
SETVO	7-13
SETWC	7-14
SET_WORD_	
CHARACTERS.....	7-14

U

UNDO.....	5-66
UNM	5-44
UNMARK	5-44

W

WRIF.....	5-57
WRITE FILE.....	5-57

