

---

# Standard TECO Text Editor and Corrector for the VAX, PDP-11, PDP-10, and PDP-8

**May 1990**

This manual was updated for the online version only in May 1990.

## **User's Guide and Language Reference Manual**

TECO-32 Version 40

TECO-11 Version 40

TECO-10 Version 3

TECO-8 Version 7

This manual describes the TECO Text Editor and COrrector.

It includes a description for the novice user and an in-depth discussion of all available commands for more advanced users.

General permission to copy or modify, but not for profit, is hereby granted, provided that the copyright notice is included and reference made to the fact that reproduction privileges were granted by the TECO SIG.

---

© Digital Equipment Corporation 1979, 1985, 1990 TECO SIG. All Rights Reserved.

This document was prepared using DECdocument, Version 3.3-1b.

---

# Contents

<b>Preface</b> .....	xvii
<b>Introduction</b> .....	xix
<b>Preface to the May 1985 edition</b> .....	xxiii
<b>Preface to the May 1990 edition</b> .....	xxv
<b>1 Basics of TECO</b>	
1.1 Using TECO .....	1-1
1.2 Data Structure Fundamentals .....	1-2
1.3 File Selection Commands .....	1-3
1.3.1 Simplified File Selection .....	1-3
1.3.2 Input File Specification (ER command) .....	1-4
1.3.3 Output File Specification (EW command) .....	1-4
1.3.4 Closing Files (EX command) .....	1-5
1.4 Input and Output Commands .....	1-5
1.5 Pointer Positioning Commands .....	1-5
1.6 Type-Out Commands .....	1-6
1.6.1 Immediate Inspection Commands [not in TECO-10] .....	1-7
1.7 Text Modification Commands .....	1-7
1.8 Search Commands .....	1-8
1.9 Sample Editing Job .....	1-9
1.10 Interlude .....	1-11
<b>2 Invoking TECO</b>	
2.1 Running TECO .....	2-1
2.2 Creating a new file .....	2-1
2.3 Editing an existing file .....	2-1
2.4 Switches on TECO and MAKE Commands .....	2-2
2.5 Invoking a TECO program .....	2-3
2.6 User Initialization .....	2-3
<b>3 Conventions and Structures</b>	
3.1 TECO character set .....	3-1
3.2 TECO Command Format .....	3-2
3.2.1 Numeric Arguments .....	3-2
3.2.2 Text Arguments .....	3-4
3.2.3 Colon Modifiers .....	3-5
3.3 Data Structures .....	3-5

3.3.1	Text Buffer .....	3-6
3.3.2	Q-registers .....	3-6
3.3.3	Q-register Push-down List [not in TECO-8] .....	3-7
3.3.4	Numeric Values and Flags .....	3-7

## 4 Command String Editing

4.1	Immediate Action Editing Characters .....	4-1
4.2	Immediate Action Commands .....	4-3
4.3	Introduction to Macros .....	4-4
4.4	Immediate ESCAPE-Sequence Commands [TECO-11 only] .....	4-4
4.5	Operating System Character Filters .....	4-6

## 5 Command Descriptions

5.1	File Specification Commands .....	5-1
5.1.1	File Opening Commands .....	5-1
5.1.2	File Specification Switches .....	5-2
5.1.3	File Close and Exit Commands .....	5-3
5.1.4	Secondary Stream Commands .....	5-5
5.1.5	Wildcard Commands .....	5-6
5.1.6	Direct I/O to Q-Registers .....	5-7
5.2	Page Manipulation Commands .....	5-7
5.3	Buffer Pointer Manipulation Commands .....	5-10
5.4	Text Type-Out Commands .....	5-12
5.5	Deletion Commands .....	5-13
5.6	Insertion Commands .....	5-14
5.7	Search Commands .....	5-16
5.8	Search Arguments .....	5-20
5.9	Q-Register Manipulation .....	5-23
5.10	Arithmetic and Expressions .....	5-26
5.11	Special Numeric Values .....	5-28
5.12	Command Loops .....	5-32
5.13	Branching Commands .....	5-32
5.14	Conditional Execution Commands .....	5-36
5.15	Retrieving Environment Characteristics .....	5-38
5.16	Mode Control Flags .....	5-40
5.17	Scope Commands .....	5-47
5.17.1	Video Terminal Scope Commands .....	5-47
5.17.2	Refresh Scope Commands .....	5-49
5.18	Programming Aids .....	5-49
5.18.1	Text Formatting .....	5-50
5.18.2	Comments .....	5-50
5.18.3	Messages .....	5-51
5.18.4	Tracing .....	5-51
5.18.5	Convenience Characters .....	5-51
5.18.6	Memory Expansion .....	5-52
5.18.7	Case Control .....	5-52
5.19	Manipulating Large Pages .....	5-53
5.20	Techniques and Examples .....	5-53

## **A Octal, Decimal & Hexadecimal ASCII Character Set**

## **B Error Messages**

## **C Incompatible, Obsolete, and System-Specific Commands**

C.1	Specific Features of TECO-11 . . . . .	C-1
C.2	Specific Features of RT-11 . . . . .	C-3
C.3	Specific Features of RSTS/E . . . . .	C-3
C.4	Specific Features of RSX-11/M, RSX-11/D, and IAS . . . . .	C-3
C.5	Specific Features of VAX/VMS . . . . .	C-4
C.6	Specific Features of OS/8 . . . . .	C-4
C.7	Specific Features of TOPS-10 . . . . .	C-4

## **D RT-11 Operating Characteristics**

D.1	Startup . . . . .	D-1
D.2	File Specification . . . . .	D-2
D.3	Backup Files . . . . .	D-2
D.4	EXIT and GO . . . . .	D-2
D.5	REENTER and CLOSE . . . . .	D-2
D.6	File Recovery . . . . .	D-3
D.7	System Crash Recovery . . . . .	D-3
D.8	VT11 Graphics Support . . . . .	D-3

## **E RSTS/E Operating Characteristics**

E.1	Startup . . . . .	E-1
E.2	File Specification . . . . .	E-1
E.3	Backup Files . . . . .	E-2
E.4	EXIT and GO . . . . .	E-2
E.5	ET Flag Handling . . . . .	E-2

## **F RSX-11 Operating Characteristics**

F.1	Startup . . . . .	F-1
F.2	Initialization . . . . .	F-1
F.3	File Specification . . . . .	F-2
F.4	Wild Card Lookup . . . . .	F-3
F.5	Exiting From TECO . . . . .	F-3
F.6	<CTRL/C> . . . . .	F-3
F.7	EXIT and GO . . . . .	F-3
F.8	ET Flag Handling . . . . .	F-4
F.9	File Record Format . . . . .	F-4
F.10	Command Line Processing . . . . .	F-5

## **G VAX/VMS Operating Characteristics**

G.1	Startup . . . . .	G-1
G.2	Initialization . . . . .	G-1
G.3	File Specification . . . . .	G-2
G.4	Wildcard Lookup . . . . .	G-3
G.5	Symbol Constituents . . . . .	G-3
G.6	Exiting From TECO . . . . .	G-3

G.7	<CTRL/C> . . . . .	G-3
G.8	<CTRL/Y> . . . . .	G-4
G.9	EXIT and GO . . . . .	G-4
G.10	Operating System Functions . . . . .	G-4
G.11	Flag Handling . . . . .	G-5
G.11.1	The EE flag . . . . .	G-5
G.11.2	The EJ flag . . . . .	G-5
G.11.3	The ET flag . . . . .	G-6
G.11.4	The :W flag . . . . .	G-6
G.12	Q-registers . . . . .	G-6
G.13	File Record Format . . . . .	G-6
G.14	Command Line Processing . . . . .	G-8
G.15	Help . . . . .	G-9
G.16	Installing TECO . . . . .	G-9
G.17	Callable TECO interface routines . . . . .	G-10
	TECO\$INIT . . . . .	G-12
	TECO\$EDIT . . . . .	G-13
	TECO\$POINTERS . . . . .	G-15
	TECO\$SET_DOT . . . . .	G-16
	TECO\$ADVANCE_LINE . . . . .	G-17
	TECO\$DELETE_TEXT . . . . .	G-18
	TECO\$GET_Q_REG . . . . .	G-19
	TECO\$LOAD_Q_REG . . . . .	G-20
	TECO\$MUNG . . . . .	G-21
	TECO\$LOAD_TEXT . . . . .	G-22
	TECO\$GET_TEXT . . . . .	G-23
	TECO\$DO_COMMAND . . . . .	G-24
	TECO\$START . . . . .	G-25
	TECO\$GET_VALUES . . . . .	G-26
	TECO\$SET_VALUES . . . . .	G-27

## H OS/8 Operating Characteristics

H.1	Startup . . . . .	H-1
H.2	Startup Conditions . . . . .	H-2
H.3	File Specification . . . . .	H-3
H.4	Backup Files . . . . .	H-3
H.5	EXIT and GO . . . . .	H-3
H.6	<CTRL/C> . . . . .	H-4
H.7	File Recovery . . . . .	H-4
H.8	VR12 Graphics Support . . . . .	H-5
H.9	Exceptions . . . . .	H-5
H.10	Chaining to TECO . . . . .	H-6
H.11	User Initialization . . . . .	H-6
H.12	Returned Values from TECO.INI . . . . .	H-7
H.13	TECO.TEC Support . . . . .	H-7
H.14	Overlays . . . . .	H-8
H.15	Installation Instructions . . . . .	H-9
H.16	Arithmetic Precision . . . . .	H-9
H.17	Alternate Starting Address . . . . .	H-9
H.18	VT05 Support . . . . .	H-10

## **I TOPS-10 Operating Characteristics**

I.1	STARTUP .....	I-1
I.2	Startup Conditions .....	I-2
I.3	File Specification .....	I-3
I.4	Backup Files .....	I-3
I.5	EXIT and GO .....	I-3
I.6	<CTRL/C> .....	I-3
I.7	Exceptions .....	I-4
I.8	User Initialization .....	I-4
I.9	Installation Instructions .....	I-4
I.10	TMPCOR Support .....	I-4
I.11	Q-Register Names .....	I-5
I.12	Referencing the Text Buffer as a Q-Register .....	I-5
I.13	Sharing of Q-Register Pointers .....	I-5
I.14	Editing Line Sequence Numbered Files .....	I-5
I.15	Compiler Restrictions .....	I-6

## **J Basic-Plus/Basic-Plus-2 Handling**

J.1	Purpose .....	J-1
J.2	Method .....	J-1
J.3	Input Processing .....	J-1
J.4	Output Processing .....	J-2
J.5	Form Feed Interaction .....	J-2

## **K VTEDIT—A keypad editing macro for TECO-11 and TECO-32**

K.1	Introduction .....	K-1
K.1.1	How to read this appendix .....	K-1
K.1.2	Invoking the Video Terminal Editor Macro .....	K-2
K.1.3	Interpreting the Terminal Screen .....	K-2
K.1.4	Numeric Arguments .....	K-4
K.2	Editing Text .....	K-5
K.2.1	Moving the Cursor .....	K-5
	Set Mark .....	K-6
	Jump .....	K-7
	Jump to End of Buffer .....	K-8
	Advance Line .....	K-9
	Back Line .....	K-10
	Start of Line .....	K-11
	End of Line .....	K-12
	Advance Character .....	K-13
	Back Character .....	K-14
	Down in Column .....	K-15
	Up in Column .....	K-16
	Advance Word .....	K-17
	Back Word .....	K-18
	Advance or Back Continuously .....	K-19
	Advance Screen Image .....	K-20
	Back Screen Image .....	K-21
	Back Over Last Operation .....	K-22

K.2.2	Searching Text .....	K-23
	Set Search Argument and Search Text Buffer .....	K-24
	Set Search Argument and Search through File .....	K-25
	Set Search Argument and Destructively Search File .....	K-26
	Search Again .....	K-27
K.2.3	Deleting Text .....	K-28
	Rub Out Character .....	K-29
	Delete Character .....	K-30
	Rub Out Word .....	K-31
	Delete Word .....	K-32
	Rub Out Line .....	K-33
	Delete Line .....	K-34
	Delete Rest of Line .....	K-35
	Delete or Restore Text .....	K-36
K.2.4	Inserting and Moving Text .....	K-37
	Open Line .....	K-38
	Get Q-register Contents .....	K-39
	Load Q-register .....	K-40
	Load Q-register and Delete Text .....	K-41
	Save Text .....	K-42
	Paste Text .....	K-43
	Insert ASCII Value .....	K-44
	Insert Special Character .....	K-45
K.2.5	Advancing through the Input File .....	K-46
	Advance Page .....	K-47
	Append Page or Line .....	K-48
	Append Page or Line with no Form Feeds .....	K-49
	Yank Page .....	K-50
	Inhibit Yank Protection .....	K-51
	Enable Yank Protection .....	K-52
K.2.6	Executing TECO commands .....	K-53
	Execute Q-register Macro .....	K-54
	Repeat Q-register Macro .....	K-55
	Set TECO Command and Execute .....	K-56
	Re-execute TECO Command .....	K-57
	Set TECO Command and Repeat Execution .....	K-58
	Start/End Learning Mode .....	K-59
	Execute Learned Sequence .....	K-60
K.2.7	Controlling the Video Terminal .....	K-61
	Repaint Screen .....	K-62
	Set Cursor Line .....	K-63
	Toggle See-all Display Mode .....	K-64
	Set Terminal Characteristics .....	K-65
	Enter/Exit Screen Holding Mode .....	K-66



K.2.8	Selecting Word Delimiters . . . . .	K-67
	Use Standard Words . . . . .	K-68
	Use BASIC-PLUS Words . . . . .	K-69
	Clear Delimiter Set . . . . .	K-70
	Add Word Delimiter . . . . .	K-71
	Get Next Word . . . . .	K-72
K.2.9	Exiting . . . . .	K-73
	Exit from VTEDIT . . . . .	K-74
	Exit from TECO . . . . .	K-75
	Exit from TECO Without Preserving Output . . . . .	K-76
K.2.10	Miscellaneous Commands . . . . .	K-77
	Find Front of File . . . . .	K-78
	Worry About Operating System Crashing . . . . .	K-79
	Exit from TECO Marking Position . . . . .	K-80
	Find Marked Position . . . . .	K-81
	Redisplay Keypad Layout . . . . .	K-82
	Set Left Margin . . . . .	K-83

## L Index to TECO commands and special characters

### Glossary of Obscure TECO Terms

#### Examples

1-1	Sample Editing Job . . . . .	1-9
5-1	Saving and Restoring values and/or text . . . . .	5-54
5-2	Splitting, Merging, And Rearranging Files . . . . .	5-55
5-3	Alphabetizing by Interchange Sort . . . . .	5-57

#### Tables

1	Notation . . . . .	xx
2-1	Switches on command lines . . . . .	2-2
2-2	Startup flags . . . . .	2-4
3-1	Restrictions on numeric arguments . . . . .	3-3
3-2	Default Arguments . . . . .	3-4
4-1	Immediate Action Editing Characters . . . . .	4-2
4-2	Immediate Action Commands . . . . .	4-3
4-3	Operating System Character Filters . . . . .	4-7
5-1	File Open Commands . . . . .	5-2
5-2	Switches on File Specifications . . . . .	5-2
5-3	File Close and Exit . . . . .	5-3
5-4	Secondary Stream Commands . . . . .	5-5
5-5	Wildcard Commands . . . . .	5-7
5-6	Direct I/O to Q-registers . . . . .	5-7
5-7	Page Manipulation Commands . . . . .	5-8
5-8	Buffer Pointer Manipulation Commands . . . . .	5-10

5-9	Text Type-Out Commands .....	5-12
5-10	Type-Out Time Commands .....	5-13
5-11	Text Deletion Commands .....	5-13
5-12	Text Insertion Commands .....	5-15
5-13	Search Commands .....	5-16
5-14	Bounded Search Commands .....	5-18
5-15	Search and Replace Commands .....	5-18
5-16	TECO-10 Specific Search and Replace commands .....	5-19
5-17	String Building Characters .....	5-20
5-18	Match Control Characters .....	5-21
5-19	Q-Register Loading Commands .....	5-23
5-20	Q-Register Retrieval Commands .....	5-25
5-21	Macro Invocation Commands .....	5-25
5-22	Arithmetic Operators .....	5-27
5-23	Conversion and Radix Control Commands .....	5-27
5-24	Characters Associated with Numeric Quantities .....	5-29
5-25	Branching Commands .....	5-33
5-26	Conditional Execution Commands .....	5-37
5-27	Retrieving Environment Characteristics .....	5-38
5-28	Setting Environment Information .....	5-39
5-29	Flag Manipulation Commands .....	5-40
5-30	Mode Control Flags .....	5-40
5-31	Video Terminal Watch Commands .....	5-47
5-32	Video Terminal Status Commands .....	5-48
5-33	Refresh Scope Watch Commands .....	5-49
5-34	Convenience Characters .....	5-51
5-35	Case Control Characters .....	5-52
A-1	Hexadecimal, Octal & Decimal ASCII Character Set .....	A-2
C-1	TECO-11 and TECO-32 Commands .....	C-1
C-2	String Build Constructs .....	C-3
C-3	TECO/E Commands .....	C-3
C-4	TECO/RSX Commands .....	C-4
C-5	TECO/VMS Commands .....	C-4
C-6	TECO-8 Commands .....	C-4
C-7	TECO-10 Commands .....	C-5
G-1	TECO symbols .....	G-10
L-1	Index to TECO commands .....	L-1

---

## Preface

TECO's history goes back a long time. In fact, TECO is older than many of the readers of this manual, its roots go back to 1962, when it was conceived by Daniel L. Murphy at MIT on Digital's first computer, the PDP-1.

Since then, TECO has seen many implementations and additions to its power.



---

# Introduction

TECO is a powerful **text editing language** available under most DIGITAL operating systems. TECO may be used to edit any form of ASCII text: program sources, command procedures, or manuscripts, for example. TECO is a character-oriented editor, and also offers a number of facilities for dealing with lines of text.

You can use TECO interactively, issuing commands which are to be executed as soon as they are typed in. (TECO is an **interpreter**.) In this mode, TECO can be used for tasks ranging from very simple to quite complex.

You can write sequences of commands called **macros**, which can be stored and then invoked by short commands or even single keystrokes. You can write sequences of commands to be run as TECO **programs**, and store them on disk for execution at convenience.

TECO can be used as an implementation language, as it provides powerful primitive functions for text processing, screen handling, and keyboard management. (The VTEDIT program included with some TECO distributions is an example of a full keypad editor developed in the TECO language.)

TECO correctly handles most sequential ASCII formats supported by the operating systems upon which it runs. It provides flexible pattern-matching constructs for text searching, file wildcarding, and special support for editing BASIC-PLUS/BASIC-PLUS-2 source programs. A split screen scrolling feature allows command dialogue to share the screen with an automatically-updated window into the editing buffer.

TECO's syntax is terse; commands are designed to minimize keystrokes. Also, the versatility of TECO "in all its glory" makes it complex. However, just a few commands suffice to get real work done, and a novice TECO user can begin creating and editing text files after only a few hours of instruction. More powerful features can be learned one at a time, and at leisure.

This manual presents TECO in two stages. The first part (Chapter 1, Basics of TECO) contains basic information and introduces that set of "just a few" commands.

Subsequent chapters describe the full TECO command set, including a review of the those commands presented in Chapter 1. These chapters also introduce the concept of TECO as a programming language and explain how basic editing commands may be combined into "programs" sophisticated enough to handle the most complicated editing tasks.

The early sections of this manual include few specific examples of commands, since all TECO commands have a consistent, logical format which will quickly become apparent to the beginning user. There is, however, an extensive example at the end of Chapter 1 which employs most of the commands introduced up to

that point. Students of TECO should experiment with each command as it is introduced, and then duplicate the examples on their computer.

(If a video terminal which supports split-screen scrolling such as a member of the VTx00 families is available and the TECO being used supports it, seeing both your commands and their effects in the text buffer simultaneously can make learning easier. A command such as "5,7:W" allows 5 lines for command dialogue, while "watching" the text buffer on the remainder of the screen. The 7:W command is described in Table 5–32.)

This manual is intended to be a reference manual, and except for Chapter 1, Basics of TECO, is not a tutorial. After Chapter 1, it is assumed that the reader has a general familiarity with TECO and is referring to this manual to find detailed information.

Table 1 shows the notation which is used in this manual to represent special characters.

**Table 1 Notation**

Notation	ASCII code (octal)	Name
<NULL>	0	Null
<BS>	10	Backspace
<TAB>	11	Tab
<LF>	12	Line Feed
<VT>	13	Vertical Tab
<FF>	14	Form Feed
<CR>	15	Carriage Return
<ESCAPE> or \$	33	ESCAPE or Altmode
<CTRL/x>	-	Control-x
<space>	40	Space
<DELETE>	177	Delete or Rubout
<DELIM> or `	-	See below

<DELIM> is a character used to signify the end of text strings and TECO commands typed at the terminal. (TECO presumes that you may need to deal with carriage return and line feed as ordinary characters in ASCII text, so another character must be used as TECO's text-string and command-string delimiter.)

When a command string is being executed, TECO looks for the ESCape character as its delimiter. Some newer terminals, however, no longer possess an ESCape key. Late-model TECOs are capable of recognizing a surrogate (some other, user-specified) character as signifying an ESCape when typed at the terminal. Such a character echos as accent grave—you see an ` character, and TECO receives an ESCape character. (Note that TECO programs, command files, and macros cannot use the surrogate, since it is translated to ESCape only when you type it at the terminal.) For details on choosing a surrogate for ESCape, see the EE flag or the 8192 ET bit (Section 5.16).

Throughout this manual, the term <DELIM> is used to mean whatever character you type as a text-string or command-string delimiter. The ` character is used throughout to mean whichever character is echoed for a keystroke which passes an ESCape to TECO; if no ESCape surrogate is active, you will see a \$ instead.

Control characters, <CTRL/x>, are produced by striking the CONTROL key and a character key simultaneously.

Throughout this manual, upper case characters will be used to represent TECO commands.





---

# Preface to the May 1985 edition

## Trading in the ESCape key

In the beginning, terminals had ESCape keys. (Go back far enough and the device TT: stands for might actually have existed on the system, too...) Programs wanting to treat <CR> & Co. as data needed another character to serve as command delimiter, and ESCape looked available.

Times change. Welcome to ESCape sequences, to cursor-control and function keys. Farewell to \$\$\$. Hello ``.

See the Introduction for the notation used in this manual (Table 1).

## Changes to the manual

This edition of the manual incorporates new material pertinent to the enhancements to TECO-11 between Version 36 and Version 40 (see below).

Although this manual is still not intended to be a tutorial, some "how to learn" suggestions (such as use of split-screen scrolling) were added, as were indications of how TECO can be used to build custom-designed editors.

Numerous smaller changes were also made: clarifications, new index and glossary entries, additional explanatory material and cross referrals, correction of old typographic errors, and (no doubt) addition of new ones.

## From Version 36 to Version 40 - affected section numbers

- Manual organization/additions
  - Introduction to Macros (new section), 4.3
  - Split-screen scrolling, 5.17
  - "Immediate action commands" renamed to "immediate inspection commands", 1.6.1
  - "Immediate action editing characters" (new name), 4.1
  - VAX/VMS filespec qualifiers, format/attribute table, G.13
  - Eight-bit ASCII character set table, Appendix A
  - Handling for BASIC-PLUS-x source files, Appendix J
- "Significant" software changes
  - ESCape surrogate - EE flag, 8192 ET bit, 5.16
  - Reverse paging (-P, et al, VAX/VMS only), 5.2
  - Reverse searching (-N, et al, VAX/VMS only), 5.7
  - Local Q-registers, 3.3.2
  - Local Q-register manipulation, 5.9

- Immediate ESCape-sequence commands, 4.4
- :^T, read/decode a keystroke, 5.11
- 32 ED bit controls immediate ESC-seq commands, 5.16
- Acceptance and handling of 8-bit characters, [throughout]
- Timeout of 8-bit data, and the 4096 ET bit, 5.16
- n:^T, one-shot binary-mode character timeout, 5.4
- TECO HELP facilities on VAX/VMS, G.15
- "Minor" software changes
  - ^W as immediate mode command (screen re-paint), 4.2
  - <DELIM> character as immediate inspection command, 1.6.1,4.2
  - Null tag allowed in computed GOTO, 5.13
  - nFRtext` command (replace n characters), 5.6
  - m,nFRtext` command (replace from m to n), 5.6
  - ^E (form feed flag) can now be user-set, 5.11
  - Setting 1:W changes width and O/S characteristic, 5.17
  - WRAP/NOWRAP set by 256 ET bit on VAX/VMS, 5.16
  - 128 ED bit inhibits auto-refresh in split scrolling, 5.16
  - Search matching with diacritical marks, 5.16
  - New 0:W codes for VT102 and GIGI scope types, 5.17
  - VAX/VMS handling of ^T (disabled when), 4.5

---

## Preface to the May 1990 edition

### TECO-32 goes Native

TECO for VAX/VMS started life as a **hybrid** image. The command processor and editor proper were the same as TECO-11 and ran in PDP-11 compatibility mode on VAX processors. The operating system interface and file service logic ran in VAX native mode.

With the advent of new processors without microcode support for PDP-11 compatibility mode, TECO could only be run on such processors when VAX RSX, which supplies PDP-11 instruction emulation, was present. This resulted in a dramatic decrease in performance.

Included with VMS version 5.0 is TECO-32, a brute force translation of TECO-11 to VAX native mode performed by Richard H. Murphy of Digital Equipment Corporation.

It was included into VMS V5.0 by Andrew C. Goldstein, also of Digital.

This version of the manual is a brute force translation from the May 1985 edition into Standard Document Markup Language (SDML) done by Rien Timmer, Digital. Enhancements to the manual were made to include TECO-32.

Added and changed features of TECO-32 are described in Appendix G.



## Basics of TECO

### 1.1 Using TECO

TECO may be called from command level by typing the appropriate command, followed by a carriage return:

Operating System	Command
RT-11	R TECO
OS/8	R TECO
TOPS-10	R TECO
TOPS-20	R TECO
RSTS/E	RUN TECOS:TECO -or- EDIT/TECO
RSX-11	RUN \$TEC
VAX/VMS	RUN SYSSYSTEM:TECO32

TECO will respond by printing an asterisk at the left margin to indicate that it is ready to accept user commands. At this point, you may type one or more commands.

A **TECO command** consists of one or two characters which cause a specific operation to be performed. Some TECO commands may be preceded or followed by **arguments**. Arguments may be either **numeric** or **textual**. A numeric argument is simply an integer value which can be used to indicate, for example, the number of times a command should be executed. A text argument is a string of ASCII characters which might be, for example, words of text or a file specification.

If a command requires a numeric argument, the numeric argument always precedes the command. If a command requires a text argument, the text argument always follows the command. Each text argument is terminated by a special character (usually a <DELIM> character, which TECO hears as an ESCape - see Introduction). This indicates to TECO that the next character typed will be the first character of a new command.

TECO accumulates commands as they are typed in a **command string**, and executes commands upon receipt of two consecutive <DELIM> characters. <DELIM> may be any character you select (if your TECO and operating system support user-selectable ESCape surrogates - see sections on ET and EE flags). When you type the character you have designated as <DELIM>, TECO receives an ESCape character, and an ` (accent grave) is echoed. (If you are not using an ESCape surrogate—that is, you are actually pressing an ESCape key—a dollar sign is echoed.) The **accent grave** character is used in examples throughout this manual to represent typed <DELIM>s. Note that the carriage return

## Basics of TECO

### 1.1 Using TECO

character has no special significance to TECO; only the <DELIM><DELIM> forces execution of the command string.

TECO executes command strings from left to right until either all commands have been executed or a **command error** is recognized. It then prints an asterisk to signal that additional commands may be entered.


If TECO encounters an erroneous command, it prints an error message and ignores the erroneous command as well as all commands which follow it. All error messages are of the form:

?XXX Message

where XXX is an error code and the message is a description of the error. Some error messages mention the specific character or string of characters in error. In these error messages, TECO represents the non-printing special characters as follows:

Character	Form Displayed
<TAB>	<TAB>
<LF>	<LF>
<VT>	<VT>
<FF>	<FF>
<CR>	<CR>
<ESCAPE>	<ESC>
<CTRL/x>	<^x>

Every error message is followed by an asterisk at the left margin, indicating that TECO is ready to accept additional commands. If you type a single **question mark** character after a TECO-generated error message, TECO will print the erroneous command string up to and including the character which caused the error message. This helps you to find errors in long command strings and to determine how much of a command string was executed before the error was encountered.

You can correct typing errors by hitting the DELETE key, which may be labeled DEL or RUBOUT or  on your keyboard. Each depression of the DELETE key deletes one character and echoes it on your terminal, beginning with the last character typed. If your terminal is a CRT, TECO will actually erase the deleted character from the screen. You can delete an entire command string this way, if necessary. To delete an entire line of commands, enter the character <CTRL/U>, typed by holding down the CONTROL key while depressing the "U" key.

When you are done editing, use the EX command to exit TECO, as described below in Section 1.3.4.

## 1.2 Data Structure Fundamentals

TECO considers any string of ASCII codes to be **text**. Text is broken down into units of **characters**, **lines**, and **pages**. A character is one ASCII code. A line of text is a string of ASCII codes including one **line terminator** (usually a line feed) as the last character on the line. A page of text is a string of ASCII codes including one form feed character as the last character on the page.

TECO maintains a **text buffer** in which text is stored. The buffer usually contains one page of text, but the terminating form feed character never appears in the buffer. TECO also maintains a text buffer **pointer**. The pointer is a movable position indicator which is never located directly on a character, but is always **between** characters: between two characters in the buffer, before the first character in the buffer, or after the last character in the buffer.

Line feed and form feed characters are inserted automatically by TECO. A line feed is automatically appended to every carriage return typed to TECO and a form feed is appended to the contents of the buffer by certain output commands. Additional line feed and form feed characters may be entered into the buffer as text. If a form feed character is entered into the buffer, it will cause a page break upon output; text following the form feed will begin a new page.

Finally, TECO maintains an **input file** and an **output file**, both of which are selected by the user through file specification commands. The input file may be on any device from which text may be accepted. The output file may be on any device on which edited text may be written.

TECO functions as a **pipeline** editor. Text is read from the input file into the text buffer, and is written from the buffer onto the output file. In the VAX/VMS implementation, it is possible to "back up" as well as page forward in the file being edited. In other implementations, once text has been written to the output file, it cannot be accessed again without closing the output file and reopening it as an input file.

## 1.3 File Selection Commands

Input and output files may be specified to TECO in several ways. The following sections present first a simple method for specifying files, and then more sophisticated commands that permit flexible file selection.

---

**Note**

---

All of the following file selection commands are shown with a general argument of "filespec". The actual contents of this filespec argument are operating system dependent. See the operating characteristics appendices. Examples include a mixture of file specifications from various operating systems.

---

### 1.3.1 Simplified File Selection

For most simple applications, you can use special operating system commands to specify the name of the file you wish to edit at the same time that you start up TECO.

To create a new file:

MAKE filespec

This command starts up TECO and creates the specified file for output.

To edit an existing file:

TECO filespec

This command starts up TECO and opens the specified file for editing while preserving the original file (as a backup file). It also automatically brings the first page of the file into the text buffer. These functions simulate the EB command described in Chapter 5.

## Basics of TECO

### 1.3 File Selection Commands

If any of the above commands do not seem to work on your operating system, consult the appropriate appendix for information about how to install TECO and its associated operating system commands.

#### 1.3.2 Input File Specification (ER command)

TECO will accept input text from any input device in the operating system. The input device may be specified by the text string supplied in the ER command (and which, like any text argument, is terminated by a <DELIM> character). The ER command causes TECO to open the specified file or print an error message if the file is not found. This command does not cause any portion of the file to be read into the text buffer, however. The following examples illustrate use of the ER command.

Command	Function
ERfilespec`	General form of the ER command where "filespec" is the designation of the input file. The command is terminated by a <DELIM> character.
ERPR:`	Prepare to read an input file from the paper tape reader.
ERPROG.MAC`	Prepare to read input file PROG.MAC from the system's default device.
ERDX1:PROG.FOR`	Prepare to read input file PROG.FOR from DX1:.

TECO will only keep one input and one output file open and selected at a time. The current input file may be changed by simply using the ER command to specify a new file.

It is not always necessary to specify an input file. If you want to create a file without using any previously edited text as input, you may type commands to insert the necessary text directly into the text buffer from the keyboard and, at the end of each page, write the contents of the buffer onto an output file. Since all input is supplied from the keyboard, no input file is necessary.

#### 1.3.3 Output File Specification (EW command)

TECO will write output text onto any output device in the operating system. The output file may be specified by means of the text string supplied in an EW command. If the output device is a file-structured device (for example, a disk), the file name and any extension must be supplied. If a file name is specified but no device is explicitly specified, the system's default device is assumed. The following examples illustrate use of the EW command.

Command	Function
EWfilespec`	General form of the EW command where "filespec" is the designation of the input file. The command is terminated by a <DELIM> character.
EWSYS:TEXT.LST`	Prepare to write output file TEXT.LST on SYS:.
EWPROG`	Prepare to write output file PROG on the system's default device.
ERDX1:INPUT.MAC` EWOUTPUT.MAC``	Open an input file INPUT.MAC to be found on DX1: and open an output file named OUTPUT.MAC. The double <DELIM> (echoed as ``) terminates the command string and causes the string to be executed. Note that the <DELIM> which terminates the EW command may be one of the two <DELIM>s which terminates the command string.



You do not need to specify an output file if you only want to examine an input file, without making permanent changes or corrections. In this case, the contents of the input file may be read into the text buffer page by page and examined at the terminal. Since all output is printed on the user terminal, no output file is needed.

#### 1.3.4 Closing Files (EX command)

When you are finished editing a file, use the EX command to close out the file and exit from TECO. The current contents of the text buffer and any portion of the input file that has not been read yet are copied to the output file before TECO exits. The EX command takes no arguments.

Command	Function
EX`	Write the text buffer to the current output file, move the remainder of the current input file to the current output file, close the output file, then return to the operating system.
ERFILE.MAC` EWCOPY.MAC` EX`	Open an input file FILE.MAC and open an output file named COPY.MAC, then copy all the text in the input file to the output file, close the output file, and exit from TECO.

### 1.4 Input and Output Commands

The following commands permit pages of text to be read into the TECO text buffer from an input file or written from the buffer onto an output file. Once a page of text has been written onto the output file, it cannot be recalled into the text buffer unless the output file is closed and reopened as an input file.

Command	Function
Y	Clear the text buffer, then read the next page of the input file into the buffer. Since the Y command causes the contents of the text buffer to be lost, it is not permitted if an output file is open and there is text in the buffer.
P	Write the contents of the text buffer onto the next page of the output file, then clear the buffer and read the next page of the input file into the buffer.
nP	Execute the P command n times. If n is not specified, a value of 1 is assumed.

After each Y, P, or nP command, TECO positions the pointer before the first character in the buffer.

### 1.5 Pointer Positioning Commands

The buffer pointer provides the means of specifying the location within a block of text at which insertions, deletions or corrections are to be made. The following commands permit the buffer pointer to be moved to a position between any two adjacent characters in the buffer.

## Basics of TECO

### 1.5 Pointer Positioning Commands

Command	Function
J	Move the pointer to the beginning of the buffer.
L	Move the pointer forward to a position between the next line feed and the first character of the next line. That is, advance the pointer to the beginning of the next line.
nL	Execute the L command n times, where n is a signed integer. A positive value of n moves the pointer to the beginning of the nth line following the current pointer position. A negative value moves the pointer backward n lines and positions it at the beginning of the nth line preceding the current position. If n is zero, the pointer is moved to the beginning of the line on which it is currently positioned.
C	Advance the pointer forward across one character.
nC	Execute the C command n times, where n is a signed integer. A positive value of n moves the pointer forward across n characters. A negative value of n moves the pointer backward across n characters. If n is zero, the pointer position is not changed. Remember that there are two characters, <CR> and <LF>, at the end of each line in the buffer.

These commands may be used to move the buffer pointer across any number of lines or characters in either direction; however, they will not move the pointer across a page boundary. If a C command attempts to move the pointer backward beyond the beginning of the buffer or forward past the end of the buffer, an error message is printed and the command is ignored.

If an L command attempts to exceed the page boundaries in this manner, the pointer is positioned at the boundary which would have been exceeded. Thus, in a page of 2000 lines, the command "-4000L" would position the pointer before the first character in the buffer. The command "4000L" would position the pointer after the last character in the buffer. No error message is printed in either case.

### 1.6 Type-Out Commands

The following commands permit sections of the text in the buffer to be printed out on your terminal for examination. These commands do not move the buffer pointer.

Command	Function
T	Type the contents of the text buffer from the current position of the pointer through and including the next line feed character.
nT	Type n lines, where n is a signed integer. A positive value of n causes the n lines following the pointer to be typed. A negative value of n causes the n lines preceding the pointer to be typed. If n is zero, the contents of the buffer from the beginning of the line on which the pointer is located up to the pointer is typed. This is useful for verifying the location of the buffer pointer.
HT	Type the entire contents of the text buffer.
V	Type the current line. Equivalent to the sequence "0TT".

### 1.6.1 Immediate Inspection Commands [not in TECO-10]

In addition, there are available as a convenience abbreviations of two frequently used type out commands. Each one consists of a single character, and must be the very first character typed after TECO prints its prompting asterisk. Each of these commands takes effect immediately; there is no need to follow any of these commands by any <DELIM> character. For this reason, these commands are known as "immediate" commands (see Chapter 4 for more information on immediate commands).

Command	Function
<LF>	Immediately execute an LT command. This command is issued by typing the line feed key as the first keystroke after TECO's prompt. It causes TECO to move the pointer ahead one line and then type out the new line. On terminals without a line feed key, typing <CTRL/J> has the same effect. (See also the next command.)
<DELIM>	Immediately execute an LT command.(Action identical to that of the command above.) This command is available when an ESCape surrogate is active, and causes TECO to move the pointer ahead one line and then type out the new line when a <DELIM> is the first thing typed after TECO's prompting asterisk. [RSTS/E only]
<BS>	Immediately execute a -LT command. This command is issued by typing the backspace key as the first keystroke after TECO's prompt. It causes TECO to move the pointer back one line and then type the line just moved over on the terminal. On terminals without a backspace key, typing <CTRL/H> has the same effect.

These commands are useful for "walking through" a file, examining and/or modifying lines one at a time.

## 1.7 Text Modification Commands

You can insert or delete text from the buffer using the following commands:

## Basics of TECO

### 1.7 Text Modification Commands

Command	Function
Itext`	Where "text" is a string of ASCII characters terminated by a <DELIM> character. The specified text is inserted into the buffer at the current position of the pointer. The pointer is positioned immediately after the last character of the insertion.
K	Delete the contents of the text buffer from the current position of the pointer up to and including the next line feed character.
nK	Execute the K command n times, where n is a signed integer. A positive value of n causes the n lines following the pointer to be deleted. A negative value of n causes the n lines preceding the pointer to be deleted. If n is zero, the contents of the buffer from the beginning of the line on which the pointer is located up to the pointer is deleted.
HK	Delete the entire contents of the text buffer.
D	Delete the character following the buffer pointer.
nD	Execute the D command n times, where n is a signed integer. A positive value of n causes the n characters following the pointer to be deleted. A negative value of n causes the n characters preceding the pointer to be deleted. If n is zero, the command is ignored.

Like the L and C commands, the K and D commands may not execute across page boundaries. If a K command attempts to delete text up to and across the beginning or end of the buffer, text will be deleted only up to the buffer boundary and the pointer will be positioned at the boundary. No error message is printed. A D command attempting to delete text past the end or beginning of the text buffer will produce an error message and the command will be ignored.

### 1.8 Search Commands

The following commands may be used to search the input file for a specified string of characters.

Command	Function
Stext`	Where "text" is a string of ASCII characters terminated by a <DELIM> character. This command searches the text buffer for the next occurrence of the specified character string following the current pointer position. If the string is found, the pointer is positioned after the last character on the string. If it is not found, the pointer is positioned immediately before the first character in the buffer and an error message is printed.
Ntext`	Performs the same function as the S command except that the search is continued across page boundaries, if necessary, until the character string is found or the end of the input file is reached. If the end of the input file is reached, an error message is printed. On VAX/VMS, you can "back up" into the file and continue editing, although doing so uses TECO features beyond those introduced in this chapter. Using only basic features (or a TECO implementation other than VAX/VMS), you must close the output file and reopen it as an input file before you can edit the file further.

Both the S command and the N command begin searching for the specified character string at the current position of the pointer. Therefore, neither command will locate any occurrence of the character string which precedes the

current pointer position, nor will it locate any character string which is split across a page boundary.

Both commands execute the search by attempting to match the command argument, character for character, with some portion of the buffer contents. If an N command reaches the end of the buffer without finding a match for its argument, it writes the contents of the buffer onto the output file, clears the buffer, reads the next page of the input file into the buffer, and continues the search.

## 1.9 Sample Editing Job

The following sample editing job is included to help the new user to achieve a greater understanding of the basic TECO commands. The entire terminal output from the editing run is reproduced intact, with numbers added in the left margin referring to the explanatory paragraphs which follow.

### Example 1-1 Sample Editing Job

```
1 ! * EWD1:FILE1.TXT``
2 ! * HKIMR. JOHN P. JONES
! COMPUTER ELECTRONICS CORPORATION
! BOSTON, MASSACHUSETTS
!
! DEAR MR. JONES:
!
! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
! PERTAINING TO THE NEW TECO TEXT EDITING AND CORRECTING
! PROGRAM.
!
! ENCLOSED IS A COPY OF THE TECO USERS'S GUIDE, WHICH
! SHOULD ANSWER ALL OF YOUR QUESTIONS.
!
! SINCERELY,
!
!
!
! ``
3 ! * -20LSETTS`I 02150``
! * STION`2C13DIREGARDING``
4 ! * SGUIDE`-5DIMANUAL``
! * SELY`0T``
! SINCERELY* 0KIVERY TRULY YOURS``
! * HT``
! MR. JOHN P. JONES
! COMPUTER ELECTRONICS CORPORATION
! BOSTON, MASSACHUSETTS 02150
!
! DEAR MR. JONES:
!
! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
! REGARDING THE NEW TECO TEXT EDITING AND CORRECTING
! PROGRAM.
!
! ENCLOSED IS A COPY OF THE TECO USER'S MANUAL, WHICH
! SHOULD ANSWER ALL OF YOUR QUESTIONS.
!
! VERY TRULY YOURS,
```

(continued on next page)

## Basics of TECO

### 1.9 Sample Editing Job

#### Example 1-1 (Cont.) Sample Editing Job

```
!
!
!
!
!
5 ! * EX``
! (TECO is rerun, operating system dependent)
! * ERDT1:FILE1.TXT`EWLP:``
6 ! * Y5KIMR. JAMES B. SMITH
! DATEK ASSOCIATES, INC.
! 122 MAIN STREET WEST
! AUSTIN, TEXAS
!
! DEAR MR. SMITH:
! ``
! * HT``
! MR. JAMES B. SMITH
! DATEK ASSOCIATES, INC.
! 122 MAIN STREET WEST
! AUSTIN, TEXAS
!
! DEAR MR. SMITH:
!
! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
! REGARDING THE NEW TECO TEXT EDITING AND CORRECTING
! PROGRAM.
!
! ENCLOSED IS A COPY OF THE TECO USER'S MANUAL, WHICH
! SHOULD ANSWER ALL OF YOUR QUESTIONS.
!
! VERY TRULY YOURS,
!
!
!
!
!
! * EX``
```

- 1 At this point, the user called TECO into memory. TECO responded by printing an asterisk at the left margin. The user then entered an EW command, opening an output file called "FILE1.TXT" on DT1. There is no input file. Upon receipt of the double <DELIM> (echoed as ``), TECO created the designated output file, then printed another asterisk at the left margin.
- 2 The user entered a command string consisting of two commands. The HK command cleared the text buffer (not really necessary, since it was already empty), and the I command inserted 18 lines of text into the buffer, including 8 blank lines. TECO executed these commands upon receipt of the second double <DELIM>. At this point, the buffer pointer was positioned at the end of the buffer, following the last line feed character in the text. Note that the user made an error while typing the word "MASSACHUSETTS". He typed "MASA", then realized his mistake and struck the DELETE key once to delete the second "A". TECO echoed the deleted character. The user then typed the correct character and continued the insertion.
- 3 The user typed -20L to move the pointer to the beginning of the buffer and SETTS` to position the pointer immediately after the character string "ETTS" (which terminates the word "MASSACHUSETTS"). He then used an I command to insert one space and a five-digit zip code. A second S command

positioned the pointer after the word "INFORMATION". The 2C command moved the pointer to the beginning of the next line (carriage return and line feed count two characters), and the user deleted the words "PERTAINING TO" and replaced them with the word "REGARDING".

- 4 The user continued editing by positioning the pointer after the word "GUIDE". He then deleted this word, and replaced it with the word "MANUAL". Finally, he searched for the word "SINCERELY", typed 0T to determine that the pointer was correctly positioned between the Y and the comma which follows it, and typed 0K to delete everything on the line except the comma. He then inserted "VERY TRULY YOURS" in place of the word "SINCERELY". An HT command caused the edited text to be printed at the terminal.
- 5 The command string EX`` caused the contents of the buffer to be written onto the output file and the output file to be closed. The user then reentered TECO and reopened the file "FILE1.TXT" as an input file and specified the line printer as an output file.
- 6 This command string reads the first (and only) page of "FILE1.TXT" into the buffer, deleted the first 5 lines, replaced them with a different address and salutation, then printed the contents of the buffer on the terminal for verification and finally printed the new version of the letter onto the line printer. Note that the previous version of the letter still resides in file "FILE1.TXT" on DT1.

## 1.10 Interlude

The rest of this manual is a description of TECO in all its glory. TECO is a complex editor and has many features because it has been under development for a long time. (TECO is older than some of the readers of this manual!) Do not try to understand everything the first time through. If you find that what you are reading seems hopelessly obscure, or makes no sense whatsoever, skip to the next section and come back to it some time later. It will be a while before you need all of TECO's features.

This manual is meant to be a reference manual and not a tutorial.

The commands described in this manual are those implemented in TECO-32 Version 40, TECO-11 Version 40, TECO-8 Version 7, and TECO-10 Version 3. Some of the more obscure commands may not be present under some operating systems, in which case this is indicated by a note in the form [Not in TECO-x].

This manual also describes some (but not necessarily all) of the obscure commands that belong to one implementation of TECO but not to the other implementations. Such commands are flagged by a note of the form [TECO-x only]. These commands are not to be considered part of the so-called "Standard TECO" and should not be used in applications that may be run on multiple operating systems. Consult also the appendices for commands that are very operating system dependent.





---

## Invoking TECO

### 2.1 Running TECO

To run TECO with no special options or defaults, use the same command on your operating system that you would use to run any other program. The command is system dependent but is usually something like

RUN TECO

Consult the appendix that describes your operating system's characteristics for details.

### 2.2 Creating a new file

As described in Chapter 1, most operating systems use the same command to invoke TECO for the purpose of creating a new file:

MAKE filespec

The "MAKE filespec" command takes as its single argument the file specification for the file to be created. This file specification must conform to the conventions used by your operating system. If a file by that name already exists, TECO will give you a warning message telling you that you are superseding an existing file.

The MAKE command invokes TECO and performs an effective `EWfilespec` command, as described in Chapter 1.

### 2.3 Editing an existing file

As Chapter 1 states, most operating systems use the same command to invoke TECO for the purpose of editing an existing file:

TECO filespec

The "TECO filespec" command takes as its argument the file specification for the file to be edited. The file will be opened for input and output, with back-up protection. That means that the system will save the original version of the source file (in case you 'blow' the edit). If your operating system supports file version numbers, a new version will be created. If your operating system does not support file version numbers, the original file will be preserved in a file with the same name but with a backup extension (.BAK).

## Invoking TECO

### 2.3 Editing an existing file

---

#### Note

---

On VMS systems you can set the version limit of files and directories to a maximum value, so that VMS will automatically delete the oldest versions when more files than the version limit are present. When you use the EB command to open a file (either explicitly or implicitly) TECO will open the input file and the output file. If the version limit for that file is 1, VMS will automatically mark the input file for deletion upon closure of that file. So make sure that you do not terminate the edit in any way without writing the contents of the buffers and the remainder of the input file to the output file, or your input file will be lost!

---

The TECO command invokes TECO and performs an effective EBfilespec`Y` command. Note that the first page of the file is brought into memory and that the text buffer pointer is positioned at the start of the file.

If, at any time during the edit, you discover that the edit is invalid, slowly (so TECO can respond to them) type enough <CTRL/C>s to get you back to the operating system. You will find s to get you back to the operating system. You will find that your original file has been preserved.

TECO remembers the filespec given in a MAKE or TECO command. If TECO is invoked with the command "TECO", with no filespec, it will open the file edited last (i.e., the remembered filespec).

## 2.4 Switches on TECO and MAKE Commands

The TECO and MAKE commands can take switches (qualifiers) of the form /SWITCH. These switches are described below.

**Table 2–1 Switches on command lines**

System	Switch	Meaning
RSTS/E	/SIZE:n	start with nK word editing area
	/SIZE:+n	start with nK additional words of editing area
RSTS/E — RSX-11 — VAX/VMS	/INSPECT	Do not create an output file.
	/FIND	Initially position to the position marker left in the file by the VTEDIT macro and delete the marker.
	/NOCREATE	Do not automatically create a new file if the file specified by the TECO command does not exist.
	/NOINI	Do not use TECO.INI to perform initialization.
	/NOMEMORY	Do not remember the argument to the invocation command.
	/SCROLL	Automatically enter split screen scrolling mode, using 1/4 of the screen's lines as the scrolling area (available on DEC_CRT terminals only).
	/VTEDIT	Load VTEDIT video terminal editor.

(continued on next page)

**Table 2–1 (Cont.) Switches on command lines**

System	Switch	Meaning
<b>The /SCROLL switch may take a value of the following form:</b>		
	/SCROLL:n	Enter split screen scrolling mode, using n lines for the scrolling area.
<b>The /VTEDIT switch may also take values (of the form :value)</b>		
	/VTEDIT:HOLD	Start up in hold screen mode.
	/VTEDIT:SEEALL	Start up in SEEALL mode.
<b>These values can be combined, viz.: /VTEDIT:HOLD:SEEALL.</b>		

## 2.5 Invoking a TECO program

All operating systems except TOPS-10 and TOPS-20 use the same command to let you start up execution of a TECO program (macro). This is the MUNG command.

The MUNG command has the form:

MUNG filespec

where filespec is the name of the TECO program that is to be run. If no file extension (file type) is specified, .TEC is assumed. This command executes the TECO code that appears within the specified file. It invokes TECO and performs an effective EIfilespec`` command (consult the appendices for operating-system dependent differences). The contents of the specified file should generally end with a double ESCAPE so that execution will start immediately.

Another form of this command is:

MUNG filespec,data

where "data" is any string of ASCII characters to be passed to the TECO program. This version of the MUNG command invokes TECO and issues an effective:

Idata`EIfilespec``

command. Under TECO-11, a space, tab, or a slash (/) may be used instead of the comma.

## 2.6 User Initialization

You can specify initialization commands to TECO by creating a file called TECO.INI. If, upon start-up, TECO finds a file called TECO.INI in your area, TECO executes the commands in that file. You can use TECO.INI commands to set initial values of flags and to tailor TECO to your needs. You must, however, be very careful in constructing code for your TECO.INI file: an error in this code may keep TECO from running at all!

## Invoking TECO

### 2.6 User Initialization

If you include unusual commands in your initialization file, you would be prudent to surround such commands with the `? command`. This causes TECO to type the commands out when they are executed (see Section 5.18.4). You should also print an informative message on the terminal reminding other users that this version of TECO has been customized.

Example 1:

```
?1ED?
```

The user initialization file sets the ED flag to 1 so that `^` characters in search strings have their traditional meaning (do not convert the next character to a control character). The file also causes the command to be typed out when it is executed.

Example 2:

```
0,16ED ^A[Dot preserved on failing searches]^A 13^T 10^T
```

The user initialization file causes future search string failures to preserve the pointer position. It also prints a message informing all users of this feature.

In TECO-11, the TECO.INI commands may return a value to the command processor. Such a value, if present, is interpreted as a set of bit encoded flags that control the startup processing. The following bits may be set:

**Table 2–2 Startup flags**

Value	Meaning
Value&1	Automatically load the VTEDIT macro (as if the user had typed TECO/VTEDIT).
Value&4	Inhibit use of the memory file (as if the user had typed TECO/NOMEMORY).
Value&16	Automatically load VTEDIT and start it in SEEALL mode (as if the user had typed TECO/VTEDIT:SEEALL).
Value&32	Automatically load VTEDIT and start it in HOLDSCREEN mode (as if the user had typed TECO/VTEDIT:HOLD).
Value&128	Automatically enter split screen scrolling mode (as if the user had typed TECO/SCROLL).
Value&256	Inhibit automatic creation of the output file if the input file does not exist (as if the user had typed TECO/NOCREATE).

For additional information on initialization, consult the operating system specific appendices.

---

## Conventions and Structures

### 3.1 TECO character set

TECO accepts the full 8-bit ASCII character set. (In other than TECO-11, characters typed at the terminal have their 8th bit trimmed off. If your terminal does not transmit or receive all of the ASCII codes, you can still insert the full character set into your TECO buffer, using special commands (see Section 5.6).

For more information on TECO's handling of 8-bit characters (including on terminals capable of displaying only 7-bit data), see the description of the 4096 ET bit, in Section 5.16.

When TECO is maintaining a window into the text buffer, a special facility (called "SEEALL" or "View All" mode) is available to explicitly show certain characters which normally do not print, such as <CR>, <FF>, and <TAB>. This mode is controlled by the 3:W flag, which is described in Section 5.17.

TECO command strings may be entered using upper case characters (as shown throughout this manual) or lower case characters. For example, the commands MQ, mQ, Mq, and mq are executed identically. A file containing upper and lower case text can be edited in the same way as a file containing upper case only, although this may be clumsy to do from an upper case only terminal. TECO can be set to convert lower case alphabets to upper case as they are typed in; commands to enable or disable lower case type-in will be presented in Section 5.16.

Control characters are generally echoed by TECO as a caret or up-arrow followed by the character. Some control characters, such as <CTRL/L> (form feed) and <CTRL/G> (bell) echo as the function they perform. In many cases, you can type a control character as a caret (up-arrow) followed by a character, and it will be treated as if it had been entered using the control key.

There are exceptions to the interchangeable use of the CONTROL key and the caret. When a control character is used as the delimiter of a text string (as explained in Section 3.2.2 below), it must be entered in its <CTRL/x> form. This form must also be used if the control character is the second character of a two-character command, or is being entered as an immediate action command. Since certain control characters have special meaning in text arguments, some of them (such as <CTRL/N> and <CTRL/X>), must be entered into a text string using the CONTROL key and preceded by <CTRL/Q>, ^Q, <CTRL/R>, or ^R.

## 3.2 TECO Command Format

Each TECO command consists of one or two characters.

TECO commands may be strung together (concatenated) into a command string. If a command in the string returns numeric value(s), it or they will be passed to the following command.

---

**Note**

---

Most commands which return a value are expressly designed for that purpose. A few commands (such as `n%q` and `m,n:W, q.v.`) can be used to cause an action, and also return a value. Good programming practice suggests following such a command with `<DELIM>` or `caret-[` if the returned value is not intended to affect the following command.

---

A TECO command may be preceded by one or two numeric arguments. Some TECO commands may be followed by a text argument. Many TECO commands can be modified by `"@"` (see Section 3.2.2) and/or by `":"` (see Section 3.2.3).

When TECO commands are concatenated into a command string, no delimiter is necessary between commands except for one necessary to delimit a text argument or one as suggested in the note above.

TECO commands are accumulated into a command string as they are typed. The command string is executed when it is terminated by typing two consecutive `<DELIM>` characters. TECO then executes the commands in the order in which they appear in the command string, until the string is exhausted or an error occurs. (Some TECO commands cause branching or looping, so altering the order of execution.)

TECO's `<DELIM>` character is (internally) the ESCape (octal 33). Because newer terminals use the ESCape character as part of control sequences (and some terminals no longer even possess an ESCape key), provision is made for you to designate a key on your terminal to cause an ESCape to be sent to TECO—that is, to act as an ESCape surrogate. (See the EE flag and the 8192 ET bit, Section 5.16.) Because the key you press to delimit text strings and TECO commands may be the ESCape key or may be the surrogate, this manual uses the indirect term `<DELIM>` to mean whichever one you are using. When an ESCape surrogate is active, the character you see is ```. (When no surrogate is active, you see `$`.) This manual uses the ``` character.

### 3.2.1 Numeric Arguments

Most TECO commands may be preceded by a numeric argument. Some numeric arguments must be positive; others can be negative or zero. The maximum size of any numeric argument is restricted, as summarized in the following table:

Table 3–1 Restrictions on numeric arguments

System	Signed		Unsigned
	Min	Max	Max
TECO-8	$-2^{12}+1$	$+2^{12}-1$	$2^{13}-1$
TECO-10	$-2^{34}$	$+2^{34}-1$	$2^{35}-1$
TECO-11	$-2^{15}$	$+2^{15}-1$	$2^{16}-1$
TECO-32	$-2^{31}$	$+2^{31}-1$	$2^{32}-1$

Exceeding these ranges of values can cause unpredictable results. So can using a negative argument with a command that takes only an unsigned argument.

Numeric arguments can be used in the following ways:

- Character commands such as J, C, R, and D take a single numeric argument which represents the number of characters that the command is to act on.
- Such commands as P, PW, and < perform an action that can be repeated. The numeric argument is the repetition count.
- Some commands, such as ED, ET, ^E, ^X, ES, EU, and EV, control the setting of variables called flags. When a numeric argument is specified, the value of that argument becomes the new value of the associated flag. When no numeric argument is specified, these command return the value of the associated flag.
- Line commands such as T, K, X, FB, and FC operate on lines. They take zero, one, or two numeric arguments. If one argument (n) is specified, it represents the number of lines over which the command is to have effect, beginning at the current buffer pointer position. A positive (non-zero) n affects a text running from the current pointer position to the nth following line delimiter. A negative n affects a text running from the pointer back to the beginning of the line containing the nth previous line delimiter. When n is zero, the affected text runs from the beginning of the current line to the current pointer position. When n is omitted, n = 1 is assumed.

When a line command contains two numeric arguments (m,n), these represent the pointer positions between which text is affected. Unless the description of the command says the order of these two arguments is important, they may be entered in either order.

When a command that normally takes an argument is specified with no argument, TECO executes the command in the most common or most useful way, as shown in the following table:

## Conventions and Structures

### 3.2 TECO Command Format

Table 3–2 Default Arguments

Command	Default Argument	Default Action
C	1	Advance 1 character
R	1	Back over 1 character
L	1	Advance 1 line
J	0	Jump to start of buffer
V	1	View 1 line
D	1	Delete 1 character
K	1	Kill 1 line
S, N, etc.	1	Search for first occurrence
%	1	Increment Q-register by 1
X	1	Extract one line

These default arguments reduce the number of keystrokes needed for common TECO actions.

#### 3.2.2 Text Arguments

Many TECO commands take a text (character string) argument. The string is placed immediately after the command and consists of a sequence of ASCII characters terminated by a <DELIM> character (or in the case of ! and ^A commands, by the command character). The string of ASCII characters may not include a <DELIM>, since this would terminate the string prematurely, but may include any other character. (Some characters may be difficult to enter from a terminal because they are TECO immediate action commands or because they have been filtered out by the operating system).

Examples of text arguments:

```
Sabc` Search for the string "abc"
^UAHELLO` Insert the text "HELLO" into Q-register A
OBEGIN` Branch to the tag specified by the string "BEGIN"
```

Some TECO commands require two text arguments. Each argument must be followed by a <DELIM> character, as follows:

```
FSabc`def` Replace string "abc" by "def"
```

You can include a <DELIM> character in a text string by using another format of text argument. In this alternate form, the string is delimited on both sides by any ASCII code that does not otherwise appear in the string. You signal that this format is being used by inserting an @ character before the command, as follows:

```
@ER5TEST.FOR5 Open the file "TEST.FOR" for input. The delimiter used is
"5"
@^A+Hello out there!+ Type the message "Hello out there!" on the terminal.
The delimiter is "+"
```

Unpredictable results will occur if another TECO command intervenes between an @ sign and the command that it is supposed to affect. Note that a control character used as a delimiter must be entered as <CTRL/x> (that is, in its single-keystroke form).



### 3.2.3 Colon Modifiers

The colon (:) command modifies the action of the next command. In some cases, it will cause the next command to return a value indicating whether it has failed or succeeded. A zero (0) indicates that the command has failed, while a -1 indicates that it has succeeded. The colon modifier is used this way with such commands as :ER, :EB, :EN, :S, :N, :FS, :FN, :FB, and :FC. If the next sequential command requires a positive argument, the -1 is interpreted as the largest possible positive number. In other cases, such as :Gq and :=, the colon modifier changes the meaning of the command. Unpredictable results may occur if you place a colon directly before a TECO command that does not normally accept a colon modifier.

If both the : and the @ (string delimiter) are used with the same command, they may be placed in either order.

## 3.3 Data Structures

A good way to begin the study of a programming language is to forget the commands, for the moment, and concentrate instead on the data structures. This section follows that approach, describing both the values on which TECO operates and the buffers and registers in which these values are stored.

TECO manipulates two types of data, namely,

- The character string: a sequence of zero or more ASCII characters, and
- The integer: a numeric value that may be signed or unsigned.

The text that TECO edits is, of course, a character string. Less obviously, the command string by which the user controls TECO is also a character string. The counters and indices for character string manipulation, and the repetition counts for loops are integers.

Character strings and integers have distinct internal representation and this is reflected in the design of the TECO commands. Commands designed for character strings do not work on integers and vice versa.

The data structures described in this section are frequently applied to character strings. Structure is never "built into" the data, but rather is attributed to the data by particular commands and conventions. Thus "lines" of characters are recognized by line manipulation commands, but not by character manipulation commands, which view an end-of-line character as just another ASCII code.

The following are definitions for the line and the page in TECO:

- Any character string can be divided into TECO lines by considering the line to be ended by either:
  - a line feed (octal 12)
  - a form feed (octal 14)
  - a vertical tab (octal 13) or
  - the end of the given character string
- Any character string can be divided into TECO pages by considering the page to be ended by either
  - a form feed (octal 14) or
  - the end of the given character string

These data structures are used to achieve two quite separate results: the formatting of type out and the logical division of data.

## Conventions and Structures

### 3.3 Data Structures

#### 3.3.1 Text Buffer

The main storage of TECO is the text buffer. The text buffer stores a single character string that TECO edits. A text buffer pointer is used to address text in the buffer; it is moved about by many TECO commands. The text buffer pointer never points to characters in the buffer; it is always pointing at pointer positions (between characters). The available pointer positions in the text buffer are sequentially numbered beginning with 0. Pointer position 0 is the position at the start of the buffer, just to the left of the first character. Pointer position 1 is the next position, just to the right of the first character, etc. As an example, suppose the text buffer contains the string FOOBAR. Then seven text buffer pointer positions are determined as shown by the arrows in the following figure:

```
  F O O B A R
  ↑ ↑ ↑ ↑ ↑ ↑
  0 1 2 3 4 5 6
```

Note that there are 6 characters in the buffer and that the highest numbered pointer position is 6. The pointer position number is equal to the number of characters preceding that position.

Useful definitions of "current" objects are made with respect to the text buffer pointer as follows:

1. The current character of the text buffer is the character just to the right of the pointer. If the pointer is at the end of the text buffer, there is no character to the right of the buffer pointer, and the current character does not exist.
2. The current line of the text buffer is the TECO line that contains the current character. In the special case that the pointer is at the end of the buffer, the current line is everything back to (but not including) the last end-of-line character.
3. The current page of the text buffer is the TECO page that contains the current character. In the special case that the pointer is at the end of the buffer, the current page is everything back to (but not including) the last form feed character (or the beginning of the buffer).

When the text buffer pointer is at the end of the text buffer and the last character of the buffer is an end-of-line character, then the current line is an empty string according to the definition just given. In this case, TECO performs the required operation on this empty string. For example, if the pointer is at the end of the buffer and TECO is commanded to kill (delete) the next 3 lines, then the empty string is killed 3 times. Similarly, if the pointer is at the end of the buffer and TECO is commanded to advance the pointer 5 lines, the pointer will remain at the end of the buffer. No error message will be given. In a like fashion, if the pointer is at the end of the first line of the buffer, and TECO is instructed to extract the previous 4 lines (into a Q-register), then only the first line is extracted since it is presumed to be preceded by 3 empty lines.

#### 3.3.2 Q-registers

TECO provides data storage registers, called Q-registers, which may be used to store single integers and/or ASCII character strings. Each Q-register is divided into two storage areas. In its numeric storage area, each Q-register can store one signed integer. In its text storage area, each Q-register can store an ASCII character string (which can be any text, including the important case of a TECO command string).

There are global Q-registers and (in TECO-11) local Q-registers.

In every TECO, there are 36 global Q-registers, each of which has a one-character name: A through Z and 0 through 9. The global Q-registers are available to all macro levels, including "outside" macros altogether (that is, at prompt level).

In TECO-11, there are additionally 36 local Q-registers, each of which has a two-character name: .A through .Z and .0 through .9. Effectively, there is a complete and unique set of local Q-registers available to each and every macro level, including prompt level. TECO automatically saves and restores a given macro level's local Q-registers around execution of a nested (lower level) macro. When a macro is invoked via an Mq command (where the command is not colon-modified, and q is a global Q-register), the current set of local Q-registers is saved and a new set of local Q-registers created. When the macro exits, its local Q-registers are destroyed and those of the calling macro level are restored. (If a fatal error occurs and TECO goes back to prompt level, local Q-registers from all macro levels are destroyed and prompt level's set is restored.)

(A new set of local Q-registers is not created when the command is colon-modified, as in :MA, or when the invoked macro itself resides in a local Q-register, as in M.A—see Table 5–21.) A local Q-register name can be used in any command that takes a Q-register name.

Various TECO commands allow the storing and retrieving of numeric values from the numeric storage areas of the Q-registers. Other TECO commands allow the storage and retrieval of strings from the text storage areas of the Q-registers.

### 3.3.3 Q-register Push-down List [not in TECO-8]

The Q-register pushdown list is a stack that permits the numeric and text storage areas of Q-registers to be saved (the [ command) and restored (the ] command). The command sequence [A ]B replicates the text string and numeric value from Q-register A into Q-register B. (Note that in TECO-11, macros can be written to use local Q-registers, avoiding the need to save and restore Q-registers via the pushdown list.)

### 3.3.4 Numeric Values and Flags

TECO has many special numeric values and flags which are accessible through TECO commands. Some of these values, such as the text buffer pointer, reflect the state of TECO. Others control TECO's behavior in various ways.



---

## Command String Editing

### 4.1 Immediate Action Editing Characters

While you are typing command strings at a terminal, TECO considers certain ASCII characters to have special meaning. Most of the special characters cause TECO to perform a specified function immediately, without waiting for the double `<DELIM>` which normally be required to terminate a command string. Immediate action editing characters may be entered at any point in a command string - even in the middle of a command or text argument.

Many immediate action editing characters, such as `<DELETE>` (which deletes the immediately preceding character), cannot be used as regular TECO commands. For instance, if you enter a `<DELETE>` into a command string which is to be later executed as a macro, the `<DELETE>` will not delete a character as part the execution of the macro.

Some characters, like `<CTRL/U>`, are both regular TECO commands and immediate action commands. The command string `^Uqtext`` enters the specified text into Q-register q. However, `<CTRL/U>` typed while entering a command string is an immediate action editing character which deletes the current line. Thus you cannot type a `<CTRL/U>` (or any similar sequence) directly into TECO as part of a command string. Nevertheless, `<CTRL/U>` is still a valid TECO command; should TECO encounter it in a macro or indirect file, it will have its regular TECO effect (`^Uqtext``).

Control characters used as immediate action editing characters must be entered using the CONTROL key; they will not be recognized if entered in their caret or up-arrow form.

Table 4-1 lists the immediate action editing characters and explains their functions.

These characters take immediate effect and are used to edit a command string as it is being entered:

## Command String Editing

### 4.1 Immediate Action Editing Characters

**Table 4–1 Immediate Action Editing Characters**

Character	Explanation
<DELIM><DELIM>	<p>The double &lt;DELIM&gt; character sequence tells TECO to begin execution of the command string just typed in. It inserts two &lt;ESCAPE&gt;s into the command string. (&lt;DELIM&gt; may be generated by pressing a key designated by the user as an "ESCAPE surrogate" or by pressing an actual ESCAPE key. In the former case, `` is echoed; if no ESCAPE surrogate has been designated, \$\$ is echoed. See the EE flag and the 8192 ET bit, Section 5.16, for more information.)</p> <p>The two &lt;DELIM&gt;s must be typed successively. If any other character is typed in between the two &lt;DELIM&gt;s (even if subsequently DELETED), then the two &lt;DELIM&gt;s might be treated simply as two &lt;DELIM&gt;s to be entered into the command string rather than as an immediate action command.</p> <p>If you need to enter two &lt;DELIM&gt;s into a command line, as in the case where you want to use the FSstring`` command to delete a string, you can keep TECO from recognizing `` as an immediate action command. Type &lt;DELIM&gt;&lt;SPACE&gt;&lt;DELETE&gt;&lt;DELIM&gt; and then continue entering the remainder of your command string.</p> <p>A single &lt;DELIM&gt; character can also be an immediate action command (when typed immediately after TECO's prompting asterisk, see Section 4.2. Elsewhere, a single &lt;DELIM&gt; performs no immediate action.)</p>
<DELETE>	<p>Typing a DELETE character (DEL or RUBOUT on some terminals) deletes the last character typed. DELETE can be typed repeatedly to erase multiple characters. TECO echoes the deleted character whenever a DELETE is typed, indicating to you that the character has been rubbed out. If you are doing your editing on a scope terminal, then the action of this key is different: the character that has been rubbed out will disappear from the screen of your editing terminal and the cursor will be moved back one position.</p> <p>If you delete a line feed, form feed, or vertical tab, the cursor will move up the screen and position itself at the end of the text that immediately preceded the line feed, form feed, or vertical tab.</p>
<CTRL/C>	<CTRL/C> echoes as ^C (Caret-C) and aborts the entering of the command string. The exact action of the <CTRL/C> key depends on the operating system being used (See appendices).
<CTRL/U>	<CTRL/U> causes the current line of the current command line to be deleted. TECO echoes the character as ^U followed by <CR><LF> and an asterisk prompt. If you are using a scope terminal, the visible action of typing this key is different. The current line physically disappears from the screen and the cursor is positioned back at the beginning of the line.
<CTRL/G><CTRL/G>	Typing two consecutive <CTRL/G> characters causes all commands which have been entered but not executed to be erased. (If the terminal has a bell, it will ring.) This command is used to erase an entire command string. A single <CTRL/G> character is not a special character.
<CTRL/G><space>	<CTRL/G> followed by a space causes the line currently being entered into the command string to be retyped.

(continued on next page)

## Command String Editing

### 4.1 Immediate Action Editing Characters

**Table 4–1 (Cont.) Immediate Action Editing Characters**

Character	Explanation
<CTRL/G>*	<CTRL/G> followed by an asterisk causes all the lines typed by the user from the last TECO prompt (the asterisk) to be retyped.
<CR>	Typing a carriage return enters a carriage return followed by a line feed into the command string. To enter a carriage return without a line feed, type <CR><DELETE>.
<CTRL/Z> <CTRL/Z> <CTRL/Z>	RSX-11 TECO uses the triple <CTRL/Z> command as an immediate action command. See the appropriate appendix for more details.

The <CTRL/Z> character is used as an end-of-file terminator in some contexts on some operating systems. While its presence is usually harmless in disk files, it may cause premature end of file if the file is copied to other media (e.g., paper tape).

## 4.2 Immediate Action Commands

The following commands take effect if (and only if) they are typed as the very first keystroke(s) after TECO issues its asterisk prompt.

The <LF>, <DELIM>, and <BS> commands are also described in Chapter 1, where they are introduced as "immediate inspection commands".

**Table 4–2 Immediate Action Commands**

Character	Explanation
?	If the previous command aborted because of an error, this immediate action command causes TECO to print the erroneous command string from the beginning of the current macro level up to and including the character that caused the error.
/	If TECO has just printed an error message, type this immediate action command to receive a more detailed explanation of the error. (On VAX/VMS, a HELP command is also available. See section G.13.) [Not in TECO-11]
<LF>	Typing this immediate action command, line feed, as the first keystroke after TECO's prompt causes TECO to immediately execute an LT command. This aid lets you "walk through" a file on a non-scope terminal. (If the EV flag is non-zero, then the T portion of this command is redundant and therefore is not performed.) If you are already positioned at the end of the text buffer, TECO-11 will not type out anything. [Not in TECO-10]
<DELIM>	Typing this immediate action command, <DELIM>, is synonymous with typing the <LF> immediate action command (described above). This command is available on TECOs offering the ESCAPE surrogate. [TECO-11 only]

(continued on next page)

## Command String Editing

### 4.2 Immediate Action Commands

Table 4–2 (Cont.) Immediate Action Commands

Character	Explanation
<BS>	Typing this immediate action command, backspace, (as the first keystroke after TECO's prompt) causes TECO to immediately execute a -LT command. (If the EV flag is non-zero, then the T portion of this command is redundant and therefore is not performed.) [Not in TECO-10]
^W	Typing this immediate action command, <CTRL/W>, as the first keystroke after TECO's prompt when split-screen scrolling is active causes TECO to re-paint the buffer display portion of the screen. (Split-screen scrolling is active when 7:W is non-zero.) This command is useful if the display has been "confused" by characters from a broadcast message or type-ahead. It is also useful in conjunction with ED's 128 bit (which inhibits all automatic scroll display updating). [TECO-11 only]
*q	When an asterisk followed immediately by a Q-register name (any alphanumeric character, here represented by q) is the first keystroke after TECO's prompt, TECO places the previous command string into Q-register q. [In TECO-8, only the *Z command is permitted, and TECO will automatically type the Z.] Note that since *q is itself an immediate action command, it may not be edited with other immediate action commands. In other words, you can't use DELETE to delete an incorrectly typed *. [In TECO-10, *q must be followed by <DELIM><DELIM>.]

### 4.3 Introduction to Macros

One of TECO's powerful features is the ability to execute ASCII text stored in a Q-register as a command string. This is called the "macro" facility, and a command string so stored and executed is called a "macro".

A Q-register may be loaded with a macro by any of several means: directly from the terminal (via the ^Uq command), by extraction from the text buffer (via the X command), or by these or other Q-register loading commands contained in an indirect file or even in another macro.

Particularly handy is the "\*" immediate action command, discussed in a preceding section. When you type this command as the first character following TECO's asterisk prompt, whatever immediately preceding command you just typed at the terminal is saved in a Q-register. You can then use that command again (and repeatedly) by using the Mq command.

You invoke a macro by using the Mq command, where "q" is the name of the Q-register in which the macro text has been stored.

A related facility is that of immediate ESCAPE-sequence commands, which is discussed in the following section.

### 4.4 Immediate ESCAPE-Sequence Commands [TECO-11 only]

Certain keys on many newer terminals generate ESCape sequences. TECO provides a facility by which you can cause the pressing of one of these keys at the prompting asterisk to be interpreted as an immediate command. When this facility is enabled, a key such as one of the cursor control keys can cause TECO to immediately execute commands (previously stored as a macro). The single keystroke suffices; no <DELIM><DELIM> is needed.



## Command String Editing

### 4.4 Immediate ESCAPE-Sequence Commands [TECO-11 only]

The 32 bit in the ED flag enables immediate ESCape-sequence commands. When ED's 32 bit is non-zero, and the first character TECO receives after its \* prompt is ESCape, TECO will automatically read further characters and attempt to decode an ESCape sequence. The decoded ESCape sequence is resolved into a pair of numeric values, which are then passed to the macro residing in global Q-register A.

The two numeric values are a type code and a value code. They are passed to the macro in QA by an effective <type>, <value>MA`. The macro in QA can retrieve them by code such as:

```
UA          ! Save the <value> code in QA !
@O!T0,T1,T2,T3! ! Dispatch on <type> code !
!T0!       ! Type 0 codes come here !
QA@O!V0,V1,V2...
```

The <type> and <value> codes are as follows:

#### Type 0

- ESCape followed by a control character  
The <value> code is the control character's code (0 - 31.)

#### Type 1

- ESCape followed by a letter (e.g., ESC A), –or–
- ESCape followed by [ then a letter (e.g., ESC [ A), –or–
- CSI followed by a letter (e.g., CSI A), –or–
- ESCape, O, then an uppercase letter (e.g., ESC O A)
- The <value> code is the letter's ASCII value, trimmed to 5 bits (that is, a value of 0 through 31)
- Common keys of <type> 1 are:

Key	Value
Up arrow	1
Down arrow	2
Left arrow	4
Right arrow	3
PF1	16
PF2	17
PF3	18
PF4	19
ENTER	13

#### Type 2

- ESCape followed by ? then a letter (e.g., ESC ? p), –or–
- ESCape, O, then a lowercase letter (e.g., ESC O p)
- The <value> code is the letter's ASCII value, trimmed to 5 bits (that is, a value of 0 through 31)

## Command String Editing

### 4.4 Immediate ESCAPE-Sequence Commands [TECO-11 only]

- Common keys of <type> 2 are:

Key	Value
Keypad ,	12
Keypad -	13
Keypad .	14
Keypad 0	16
Keypad 1	17
Keypad 2	18
Keypad 3	19
Keypad 4	20
Keypad 5	21
Keypad 6	22
Keypad 7	23
Keypad 8	24
Keypad 9	25

#### Type 3

- ESCape followed by [ then a digit string then ~ –or–
- CSI followed by a digit string then ~
- The <value> code is the value of the digit string.
- All of the LK201 keyboard's editing and function keys generate sequences of this format.

### 4.5 Operating System Character Filters

In general, TECO accepts any ASCII character, whether encountered in a macro, read from an indirect file, or entered directly via a command string from the terminal. (Note that TECOs other than TECO-11 strip the eighth bit on command input.) In addition, some operating systems filter out certain characters typed at a terminal and do not pass them to TECO. So that you can be aware of the possible difficulty of entering these characters directly into a TECO command string, we list them below in Table 4–3. Note that these characters are still valid characters to TECO, but may have to be entered indirectly, such as by using the `nI`` command.

---

#### Note

---

On some operating systems, the `^T` character is an immediate action command (to the operating system), and invokes a one-line status report.

Where reference is made below in Table 4–3 to this note, TECO causes the operating system's recognition of `^T` to be turned off if TECO ever asks the user for input. The original state of `^T` handling is restored upon exit. MUNGed TECO macros that never ask the user for input do not affect the operating system's handling of `^T`.

---

**Table 4–3 Operating System Character Filters**

System	Character	System's Use
RT-11	^A	VT11 support [only if GT ON]
	^B	Background control [F/B systems only]
	^E	VT11 support [only if GT ON]
	^F	Foreground control [F/B systems only]
	^O	Output control
	^Q	Terminal Synchronization
	^S	Terminal Synchronization
RSTS/E	^O	Output control
	^Q	Terminal Synchronization
	^S	Terminal Synchronization
VAX/VMS	^O	Output control
	^Q	Terminal Synchronization
	^S	Terminal Synchronization
	^T	System status (see ^T Note above)
	^X	Cancel Type-ahead
	^Y	Process Interruption
RSX-11	^O	Output control
	^Q	Terminal synchronization
	^S	Terminal synchronization
	^X	Task control [RSX-11D only]
TOPS-10	^C^C	Job interruption
	^O	Output control
	^Q	Terminal synchronization
	^S	Terminal synchronization
	^T	System status
TOPS-20	^C^C	Job interruption
	^O	Output control
	^Q	Terminal synchronization
	^S	Terminal synchronization
	^T	System status
OS/8	^B	Background control [F/B systems only]
	^F	Foreground control [F/B systems only]

(continued on next page)

## Command String Editing

### 4.5 Operating System Character Filters

**Table 4–3 (Cont.) Operating System Character Filters**

System	Character	System's Use
	^Y	Reboot indicator [F/B systems only]

---

## Command Descriptions

This chapter presents a detailed description of the full TECO command set, functionally organized. It assumes that the reader is familiar with the elementary TECO commands presented earlier.

In the sections following, the letters "m" and "n" are used in command formats to represent numerical arguments. These may be either simple integers or complex expressions. The letter "q" represents any Q-register.

### 5.1 File Specification Commands

You must specify an input file whenever you want TECO to accept text from any source other than the terminal. You must specify an output file whenever you want to make a permanent change to the input file. Input and output files are selected by means of file specification commands.

File specification formats are operating system dependent and are fully described in the operating characteristics appendices at the end of this manual.

Almost every editing job begins with at least one file specification command. Additional file specification commands may be executed during an editing job whenever required; however, TECO will keep only one input file and one output file selected at a time.

TECO-11 recognizes two input and two output "streams" called the primary and secondary streams. The primary input and output streams are initially selected when TECO is invoked. Most file selection commands, and all of the other TECO commands (page manipulation, etc.), operate on the currently selected input and/or output stream.

The following sections list all of the file specification commands. Unless otherwise noted, all of these commands leave the text buffer unchanged. Examples of some of these commands appear in Chapter 1.

#### 5.1.1 File Opening Commands

The following commands are used to open files for input and output:

## Command Descriptions

### 5.1 File Specification Commands

**Table 5–1 File Open Commands**

Command	Function
EBfilespec`	Edit Backup. This command is recommended for most editing jobs. It is used for files on file-structured devices only. It opens the specified file for input on the currently selected input stream and for output on the currently selected output stream. The EB command also keeps the unmodified file (the latest copy of the input file) available to the user; details of this process are system dependent (See appendices).
ERfilespec`	Edit Read. Opens a file for input on the currently selected input stream.
EWfilespec`	Edit Write. Opens a file for output on the currently selected output stream.
:EBfilespec`	Executes the EB command, and returns a numeric value. -1 returned indicates success: the file is open for input. A 0 indicates the specified file could not be found, and no error message is generated. Other errors (e.g., hardware errors, protection violations, etc.) generate messages and terminate command execution as usual.
:ERfilespec`	Executes the ER command, and returns a numeric value. See the :EB command, above.

#### 5.1.2 File Specification Switches

Various system-dependent switches (qualifiers) of the form /SWITCH can be used with the file specification in ER, EW, and EB commands. These switches are listed below. Consult the operating system specific appendices for further details.

In addition, on some systems switches are provided for convenient handling of BASIC-PLUS or BASIC-PLUS-2 source files which use <LF> or '&' characters for program line continuation. See Appendix J for more information.

**Table 5–2 Switches on File Specifications**

System	Switch	Meaning
OS/8	/S	Ignore end-of-file (<CTRL/Z>s) on input. (SUPER TECO mode)
RSTS/E	/B	Read and write with unfiltered 8-bit
	/n	Handle BASIC-PLUS-x line continuation
	/B+	Handle BASIC-PLUS line continuation
	/	Same as /B+
	/B2	Handle BASIC-PLUS-x line continuation
	/CLUSTER SIZE:n	Specifies output file cluster size
	/MODE:n	Use non-standard open mode
VAX/VMS,	/B2	Handle BASIC-PLUS-x line continuation
RSX-11	/CR	Implied carriage control

(continued on next page)

**Table 5–2 (Cont.) Switches on File Specifications**

System	Switch	Meaning
	/-CR	No (internal) carriage control
	/FT	FORTRAN carriage control
	/FTN	Same as /FT
	/RW	Rewind magtape before opening file
	/SH	Open the file in shared mode
	/SHR	Same as /SH
	/STM	Stream format
	/VAR	Variable format
TOPS-10	/APPEND	Append to existing output file (EW only)
	/ASCII	File is ASCII
	/GENLSN	Generate line sequence numbers
	/NOIN	Don't put user type-in into log file
	/NONSTD	Open DECTape in non-standard mode
	/NOOUT	Don't put TECO's type-out into log file (EL only)
	/OCTAL	Read file in octal
	/PROTECT:n	Specify protection code
	/SIXBIT	Read file in pure SIXBIT
	/SUPLSN	Suppress line sequence numbers

### 5.1.3 File Close and Exit Commands

The following commands are used to close files and exit from TECO:

**Table 5–3 File Close and Exit**

Command	Function
EC	Moves the contents of the text buffer, plus the remainder of the current input file on the currently selected input stream, to the current output file on the currently selected output stream; then closes those input and output files. Control remains in TECO. EC leaves the text buffer empty.
EF	Closes the current output file on the currently selected output stream. The EF command does not write the current contents of the buffer to the file before closing it.
EG`	Performs the same function as the EC command, but then exits from TECO and re-executes the last COMPIL class command (.COMPILE, .LINK, .EXECUTE, etc.) [Same as EX in TECO-11]
EGtext`	Performs the same function as the EC command, but then exits from TECO and passes "text" to the operating system as a command string to be executed (see appendices).

(continued on next page)

## Command Descriptions

### 5.1 File Specification Commands

**Table 5–3 (Cont.) File Close and Exit**

Command	Function																										
:EGcmd args`	<p>Performs operating system function "cmd" passing that function "args" as arguments. This command always returns a value as follows:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>-1</td><td>Function "cmd" successfully performed.</td></tr> <tr> <td>0</td><td>Function "cmd" is unsupported.</td></tr> <tr> <td>Other</td><td>Function "cmd" attempted, but failed. The returned value is the failure status.</td></tr> </table> <p>There are four functions that are commonly defined:</p> <table> <tr> <th>cmd</th><th>Function</th></tr> <tr> <td>INI</td><td>Locates the user's private initialization TECO macro.</td></tr> <tr> <td>LIB</td><td>Defines the user's private library of TECO macros.</td></tr> <tr> <td>MEM</td><td>Read/write TECO's last edited file memory.</td></tr> <tr> <td>VTE</td><td>Locates the user's private scope editing TECO macro.</td></tr> </table> <p>These four functions work as follows:</p> <table> <tr> <th>Command</th><th>Function</th></tr> <tr> <td>:EGcmd`</td><td>Loads Q-register * with the requested information. If there is no information (e.g., no user private scope editing TECO macro) then Q-register * is set to the null string.</td></tr> <tr> <td>:EGcmd `</td><td>Clears the information holder. Subsequent :EGcmd` commands will set Q-register * to the null string.</td></tr> <tr> <td>:EGcmd text`</td><td>Sets the information holder to "text". Subsequent :EGcmd` commands will set Q-register * to "text".</td></tr> </table>	Value	Meaning	-1	Function "cmd" successfully performed.	0	Function "cmd" is unsupported.	Other	Function "cmd" attempted, but failed. The returned value is the failure status.	cmd	Function	INI	Locates the user's private initialization TECO macro.	LIB	Defines the user's private library of TECO macros.	MEM	Read/write TECO's last edited file memory.	VTE	Locates the user's private scope editing TECO macro.	Command	Function	:EGcmd`	Loads Q-register * with the requested information. If there is no information (e.g., no user private scope editing TECO macro) then Q-register * is set to the null string.	:EGcmd `	Clears the information holder. Subsequent :EGcmd` commands will set Q-register * to the null string.	:EGcmd text`	Sets the information holder to "text". Subsequent :EGcmd` commands will set Q-register * to "text".
Value	Meaning																										
-1	Function "cmd" successfully performed.																										
0	Function "cmd" is unsupported.																										
Other	Function "cmd" attempted, but failed. The returned value is the failure status.																										
cmd	Function																										
INI	Locates the user's private initialization TECO macro.																										
LIB	Defines the user's private library of TECO macros.																										
MEM	Read/write TECO's last edited file memory.																										
VTE	Locates the user's private scope editing TECO macro.																										
Command	Function																										
:EGcmd`	Loads Q-register * with the requested information. If there is no information (e.g., no user private scope editing TECO macro) then Q-register * is set to the null string.																										
:EGcmd `	Clears the information holder. Subsequent :EGcmd` commands will set Q-register * to the null string.																										
:EGcmd text`	Sets the information holder to "text". Subsequent :EGcmd` commands will set Q-register * to "text".																										

Consult the appendices for details.

EK

Kill the current output file on the currently selected output stream. This command, which purges the output file without closing it, is useful to abort an undesired edit. Executing the EK command after an EW which is superseding an existing file leaves the old file intact. The EK command also "undoes" an EB command. (See appendices for details.)

(continued on next page)



**Table 5–3 (Cont.) File Close and Exit**

Command	Function
ELfilespec`	Open the specified file for output as a log file. Any currently open log file will be closed. If the /APPEND switch is given, future logs will append to the file (if it already exists). The default is to supersede. All type-in to TECO and all type-out from TECO goes into the log file. The log file is automatically closed by the EX and <CTRL/C><CTRL/C> TECO commands. [TECO-10 only]
EX	Performs the same function as the EC command, but then exits from TECO. For safety reasons, this command is aborted if there is text in the text buffer but no output file is open. To exit TECO after just looking at a file, use the command string HKEX.
EZfilespec`	This command is useful for outputting to magtapes and DECTapes, on which it initializes (zeros) the specified output device before switching the output to the primary output stream. In the case of a magtape, this command rewinds the magtape to load point. If the output device is a disk, this command works exactly like the EW command. [TECO-10 only]
<CTRL/C>	The <CTRL/C> (caret/C) command terminates execution of the current command string and returns control to TECO's prompt. (Under TECO-8, the <CTRL/C> command currently acts as <CTRL/C><CTRL/C>.)
<CTRL/C> <CTRL/C>	The ^C<CTRL/C> command causes an immediate abort and exit from TECO. Currently open files are not necessarily closed. See the appendices for more details. Note that the second <CTRL/C> may not be entered in up-arrow mode.

#### 5.1.4 Secondary Stream Commands

TECO-11 provides secondary input and output streams. These permit the user to have two input and two output files open at the same time, and to switch processing back and forth between them. Each stream maintains its file position independently, so that one can read from one stream (for example), switch to the other, and then switch back to the first and resume from where one left off. In addition, a separate command stream allows one to execute TECO commands from a file without disturbing either input stream.

The following commands manipulate the secondary input and output streams:

**Table 5–4 Secondary Stream Commands**

Command	Function
<b>Input commands</b>	
EP	Switches the input to the secondary input stream.
ER`	Switches the input to the primary input stream.

(continued on next page)

## Command Descriptions

### 5.1 File Specification Commands

Table 5–4 (Cont.) Secondary Stream Commands

Command	Function
<b>Output commands</b>	
EA	Switches the output to the secondary output stream.
EW`	Switches the output to the primary output stream.
<b>Indirect file commands</b>	
EIfilespec`	<p>Opens a file as an indirect command file, so that any further TECO requests for terminal input will come from this file. At end-of-file, or upon TECO's receipt of any error message, the indirect command file will be closed and terminal input will be switched back to the terminal. Note that this command only presets where input will come from; it does not "splice" the file's data into the current command string.</p> <p>While end-of-file closes the indirect command file, it does not automatically start execution of commands. Execution will begin only upon TECO's receipt of two adjacent &lt;ESC&gt;s. For instance, assume that FOO.TEC contains valid TECO commands, that it presets a return to your terminal with the customary EI`, but that it does not contain an &lt;ESC&gt;&lt;ESC&gt; pair. If you type EIFOO`, TECO will read in the contents of FOO.TEC (building a command string) and attempt to continue building the command string by reading from your terminal when end of file on FOO.TEC is encountered. Things will appear "very quiet" until it occurs to you to type &lt;DELIM&gt;&lt;DELIM&gt; and so begin execution of the (composite) command string.</p> <p>All commands encountered in the indirect file will have their normal TECO meaning (as opposed to any immediate action meaning). For example, a &lt;CTRL/U&gt; encountered in an indirect file will not erase the command line in which it occurs. Instead, it will be treated as the TECO ^Uqtext` command. The only exception to this rule is the &lt;ESC&gt;&lt;ESC&gt; command, which directs TECO to execute the preceding command string and then return to the indirect file at the point following the &lt;ESC&gt;&lt;ESC&gt;. (We can say &lt;ESC&gt; explicitly here, because &lt;DELIM&gt; can be other than ESCape only in commands typed at the terminal.)</p>
EI`	<p>If an indirect command file is active, this command will close it and resume terminal input from the terminal. Any portion of the file after a double &lt;DELIM&gt; which has not yet been read is discarded. This command has no effect if no indirect file is already open.</p>

#### 5.1.5 Wildcard Commands

TECO-11 supports wild card file processing with a set of special commands, to allow TECO programs to operate on a set of files. [Not in TECO-8 or TECO-10]

**Table 5–5 Wildcard Commands**

Command	Function
ENfilespec`	This command presets the "wild card" lookup filespec. It is only a preset; it does not open, close, or try to find any file. The "wild card" lookup is the only filespec that can contain any wild card notations. See the appendices for the allowed wild fields in each operating system.
EN`	Once the wild card lookup filespec has been preset, executing this command will find the next file that matches the preset wild card lookup filespec and will load the filespec buffer with that file's name. The G* command (see Appendix C, section C.1.1) can be used to retrieve the fully expanded filespec. When no more occurrences of the wild card filespec exist, the ?FNF error is returned.
:EN`	Executes the EN` command, and returns a numeric value. A -1 indicates that another match of the wild card filespec exists and has been loaded into the filespec buffer. A 0 indicates no more occurrences exist. No error message is generated.

The filespec argument to the file selection commands in TECO-11 can use the string building characters described in Table 5–17 (see Section 5.8). The <CTRL/E>Q\* construct, described in Appendix C, is especially useful in TECO-11.

### 5.1.6 Direct I/O to Q-Registers

TECO-10 provides commands to do I/O directly to and from the Q-registers, allowing I/O to bypass the text buffer. [Not in TECO-8 or TECO-11]

**Table 5–6 Direct I/O to Q-registers**

Command	Function
EQqfilespec`	Read specified file into Q-register q. No <NULL>s or <FF>s are removed from the file, except that trailing <NULL>s are discarded. The only switch permitted on the filespec in this command is the /DELETE switch, which causes TECO to delete the file after reading it, providing that the file is less than 500 characters long. This command supports the pseudo-device TMP:, for TMPCOR. Consult the appropriate appendix for details. [TECO-10 only]
E%qfilespec`	Create the specified file. The contents of the file will be the contents of Q-register q with no <NULL>s deleted. No switches are permitted on the filespec of this command. This command supports the pseudo-device TMP:, for TMPCOR. Consult the appropriate appendix for details. [TECO-10 only]

## 5.2 Page Manipulation Commands

The following commands permit text to be read into the text buffer from an input file or written from the buffer onto an output file.

All of the input commands listed in this table assume that the input file is organized into pages small enough to fit into available memory. If any page of the input file contains more characters than will fit into available memory, the input command will continue reading characters into the buffer until a line feed is encountered when the buffer is about 3/4 full. See the appendices for more

## Command Descriptions

### 5.2 Page Manipulation Commands

details. Special techniques for handling pages larger than the buffer capacity will be presented later in this chapter.

**Table 5–7 Page Manipulation Commands**

Command	Function
<b>APPEND commands</b>	
A	Appends the next page of the input file to the contents of the text buffer, thus combining the two pages of text on a single page with no intervening form feed character. This command takes no argument. To perform n Appends, use the n<A> construct. Note that nA is a completely different command.
:A	Equivalent to the A command except that a value is returned. -1 is returned if the append succeeded, and 0 is returned if the append failed because the end-of-the-input-file had previously been reached (^N flag is -1 at start of this command). [Not in TECO-8]
n:A	Appends n lines of text from the input file to the contents of the text buffer. The value of n must not be negative. A value is returned indicating whether or not there were in fact n lines remaining in the input file. -1 is returned if the command succeeded. 0 is returned if end-of-file on the input file was encountered before all n lines were read in. Note that the command can succeed and yet read in fewer than n lines in the case that the text buffer fills up. [Not in TECO-8]

(continued on next page)

## Command Descriptions

### 5.2 Page Manipulation Commands

**Table 5–7 (Cont.) Page Manipulation Commands**

Command	Function
<b>PAGE Commands</b>	
P	Writes the contents of the buffer onto the output file, then clears the buffer and reads the next page of the input file into the buffer. A form feed is appended to the output file if the last page read in (with a P, Y, or A command) was terminated with a form feed.
:P	Same as the P command except that a value is returned. -1 is returned if the command succeeded. 0 is returned if the command failed because the end-of-file on the input file had been reached prior to the initiation of this command. The command string <:P;> takes you to end of file. [Not in TECO-8]
nP	Executes the P command n times, where n must be a non-zero positive integer. In TECO-32 n can be a negative integer (and -P means -1P). A -nP command will back up n pages.
n:P	Executes the :P command n times, where n must be a non-zero positive integer. In TECO-32 n can be a negative integer (and -:P means -1:P). This command returns 0 when another page could not be backed up because beginning of file had been reached. [Not in TECO-8]
PW	Write the contents of the buffer onto the output file and append a form feed character. The buffer is not cleared and the pointer position remains unchanged.
nPW	Executes the PW command n times, where n must be a non-zero positive integer.
m,nPW	Writes the contents of the buffer between pointer positions m and n onto the output file. m and n must be positive integers. A form feed is not appended to this output, nor is the buffer cleared. The pointer position remains unchanged.
m,nP	Equivalent to m,nPW.
HPW	Equivalent to the PW command except that a form feed character is not appended to the output.
HP	Equivalent to HPW.

(continued on next page)

## Command Descriptions

### 5.2 Page Manipulation Commands

**Table 5–7 (Cont.) Page Manipulation Commands**

Command	Function
<b>YANK commands</b>	
Y	Clears the text buffer and then reads the next page of the input file into the buffer. Because the Y command DESTROYS the current text buffer, its use can result in the loss of data, and is subject to "Yank protection" (see ED flag in Section 5.14.)
-Y	Clears the text buffer and then reads the previous page of the file being edited back into the buffer. Because the Y command DESTROYS the current text buffer, its use can result in the loss of data, and is subject to "Yank protection" (see ED flag in Section 5.14). [TECO-11 on VAX/VMS only]
:Y	Same as the Y command but a value is returned. -1 is returned if the Yank succeeded. 0 is returned if the Yank failed because the end-of-file had been reached on the input file prior to the initiation of this command. [Not in TECO-8]
:-Y	Same as the -Y command but a value is returned. -1 is returned if the Yank succeeded. 0 is returned if the Yank failed because the beginning of file had been reached on the file being edited prior to the initiation of this command. [TECO-32 only]
EY	Same as the Y command, but its action is always permitted regardless of the value of the Yank Protection bit in the ED flag. Remember that Yank DESTROYS the current buffer; there's no way to get it back!
:EY	Same as the :Y command, but its action is always permitted regardless of the value of the Yank protection bit in the ED flag. Remember that Yank DESTROYS the current buffer; there's no way to get it back! [Not in TECO-8]

### 5.3 Buffer Pointer Manipulation Commands

Table 5–8 describes all of the buffer pointer manipulation commands. These commands may be used to move the pointer to a position between any two characters in the buffer, but they will not move the pointer across a buffer boundary. If any R or C command attempts to move the pointer backward beyond the beginning of the buffer or forward past the end of the buffer, the command is ignored and an error message is printed. If any L command attempts to exceed the buffer boundaries, the pointer is positioned at the boundary which would have been exceeded and no error message is printed.

**Table 5–8 Buffer Pointer Manipulation Commands**

Command	Function
<b>CHARACTER commands</b>	
C	Advances the pointer forward across one character.
nC	Executes the C command n times. If n is positive, the pointer is moved forward across n characters. If n is negative, the pointer is moved backward across n characters. If n is zero, the pointer position is not changed.

(continued on next page)

## Command Descriptions

### 5.3 Buffer Pointer Manipulation Commands

**Table 5–8 (Cont.) Buffer Pointer Manipulation Commands**

Command	Function
<b>CHARACTER commands</b>	
n:C	Same as nC except that a value is returned. If the command succeeded, -1 is returned. If the command failed, the pointer does not move and a value of 0 is returned. [TECO-10 only]
:C	Equivalent to 1:C.
-C	Equivalent to -1C.
<b>JUMP commands</b>	
J	Moves the pointer to a position immediately preceding the first character in the buffer. Equivalent to 0J.
nJ	Moves the pointer to a position immediately following the nth character in the buffer.
ZJ	Moves the pointer to a position immediately following the last character in the buffer.
n:J	Same as the nJ command except that if pointer position n is outside of the buffer, the pointer does not move and a value of 0 is returned. If the command succeeded, a value of -1 is returned. [TECO-10 only]
<b>LINE commands</b>	
L	Advances the pointer forward across the next line terminator (line feed, vertical tab, or form feed) and positions it at the beginning of the next line.
nL	Executes the L command n times. A positive value of n advances the pointer to the beginning of the nth line following its current position. A negative value of n moves the pointer backwards to the beginning of the nth complete line preceding its current position. If n is zero, the pointer is moved to the beginning of the line on which it is currently positioned.
-L	Equivalent to -1L.
<b>REVERSE commands</b>	
R	Moves the pointer backward across one character.
nR	Executes the R command n times. If n is positive, the pointer is moved backward across n characters. If n is negative, the pointer is moved forward across n characters. If n is zero, the position of the pointer is not changed.
-R	Equivalent to -1R.
n:R	Same as the nR command except that a value is returned. If the command succeeded, then a value of -1 is returned. If the command failed, then the buffer pointer is not moved and a value of 0 is returned. [TECO-10 only]
:R	Equivalent to 1:R.

## Command Descriptions

### 5.4 Text Type-Out Commands

### 5.4 Text Type-Out Commands

Table 5–9 describes the commands used to type out part or all of the contents of the buffer for examination. These commands do not move the buffer pointer.

**Table 5–9 Text Type-Out Commands**

Command	Function
T	Types out the contents of the buffer from the current position of the buffer pointer through and including the next line terminator character.
nT	Types n lines. If n is positive, types the n lines following the current position of the pointer. If n is negative, types the n lines preceding the pointer. If n is zero, types the contents of the buffer from the beginning of the line on which the pointer is located up to the pointer.
-T	Equivalent to -1T.
m,nT	Types out the contents of the buffer between pointer positions m and n.
..,+nT	Types out the n characters immediately following the buffer pointer. n should be greater than zero.
..-n,T	Types the n characters immediately preceding the buffer pointer. n should be greater than zero.
n^T	Types out to the terminal the character whose ASCII value is n. Whatever normal type-out conversions may currently be in effect and applicable (such as translation of control characters to up-arrow format) are done. The value of n is used modulo 256.
n:^T	Outputs to the terminal the character whose ASCII value is n. Output is done in "one-shot" binary mode; no type-out translations are done. The value of n is used modulo 256. [TECO-11 only]
HT	Types out the entire contents of the buffer.
V	Types out the current line. Equivalent to 0TT.
nV	Types out n-1 lines on each side of the current line. Equivalent to 1-nTnT. [Not in TECO-8]
m,nV	Types out m-1 lines before and n-1 lines after the current line. [Not in TECO-8]
^Atext<CTRL/A>	Types "text" on the terminal. While the command may begin with <CTRL/A> or Caret/A, the closing character must be a <CTRL/A>. A numeric argument must not be specified with this command.
@^A/text/	Equivalent to the ^A command except that the text to be printed may be bracketed with any character. This avoids the need for the closing <CTRL/A>.

You may stop or delay the output of any type-out command by typing certain special characters at the keyboard while TECO is typing out at the terminal (via a T, V, ^A, or :G command). These characters are described in the table below:



**Table 5–10 Type-Out Time Commands**

Character	Function
<CTRL/O>	Stops the terminal output of the current command string. TECO continues to run and to send characters to the terminal, however, these characters are suppressed from actually printing on the terminal. You can resume printing characters by typing another <CTRL/O> while type-out is being suppressed. TECO cancels this suppression the next time that it prompts for command string input. A TECO macro can cancel the effect of any <CTRL/O> by setting the 16's bit in the ET flag (see Section 5.16).
<CTRL/S>	Freezes the terminal output of the current command string. TECO stops running the next time it tries to output a character to your terminal, and waits for you to type a <CTRL/Q> to indicate that output should resume.
<CTRL/Q>	Causes TECO to resume any type-out that was frozen via use of the <CTRL/S> command described above. This character has this effect only while typout is frozen. Striking any key other than <CTRL/Q> or <CTRL/C> while type-out is frozen will have unpredictable results; consult the appropriate operating system manual.

Note that <CTRL/O>, <CTRL/Q>, and <CTRL/S> are legal TECO commands as well. When TECO is not typing on the terminal (when you are entering a command string, for example), these characters do not have the effect described above. They may be entered into your command string just like any other control character (except under operating systems that filter out these characters, see Section 4.5).

## 5.5 Deletion Commands

Table 5–11 summarizes the text deletion commands, which permit deletion of single characters, groups of adjacent characters, single lines, or groups of adjacent lines.

**Table 5–11 Text Deletion Commands**

Command	Function
<b>DELETE commands</b>	
D	Delete the first character following the current position of the buffer pointer.
nD	Execute the D command n times. If n is positive, the n characters following the current pointer position are deleted. If n is negative, the n characters preceding the current pointer position are deleted. If n is zero, the command is ignored.
-D	Equivalent to -1D.
m,nD	Equivalent to m,nK. [TECO-11 only]
n:D	Same as nD but returns a value (-1 if command succeeds, 0 if command failed because the range of characters to be deleted fell outside the text buffer). [TECO-10 only]

(continued on next page)

## Command Descriptions

### 5.5 Deletion Commands

Table 5–11 (Cont.) Text Deletion Commands

Command	Function
<b>DELETE commands</b>	
FDtext`	Search for the specified text string and delete it. (See search commands in Section 5.7.) [TECO-10 only]
@FD/text/	Equivalent to FDtext` except that the <DELIM> character is not necessary.
FR`	Equivalent to -nD where n is the length of the last insert, get or search command. See the description of the FRtext` command in Section 5.6 for more details.
@FR//	Form of the FR` command that does not require use of the <DELIM> character.
<b>KILL commands</b>	
K	Deletes the contents of the buffer from the current position of the buffer pointer through and including the next line terminator character.
nK	Executes the K command n times. If n is positive, the n lines following the current pointer position are deleted. If n is negative, the n lines preceding the current pointer position are deleted. If n is zero, the contents of the buffer from the beginning of the line on which the pointer is located up to the pointer is deleted. It is not an error if more lines are specified than occur when a boundary of the text buffer is encountered.
-K	Equivalent to -1K.
m,nK	Deletes the contents of the buffer between pointer positions m and n. The pointer moves to the point of the deletion. The ?POP error message (or its equivalent) is issued if either m or n is out of range.
FKtext`	Executes a Stext` command then deletes all the text from the initial pointer position to the new pointer position. [TECO-10 only]
@FK/text/	Equivalent to FKtext` except that "text" may contain any character, including <ESCAPE>, other than the delimiter (shown here as /). [TECO-10 only]
HK	Deletes the entire contents of the buffer.

### 5.6 Insertion Commands

Table 5–12 lists all of the text insertion commands. These commands cause the string of characters specified in the command to be inserted into the text buffer at the current position of the buffer pointer. Following execution of an insertion command, the pointer will be positioned immediately after the last character of the insertion.

The length of an insertion command is limited primarily by the amount of memory available for command string storage. During normal editing jobs, it is most convenient to limit insertions to about 10 or 15 lines each. When command string space is about to run out, TECO will ring the terminal's bell after each character that is typed. From the time you hear the first warning bell, you have 10 characters to type in order to clean up your command line. Attempting to enter too many characters into the current command string causes unpredictable

results to occur and should be avoided. Use the DELETE key to shorten the command to permit its termination.

As explained above in Chapter 4, certain characters are filtered out by the operating system and/or may perform special functions, and some characters are immediate action commands and have special effect. If you want to insert such characters into the text buffer, use the nI` command described in the following table. It will insert any ASCII character into the buffer, including the special characters that could not ordinarily be typed at a terminal.

**Table 5–12 Text Insertion Commands**

Command	Function
Itext`	Where "text" is a string of ASCII characters terminated by a <DELIM>. The specified text string is entered into the buffer at the current position of the pointer, with the pointer positioned immediately after the last character of the insertion.
nI`	This form of the I command inserts the single character whose ASCII code is n into the buffer at the current position of the buffer pointer. (n is taken modulo 256 in TECO-11, modulo 128 in other TECOs.) nI` is used to insert characters that are not available on the user's terminal or special characters such as DELETE which may not be inserted from a terminal with the standard I command.
@I/text/	Equivalent to the I command except that the text to be inserted may contain ESCAPE characters. A delimiting character (shown as a slash here) must precede and follow the text to be inserted, as described in Section 3.2.2 above.
n@I//	Equivalent to the nI` command, but does not require the <DELIM> character.
<TAB>text`	This command is equivalent to the I command except that the <TAB> is part of the text which is inserted into the buffer.
FRtext`	Equivalent to "-nDItext`", where "n" is obtained from the most recent occurrence of the following: (a) the length of the most recent string found by a successful search command, (b) the length of the most recent text string inserted (including insertions from the FS, FN, or FR commands), or (c) the length of the string retrieved by the most recent "G" command. In effect, the last string inserted or found is replaced with "text", provided that the pointer has not been moved. After execution of this command, the buffer pointer is positioned immediately after "text". [Not in TECO-8]
@FR/text/	Equivalent to "FRtext`", except that "text" may contain ESCAPE characters. [Not in TECO-8]
nFRtext`	Equivalent to "nDItext`". If n is less than 0, characters preceding the buffer pointer are deleted and replaced with "text". If n is greater than 0, characters following the buffer pointer are deleted and replaced with "text". In either case, the buffer pointer is left positioned immediately after "text" upon completion of this command. [TECO-11 only]
n@FR/text/	Equivalent to "nFRtext`", except that "text" may contain ESCAPE characters. [TECO-11 only]

(continued on next page)

## Command Descriptions

### 5.6 Insertion Commands

**Table 5–12 (Cont.) Text Insertion Commands**

Command	Function
m,nFRtext`	Equivalent to "m,nDItext`". The characters between buffer positions m and n are deleted and replaced with "text". The buffer pointer is left positioned immediately after "text" upon completion of this command. [TECO-11 only]
m,n@FR/text/	Equivalent to "m,nFRtext`", except that "text" may contain ESCAPE characters. [TECO-11 only]

### 5.7 Search Commands

In many cases, the easiest way to position the buffer pointer is by means of a character string search. Search commands cause TECO to scan through text until a specified string of characters is found, and then position the buffer pointer at the end of the string. A character string search begins at the current position of the pointer. It proceeds within the current buffer in a forward or a reverse direction or through the file in a forward direction. Specifying a negative numeric argument to the search command causes the search to proceed backwards from the pointer.

Your last explicitly specified search string is always remembered by TECO. If a search command is specified with a null search string argument, the last explicitly defined search string will be used. This saves having to retype a complex or lengthy search string on successive search commands.

Normally searches are "unbounded" - they search from the current position to the end of the text buffer (or in the case of backwards searches, until the beginning of the buffer). A bounded search, however, will only search from the current position to the specified bound limit. If the search string is found within the bound limits, the pointer is positioned immediately after the last character in the string. If the string cannot be found, the pointer is left unchanged.

A special case of the bounded search occurs when the upper and lower bound limits are the same. In such a case, the search command is called an anchored search, and is used to compare the search argument against the character string immediately following the text buffer pointer.

TECO-8 does not permit backward, bounded, or anchored searches. This is a general property and will not be specifically mentioned again in the following tables.

**Table 5–13 Search Commands**

Command	Function
Stext`	Where "text" is a string of characters terminated by a <DELIM>. This command searches the text buffer for the next occurrence of the specified character string following the current position of the buffer pointer. If the string is found, the pointer is positioned after the last character in the string. If it is not found, the pointer is positioned immediately before the first character in the buffer (i.e., a 0J is executed) and an error message is printed.

(continued on next page)

Table 5–13 (Cont.) Search Commands

Command	Function
<code>nStext`</code>	This command searches for the nth occurrence of the specified character string, where n is greater than zero. It is identical to the S command in other respects.
<code>-nStext`</code>	Identical to "nStext`" except that the search proceeds in the reverse direction. If the string is not found, the pointer is positioned immediately before the first character in the buffer and an error message is printed. If the pointer is positioned at the beginning of or within an occurrence of the desired string, that occurrence is considered to be the first one found. Upon successful completion, the pointer is positioned after the last character in the string found.
<code>-Stext`</code>	Equivalent to <code>-1Stext`</code> .
<code>Ntext`</code>	Performs the same function as the S command except that the search is continued across page boundaries, if necessary, until the character string is found or the end of the input file is reached. This is accomplished by executing an effective P command after each page is searched. If the end of the input file is reached, an error message is printed and it is necessary to close the output file and re-open it as an input file before any further editing may be done on that file. The N command will not locate a character string which spans a page boundary.
<code>-Ntext`</code>	Performs the same function as the -S command except that the search is continued (backwards) across page boundaries, if necessary, until the character string is found or the beginning of the file being edited is reached. [TECO-32 only]
<code>-nNtext`</code>	This command searches (backwards) for the nth occurrence of the specified character string. It is identical to the -N command in other respects. [TECO-32 only]
<code>nNtext`</code>	This command searches for the nth occurrence of the specified character string, where n must be greater than zero. It is identical to the N command in other respects.
<code>_text`</code>	The underscore command is identical to the N command except that the search is continued across page boundaries by executing effective Y commands instead of P commands, so that no output is generated. Since an underscore search can result in the loss of data, it is aborted under the same circumstances as the Y command (see the ED flag in Section 5.16). Note that underscore is backarrow on some terminals.
<code>n_text`</code>	This command searches for the nth occurrence of the specified character string, where n must be greater than zero. It is identical to the _ command in other respects. In TECO-32 n can be negative, in which case the search proceeds backwards through each text buffer and through the file being edited. It terminates upon the correct search string match and/or beginning of file. A -n:_ command returns 0 at beginning of file.
<code>E_text`</code>	Same as <code>_text`</code> command except that effective EY (rather than Y) commands are used. Thus, this command is never aborted and is not controlled by the Yank protection bit in the ED flag. Remember that Yank DESTROYS the current buffer; there's no way to get it back!

(continued on next page)

## Command Descriptions

### 5.7 Search Commands

**Table 5–13 (Cont.) Search Commands**

Command	Function
nE_text`	Same as n_text` command except that effective EY (rather than Y) commands are used.

**Table 5–14 Bounded Search Commands**

Command	Function
m,nStext`	System specific command. Consult Appendix C.
m,nFBtext`	Performs the same function as the nStext` command, but m and n (inclusive) serve as bounds for the search. In order for a search to be successful, the first character to match must occur between buffer pointer positions m and n. The string that is matched is permitted to extend beyond the search limits specified, provided that it begins within bounds. If m<n, then the search proceeds in a forwards direction. If m>n, then the search proceeds in the reverse direction.
nFBtext`	Performs a bounded search over the next n lines. If n is positive, the search proceeds forward over the next n lines; if n is negative the search proceeds backwards over the n preceding lines; if n is zero, the search proceeds backwards over the portion of the line preceding the pointer.
FBtext`	Equivalent to 1FBtext`.
-FBtext`	Equivalent to -1FBtext`.
::Stext`	Compare command. The ::S command is not a true search. If the characters in the buffer immediately following the current pointer position match the search string, the pointer is moved to the end of the string and the command returns a value of -1; i.e., the next command is executed with an argument of -1. If the characters in the buffer do not match the string, the pointer is not moved and the command returns a value of 0. Identical to "...FBtext`".

The search and replace commands listed below perform equivalent functions to the search commands listed next to them, but then delete "text1" and replace it with "text2".

**Table 5–15 Search and Replace Commands**

Search & Replace	Search Command
FStext1`text2`	Stext1`
nFStext1`text2`	nStext1`
FNtext1`text2`	Ntext1`
nFNtext1`text2`	nNtext1`
F_text1`text2`	_text1`
[not in TECO-10]	

(continued on next page)

**Table 5–15 (Cont.) Search and Replace Commands**

Search & Replace	Search Command
nF_text1`text2` [Not in TECO-10]	n_text1`
FCtext1`text2`	FBtext1`
nFCtext1`text2`	nFBtext1`
m,nFCtext1`text2`	m,nFBtext1`

In addition, the four following commands can be used on TECO-10:

**Table 5–16 TECO-10 Specific Search and Replace commands**

Command	Function
FDtext`	Identical to the "FStext`" command. [TECO-10 only]
nFDtext`	Identical to the "nFStext`" command. [TECO-10 only]
nFKtext`	Searches for the nth following occurrence of "text" and then deletes all characters in the text buffer between the pointer positions before and after the search. [TECO-10 only]
FKtext`	Equivalent to 1FKtext` [TECO-10 only]

The FS, F\_, and FN commands above can also be reverse searches (n<0) or bounded searches (m,n argument). A reverse F\_ or FN acts like a reverse S; that is the search terminates when the beginning of the text buffer is encountered.

(In TECO-32 -nFN searches backwards through each text buffer and through the file being edited. It terminates upon the correct search string match and/or beginning of file. A colon modified reverse search returns 0 at beginning of file.)

If a search command is entered without a text argument, TECO will execute the search command as though it had been entered with the same character string argument as the last search command entered. For example, suppose the command "STHE END" results in an error message, indicating that character string "THE END" was not found on the current page. Entering the command "N" causes TECO to execute an N search for the same character string. Although the text argument may be omitted, the command terminator <DELIM> must always be entered.

Search commands can make use of the colon modifier described in Chapter 3. The following examples illustrate use of the colon modifier:

**Commands:**

```
n:Stext`
m,n:Stext`
n:Ntext`
n:_text`
n:FStext1`text2`
m,n:FStext1`text2`
n:FNtext1`text2`
```



## Command Descriptions

### 5.7 Search Commands

etc.

#### Function:

In each case, execute the search command. If the search is successful, execute the next sequential command with an argument of -1. If the search fails, execute the next command with an argument of zero. If the next command does not require a numeric argument, execute it as it stands.

All search commands may also use the @ modifier to use alternate delimiters, to allow <ESCAPE> characters in search strings or to avoid the use of <ESCAPE> characters in command strings. Such search commands take the following forms:

#### Commands:

@S/text/  
m,n@FB/text/  
n@FS/text1/text2/  
@:N/text/  
etc.

## 5.8 Search Arguments

TECO builds the search string by loading its search string buffer from the supplied search command argument. To help you enter special characters or frequently used character sequences, the argument may contain special string building characters. Table 5–17 lists the string building characters and their functions.

TECO-8 does not support the extended String Build functions or Match Control Constructs that begin with <CTRL/E>. This is a general property and will not be repeated in the following tables.

Note that, as explained in Chapter 3, a caret (up-arrow) may be used to indicate that the character following it is to be treated as a control character. Any of the commands below may be entered using the caret. This function of the caret can be disabled by using the ED flag (see Section 5.16 and appendices).

**Table 5–17 String Building Characters**

Character	Function
<CTRL/Q>	A <CTRL/Q> character in a search command argument indicates that the character following the <CTRL/Q> is to be used literally rather than as a match control character.
<CTRL/R>	Same as <CTRL/Q>.
<CTRL/V>	A <CTRL/V> character in a search command argument indicates that the character following the <CTRL/V> is to be used as the equivalent character in the lower case ASCII range (i.e., octal 100 to 137 is treated as octal 140 to 177).

(continued on next page)



Table 5–17 (Cont.) String Building Characters

Character	Function
<CTRL/V><CTRL/V>	Two successive <CTRL/V> characters in a string argument indicate to TECO that all following alphabetic characters in this string are to be converted to lower case unless an explicit ^W is given to override this state. This state continues until the end of the string or until a ^W^W construct is encountered. [TECO-10 only]
<CTRL/W>	A <CTRL/W> character in a search command argument indicates that the character following the <CTRL/W> is to be used as the equivalent character in the upper case ASCII range (i.e., octal 140 to 177 is treated as octal 100 to 137).
<CTRL/W><CTRL/W>	Two successive <CTRL/W> characters indicates to TECO that all following alphabetic characters in this string are to be converted to upper case unless an explicit ^V is encountered to override this state. This state continues until the end of the string or until a ^V^V construct is encountered. [TECO-10 only]
<CTRL/E>Qq	<CTRL/E>Qq indicates that the string stored in Q-register q is to be used in the position occupied by the ^EQq in the search string. Q registers are discussed in sections 3.3.2 and 5.9. [Not in TECO-10]
<CTRL/E>Uq	<CTRL/E>Uq indicates that the character whose ASCII code is specified by the numeric storage area of Q-register q is to be used in the position occupied by the ^EUq in the search string. [TECO-11 only]

String build characters are also permitted inside the string arguments of the O, EB, ER, EW, and EG commands.

TECO executes a search command by attempting to match the search command argument character-by-character with some portion of the input file. There are several special control characters that may be used in search command arguments to alter the usual matching process. Table 5–18 lists these match control characters and their functions.

Table 5–18 Match Control Characters

Character	Function
<CTRL/X>	A <CTRL/X> character indicates that this position in the character string may be any character. TECO accepts any character as a match for <CTRL/X>.
<CTRL/S>	A <CTRL/S> character indicates that any separator character is acceptable in this position. TECO accepts any character that is not a letter (upper or lower case A to Z) or a digit (0 to 9) as a match for <CTRL/S>.
<CTRL/N>x	TECO accepts any character as a match for the <CTRL/N>x combination EXCEPT the character which follows the <CTRL/N>. <CTRL/N> can be combined with other special characters. For example, the combination <CTRL/N><CTRL/E>D means match anything except a digit in this position.

(continued on next page)

## Command Descriptions

### 5.8 Search Arguments

**Table 5–18 (Cont.) Match Control Characters**

Character	Function
<CTRL/E>A	<CTRL/E>A indicates that any alphabetic character (upper or lower case A to Z) is acceptable in this position.
<CTRL/E>B	Same as <CTRL/S>.
<CTRL/E>C	<CTRL/E>C indicates that any character that is legal as part of a symbol constituent is acceptable in this position. TECO accepts any letter (upper or lower case A to Z), any digit (0 to 9), a dot (.), or a dollar sign (\$) as a match for <CTRL/E>C. Additional characters may be matched depending upon the operating system; consult the appropriate appendix.
<CTRL/E>D	<CTRL/E>D indicates that any digit (0 to 9) is acceptable in this position.
<CTRL/E>Gq	<CTRL/E>Gq indicates that any character contained in Q-register q is acceptable in this position. For example, if Q-register A contains "A*:" then TECO accepts either A, *, or : as a match for <CTRL/E>GA. [Not in TECO-10]
<CTRL/E>L	<CTRL/E>L indicates that any line terminator (line feed, vertical tab, or form feed) is acceptable in the position occupied by <CTRL/E>L in the search string.
<CTRL/E>M	<CTRL/E>M indicates that any non-zero number of occurrences of the immediately following character or match control construct is acceptable at this position. [TECO-10 only]
<CTRL/E>R	<CTRL/E>R indicates that any alphanumeric character (letter or digit as defined above) is acceptable in this position.
<CTRL/E>S	<CTRL/E>S indicates that any non-null string of spaces and/or tabs is acceptable in the position occupied by <CTRL/E>S.
<CTRL/E>V	<CTRL/E>V indicates that any lower case alphabetic character is acceptable in this position.
<CTRL/E>W	<CTRL/E>W indicates that any upper case alphabetic character is acceptable in this position.
<CTRL/E>X	Equivalent to <CTRL/X>.
<CTRL/E><nnn>	<CTRL/E><nnn> indicates that the character whose ASCII octal code is nnn is acceptable in this position. [TECO-10 only]
<CTRL/E>[a,b,c,...] <CTRL/E>[a,b,c,...]	indicates that any one of the specified characters is acceptable in this position. One or more characters or other match control constructs are permitted. [TECO-10 only]

TECO-8 uses special symbols to represent certain match control characters when they are displayed by an error message. These display symbols are:

Character	Display
^N	<NOT>
^S	<SEP>
^X	<ANY>

## 5.9 Q-Register Manipulation

TECO provides data storage registers, called Q-registers, each of which may be used to store an integral numeric value and, simultaneously, an ASCII character string. Q-registers are described in Section 3.3.2; this section describes the commands used to load values into and retrieve values from Q-registers.

An important kind of character string which may be stored in the text portion of a Q-register is a TECO command string. Such a command is known as a macro, and is available for execution via the Mq command, described in Table 5–21 below. (See also Section 4.4, Immediate ESCAPE-Sequence Commands [TECO-11 only], for a keypad method of macro invocation, and Section 4.2 for the \*q command which saves the last-typed command for possible editing or re-execution.)

TECO-10 allows additional Q-registers other than those described in Section 3.3.2. Consult Appendix I for further details.

Table 5–19 lists the commands which permit characters to be loaded into the Q-registers.

**Table 5–19 Q-Register Loading Commands**

Command	Function
nUq	Put n in the numeric storage area of Q-register q.
m,nUq	Equivalent to the nUqm command. That is, this command puts the number n into the numeric storage area of Q-register q and then returns the number m as a value. The command UAUB is useful at the beginning of a macro to save the two arguments specified on the macro call. (See the m,nMq command below.)
n%q	Add n to the contents of the number storage area of Q-register q. The updated contents of Q-register q are also returned as a value to be passed to the next command. If your intent is only to update the Q-register, good programming practice suggests following the n%q command with a <DELIM> or ^[ to prevent the returned value from unintentionally affecting the following command.
n%q`	Same as n%q but discards the value returned.
%q	Equivalent to 1%q.
^Uqstring`	This command inserts character string "string" into the text storage area of Q-register q. When entering a command string from the terminal, you must specify ^U using the caret/U format, since the <CTRL/U> character is the line erase immediate action command.
:^Uqstring`	This command appends character string "string" to the text storage area of Q-register "q". [not in TECO-8]
n^Uq`	This form of the ^Uq` command inserts the single character whose ASCII code is n into the text storage area of Q-register "q". (n is taken modulo 256 in TECO-11, modulo 128 in other TECOs.) [not in TECO-8]
n:^Uq`	This form of the :^Uq` command appends the single character whose ASCII code is n to the text storage area of Q-register "q". (n is taken modulo 256 in TECO-11, modulo 128 in other TECOs.) [not in TECO-8]

(continued on next page)

## Command Descriptions

### 5.9 Q-Register Manipulation

**Table 5–19 (Cont.) Q-Register Loading Commands**

Command	Function
@^Uq/string/ @:^Uq/string/ n@^Uq// n@:^Uq//	Equivalent, respectively, to the ^Uqstring`, :^Uqstring`, n^Uq`, and n:^Uq` commands, except that alternate delimiters are used and no <DELIM> characters are necessary.
nXq	Clear Q-register q and move n lines into it, where n is a signed integer. If n is positive, the n lines following the current pointer position are copied into the text storage area of Q-register q. If n is negative, the n lines preceding the pointer are copied. If n is zero, the contents of the buffer from the beginning of the line on which the pointer is located up to the pointer is copied. The pointer is not moved. The text is not deleted.
Xq	Equivalent to 1Xq.
-Xq	Equivalent to -1Xq.
m,nXq	Copy the contents of the buffer from the m+1th character through and including the nth character into the text storage area of Q-register q. M and n must be positive, and m should be less than n.
..,nXq	Copy the n characters immediately following the buffer pointer into the text storage area of Q-register q. N should be greater than zero.
..-n,Xq	Copy the n characters immediately preceding the buffer pointer into the text storage area of Q-register q. N should be greater than zero.
n:Xq	Append n lines to Q-register q, where n is a signed integer with the same functions as n in the nXq command above. The pointer is not moved. [not in TECO-8]  The colon construct for appending to a Q-register can be used with all forms of the X command.
]q	Pop from the Q-register push-down list into Q-register q. Any previous contents of Q-register q are destroyed. Both the numeric and text parts of the Q-register are loaded by this command. The Q-register push-down list is a last-in first-out (LIFO) storage area. (See section 3.3.3 for a description of the push-down list.) This command does not use or affect numeric values. Numeric values are passed through this command transparently. This allows macros to restore Q-registers and still return numeric values. [Not in TECO-8]
:]q	Execute the ]q command and return a numeric value. A -1 indicates that there was another item on the Q-register push-down list to be popped. A 0 indicates that the Q-register push-down list was empty, so Q-register q was not modified. [Not in TECO-8]
*q	Save last-typed command string. See Section 4.2.

Table 5–20 lists the commands which permit data to be retrieved from the Q-registers.

**Table 5–20 Q-Register Retrieval Commands**

Command	Function
Qq	Use the integer stored in the number storage area of Q-register q as the argument of the next command.
nQq	Return the ASCII value of the (n+1)th character in Q-register q. The argument n must be between 0 and the Q-register's size minus 1. If n is out of range, a value of -1 is returned. Characters within a Q-register are numbered the same way that characters in the text buffer are numbered. The initial character is at character position 0, the next character is at character position 1, etc. Therefore, if Q-register A contains "xyz", then 0QA will return the ASCII code for "x" and 1QA will return the ASCII code for "y".
:Qq	Use the number of characters stored in the text storage area of Q-register q as the argument of the next command.
Gq	Copy the contents of the text storage area of Q-register q into the buffer at the current position of the buffer pointer, leaving the pointer positioned after the last character copied.
:Gq	Print the contents of the text storage area of Q-register q on the terminal. Neither the text buffer nor the buffer pointer is changed by this command.
[q	Copy the contents of the numeric and text storage areas of Q-register q into the Q-register push-down list. This command does not alter either the numeric or text storage areas of Q-register q. It does not use or affect numeric values. Numeric values are passed through this command transparently, allowing macros to save temporary Q-registers and still accept numeric values. (Note, however, macros written to use local Q-registers, available in TECO-11, may be able to avoid saving and restoring Q-registers via the the pushdown list.) The command sequence [A ]B replicates the text string and numeric value from Q-register A into Q-register B. [Not in TECO-8]

Table 5–21 lists the commands which cause macros (strings stored in Q-registers) to be executed.

Macro invocations can be nested recursively; the limit is set by the amount of pushdown storage TECO has available.

In this table only, a distinction is made between a global Q-register name (indicated below by "q") and a local Q-register name (indicated below by ".q"). Elsewhere in this manual, "q" indicates either a global or local Q-register name.

**Table 5–21 Macro Invocation Commands**

Command	Function
Mq	Execute the contents of the text storage area of global Q-register q as a command string. In TECO-11, a new set of local Q-registers is created before the macro is invoked.
nMq	Execute the Mq command as above, using n as a numeric argument for the first command contained in global Q-register q. In TECO-11, a new set of local Q-registers is created.

(continued on next page)

## Command Descriptions

### 5.9 Q-Register Manipulation

**Table 5–21 (Cont.) Macro Invocation Commands**

Command	Function
m,nMq	Execute the Mq command as above, using m,n as numeric arguments for the first command contained in global Q-register q. In TECO-11, a new set of local Q-registers is created.
:Mq	Execute the contents of the text storage area of global Q-register q as a command string. In TECO-11, the current set of local Q-registers remains available to the invoked macro; no new set is created.
n:Mq	Execute the :Mq command as above, using n as a numeric argument for the first command contained in global Q-register q. In TECO-11, no new set of local Q-registers is created.
m,n:Mq	Execute the :Mq command as above, using m,n as numeric arguments for the first command contained in global Q-register q. In TECO-11, no new set of local Q-registers is created.
M.q	Execute the contents of the text storage area of local Q-register .q as a command string. In TECO-11, no new set of local Q-registers is created.
nM.q	Execute the M.q command as above, using n as a numeric argument for the first command contained in local Q-register .q. In TECO-11, no new set of local Q-registers is created.
m,nM.q	Execute the M.q command as above, using m,n as numeric arguments for the first command contained in local Q-register .q. In TECO-11, no new set of local Q-registers is created.
:M.q	Execute the contents of the text storage area of local Q-register .q as a command string. In TECO-11, no new set of local Q-registers is created.
n:M.q	Execute the :M.q command as above, using n as a numeric argument for the first command contained in local Q-register .q. In TECO-11, no new set of local Q-registers is created.
m,n:M.q	Execute the :M.q command as above, using m,n as numeric arguments for the first command contained in local Q-register .q. In TECO-11, no new set of local Q-registers is created.

## 5.10 Arithmetic and Expressions

The numeric argument of a TECO command may consist of a single integer, any of the characters listed in Table 5–24, the numeric contents of any Q-register, or an arithmetic combination of these elements. If an arithmetic expression is supplied as a numeric argument, TECO will evaluate the expression. All arithmetic expressions are evaluated from left to right without any operator precedence. Parentheses may be used to override the normal order of evaluation of an expression. If parentheses are used, all operations within the parentheses are performed, left to right, before operations outside the parentheses. Parentheses may be nested, in which case the innermost expression contained by parentheses will be evaluated first. Table 5–22 lists all of the arithmetic operators that may be used in arithmetic expressions.

**Table 5–22 Arithmetic Operators**

Operator	Example	Function
+	+2=2	Ignored if used before the first term in an expression.
+	5+6=11	Addition, if used between terms.
-	-2=-2	Negation, if used before the first term in an expression.
-	8-2=6	Subtraction, if used between terms
*	8*2=16	Multiplication. Used between terms.
/	8/3=2	Integer division with loss of the remainder. Used between terms.
&	12&10=8	Bitwise logical AND of the binary representation of the two terms. Used between the terms.
#	12#10=14	Bitwise logical OR of the binary of the two terms. Used between the terms.
^_	5^_=-6	Unary one's complement. Used after an expression. This is a TECO command that complements its argument. Strictly speaking, it is not a unary operator.

**Table 5–23 Conversion and Radix Control Commands**

Command	Function
n=	This command causes the value of n to be output at the terminal in decimal followed by a carriage return and line feed. Decimal numeric conversion is signed. For example, the unsigned number 65535 will output as -1 on TECO-11. TECO's radix is unaltered.
n= =	This command causes the value of n to be output at the terminal in octal (base 8) followed by a carriage return and line feed. Octal numeric conversion is unsigned. For example, the unsigned number 8191 (decimal) will output as 17777 on TECO-8. TECO's radix is unaltered.
n= ==	This command causes the value of n to be output at the terminal in hexadecimal (base 16) followed by a carriage return and line feed. Hexadecimal output is unsigned. TECO's radix is unaltered. [TECO-11 only]
n:= n:= = n:= ==	These commands are equivalent to n=, n= =, and n= ==, except that they leave the carriage positioned at the end of the output.
^O	<CTRL/O> (caret/O) causes all subsequent numeric input to be accepted as octal numbers. Numeric conversions using the \ or n\ commands will also be octal. The digits 8 and 9 become illegal as numeric characters. The octal radix will continue to be used until the next ^D command is executed or until TECO's radix is changed by an n^R command. NOTE: On TECO-10, this command only affects the immediately following digit string.
^D	<CTRL/D> (caret/D) causes all subsequent numeric input to be accepted as decimal numbers. This is the initial setting. [Not in TECO-10]

(continued on next page)



## Command Descriptions

### 5.10 Arithmetic and Expressions

Table 5–23 (Cont.) Conversion and Radix Control Commands

Command	Function
<code>^R</code>	This command returns the binary value of TECO's current radix. [TECO-11 only]
<code>n^R</code>	This command sets TECO's radix to the value of n. It is currently implemented only in TECO-11, where n may only be one of the values 8, 10, or 16 (representing octal mode, decimal mode, or hexadecimal mode). If n is not one of these values, TECO's radix remains unchanged and the ?IRA error message is produced.
<code>\</code>	A backslash character which is not preceded by a numeric argument causes TECO to evaluate the digit string (if any) beginning with the character immediately following the buffer pointer and ending at the next character that is not valid for the current radix. The value becomes the numeric argument of the next command. The first character may be a digit or + or -. As the backslash command is evaluated, TECO moves the buffer pointer to a position immediately following the digit string. If there is no digit string following the pointer, the result is zero and the pointer position remains unchanged. Except on TECO-8, the digits 8 and 9 will stop the evaluation if TECO's current radix is octal.
<code>n\</code>	The backslash command preceded by an argument inserts the value of n into the text buffer at the current position of the pointer, leaving the pointer positioned after the last digit of the insertion. The insertion is either signed decimal (decimal radix), unsigned octal (octal radix), or unsigned hexadecimal (hexadecimal radix). Note that <code>\</code> is a "bidirectional" command. <code>n\</code> inserts a string into text while <code>\</code> (no argument) returns a numeric result.

### 5.11 Special Numeric Values

TECO maintains several internal variables which record conditions within TECO. The variable name is equivalent to the current contents of the variable and may be entered as a numeric argument to TECO commands. When the command is executed, the current value of the designated variable becomes the numeric argument of the command.

Some of the characters which stand for specific values associated with the text buffer have been introduced earlier in this manual. For example, the dot character (`.`), which represents the current pointer position, may be used in the argument of a T command. The command `"...+5T"` causes the 5 characters following the buffer pointer to be typed out. When this command is executed, the number of characters preceding the buffer pointer is substituted in each case for the "dot". The addition is then carried out, and the command is executed as though it were of the form `"m,nT"`.

Table 5–24 lists all of the characters which have special numeric values. Any of these characters may be used as numeric argument in place of the value it represents.



## Command Descriptions

### 5.11 Special Numeric Values

**Table 5–24 Characters Associated with Numeric Quantities**

Character	Function
B	Always equivalent to zero. Thus, B represents the position preceding the first character in the buffer.
Z	Equivalent to the number of characters currently contained in the buffer. Thus, Z represents the position following the last character in the buffer.
.	Equivalent to the number of characters between the beginning of the buffer and the current position of the pointer. Thus "." represents the current position of the pointer.
H	Equivalent to the numeric pair "B,Z", or "from the beginning of the buffer up to the end of the buffer." Thus, H represents the whole buffer.
nA	Equivalent to the ASCII code for the .+n+1th character in the buffer (that is, the character to the right of buffer pointer position .+n). The expression -1A is equivalent to the ASCII code of the character immediately preceding the pointer and 0A is equivalent to the ASCII code of the character immediately following the pointer (the current character). If the character position referenced lies outside the bounds of the text buffer, this command returns a -1.
Mq	The Mq command (execute the contents of the text storage area of Q-register "q" as a command string) may return a numeric value if the last command in the string returns a numeric value and is not followed by a <ESC>.
Qq	Equivalent to the value stored in the number storage area of Q-register q.
:Qq	Equivalent to the number of characters in the text storage area of Q-register q. [Not in TECO-8]
\	Backslash is equivalent to the numeric value of the digit string in the text buffer at the current pointer position, interpreted in the current radix. The pointer is moved to the end of the digit string.
^B	<CTRL/B> (caret/B) is equivalent to the current date via the following equations:
system	encoding
OS/8	$((\text{month} \times 32) + \text{day}) \times 8 + ((\text{year} - 1970) \& 7) + k$ where $k = 4096$ if $\text{year} > 1977$ and $k = 0$ otherwise
RT-11	$((\text{month} \times 32) + \text{day}) \times 32 + \text{year} - 1972$
RSTS/E	$((\text{year} - 1970) \times 1000) + \text{day within year}$
RSX-11	$((\text{year} - 1900) \times 16 + \text{month}) \times 32 + \text{day}$
VAX/VMS	$((\text{year} - 1900) \times 16 + \text{month}) \times 32 + \text{day}$
TOPS-10	$((\text{year} - 1964) \times 12 + \text{month} - 1) \times 31 + \text{day} - 1$

(continued on next page)

## Command Descriptions

### 5.11 Special Numeric Values

**Table 5–24 (Cont.) Characters Associated with Numeric Quantities**

Character	Function														
<b>^E</b>	<CTRL/E> (caret/E) determines whether or not the P command (and related operations) append a form feed to the contents of the buffer on output. If ^E is -1, a form feed is appended; if ^E is 0, no form feed is appended. Each time text is read into the text buffer TECO sets the ^E flag. If the text read terminated due to a form feed (i.e., if the text buffer was loaded with a "complete page"), ^E is set to -1. If the text read terminated because the buffer was filled to capacity before a form feed was encountered or because there was no form feed (i.e., the text buffer was not loaded with a "complete page"), ^E is set to 0. In TECO-11, you can set the ^E flag's value directly, overriding the value set by the most recent buffer read. Be sure to use only values of 0 or -1; other values may cause unpredictable results.														
<b>^F</b>	<CTRL/F> (caret/F) is equivalent to the current value of the console switch register.														
<b>n^F</b>	n<CTRL/F> is the terminal number plus 200000 (octal) for job n's terminal. -1^F is the terminal number plus 200000 (octal) for your job's terminal. The result is 0 if the specified job is detached or if there is no such job. [TECO-10 only]														
<b>^H</b>	<CTRL/H> (caret/H) is equivalent to the current time of day via the following equations: <table> <tr> <th>system</th><th>encoding</th></tr> <tr> <td>OS/8:</td><td>0</td></tr> <tr> <td>RT-11</td><td>(seconds since midnight)/2</td></tr> <tr> <td>RSTS/E</td><td>minutes until midnight</td></tr> <tr> <td>RSX-11</td><td>(seconds since midnight)/2</td></tr> <tr> <td>VAX/VMS</td><td>(seconds since midnight)/2</td></tr> <tr> <td>TOPS-10</td><td>60ths of a second since midnight (or 50ths of a second where 50 Hz power is used)</td></tr> </table>	system	encoding	OS/8:	0	RT-11	(seconds since midnight)/2	RSTS/E	minutes until midnight	RSX-11	(seconds since midnight)/2	VAX/VMS	(seconds since midnight)/2	TOPS-10	60ths of a second since midnight (or 50ths of a second where 50 Hz power is used)
system	encoding														
OS/8:	0														
RT-11	(seconds since midnight)/2														
RSTS/E	minutes until midnight														
RSX-11	(seconds since midnight)/2														
VAX/VMS	(seconds since midnight)/2														
TOPS-10	60ths of a second since midnight (or 50ths of a second where 50 Hz power is used)														
<b>^N</b>	<CTRL/N> (caret/N) is the end of file flag. It is equivalent to -1 if the file open on the currently selected input stream is at end of file, and zero otherwise.														
<b>^S</b>	<CTRL/S> (caret/S) is equivalent to the negative of the length of the last insert, string found, or string inserted with a "G" command, whichever occurred last. To back up the pointer to the start of the last insert, string found, etc., type "^SC". [Not in TECO-8]														
<b>^T</b>	<CTRL/T> (caret/T) is equivalent to the ASCII code for the next character typed at the terminal. Every ^T command executed causes TECO to pause and accept one character typed at the terminal. See the ET flag description (Section 5.16) for variations.														

(continued on next page)

**Table 5–24 (Cont.) Characters Associated with Numeric Quantities**

Character	Function														
:^T	<p>Reads and decodes the next keystroke typed at the terminal. The decoded keystroke is returned in a manner similar to that used for immediate ESCape sequence commands (see Section 4.4), i.e., via &lt;type&gt; and &lt;value&gt; codes, as follows:</p> <table> <tr> <th>&lt;type&gt;</th><th>&lt;value&gt;</th></tr> <tr> <td>0</td><td>Normal character, &lt;value&gt; is ASCII value</td></tr> <tr> <td>1</td><td>Control character (except ESCape or CSI); &lt;value&gt; is ASCII value of control char. A typed &lt;CR&gt; yields &lt;type&gt; 1 &lt;value&gt; 13, and subsequent ^T or :^T does not return a &lt;LF&gt;.</td></tr> <tr> <td>2</td><td>Same as immed ESC seq &lt;type&gt; 0</td></tr> <tr> <td>3</td><td>Same as immed ESC seq &lt;type&gt; 1</td></tr> <tr> <td>4</td><td>Same as immed ESC seq &lt;type&gt; 2</td></tr> <tr> <td>5</td><td>Same as immed ESC seq &lt;type&gt; 3</td></tr> </table>	<type>	<value>	0	Normal character, <value> is ASCII value	1	Control character (except ESCape or CSI); <value> is ASCII value of control char. A typed <CR> yields <type> 1 <value> 13, and subsequent ^T or :^T does not return a <LF>.	2	Same as immed ESC seq <type> 0	3	Same as immed ESC seq <type> 1	4	Same as immed ESC seq <type> 2	5	Same as immed ESC seq <type> 3
<type>	<value>														
0	Normal character, <value> is ASCII value														
1	Control character (except ESCape or CSI); <value> is ASCII value of control char. A typed <CR> yields <type> 1 <value> 13, and subsequent ^T or :^T does not return a <LF>.														
2	Same as immed ESC seq <type> 0														
3	Same as immed ESC seq <type> 1														
4	Same as immed ESC seq <type> 2														
5	Same as immed ESC seq <type> 3														
	[TECO-11 only]														
^Y	<p>&lt;CTRL/Y&gt; (caret/Y) is equivalent to ".+^S,.", the n,m numeric argument spanning the text just searched for or inserted. This value may be used to recover from inserting a string in the wrong place. Type "^YXSFR`" to store the string in Q-register S and remove it from the buffer. You can then position the pointer to the right place and type "GS" to insert the string. [TECO-11 only]</p>														
^Z	<p>&lt;CTRL/Z&gt; (caret/Z) is equivalent to the total space occupied by text in the Q-registers (including the command line currently being executed). [TECO-11 only]</p>														
^^x	<p>The combination of the Control-caret (double caret or double up-arrow) followed by any character is equivalent to the value of the ASCII code for that character. The "x" in this example may be any character that can be typed in to TECO.</p>														

### Mode Control Flags

The following mode control flags return numeric values. The use of these flags is described below in Section 5.16.

Flag	Meaning
ED	Equivalent to the current value of the edit level flag.
EH	Equivalent to the current value of the help level flag.
EO	Equivalent to the version number of the version of TECO which is currently being run. This manual describes TECO-11 Version 40, TECO-8 Version 7, and TECO-10 Version 3.
ES	Equivalent to the current value of the search verification flag. [Not in TECO-8]
ET	Equivalent to the current value of the type out control flag.

## Command Descriptions

### 5.11 Special Numeric Values

Flag	Meaning
EU	Equivalent to the current value of the upper/lower case flag.
EV	Equivalent to the current value of the edit verify flag. [TECO-11 only]
^X	<CTRL/X> (caret/X) is equivalent to the current value of the search mode flag. [Not in TECO-8]

### 5.12 Command Loops

You can cause a command string to be executed any number of times by placing the command string within angle brackets and preceding the brackets with a numeric argument designating the number of iterations. Iterated command strings are called command loops. Loops may be nested so that one command loop contains another command loop, which, in turn, contains other command loops, and so on. The maximum depth to which command loops may be nested is determined by the size of TECO's push-down list (system dependent), but is always greater than 10.

The general form of the command loop is:

n<command string>

where "command string" is the sequence of commands to be iterated and n is the number of iterations. If n is not supplied then no limit is placed on the number of iterations. If n is 0 or less than 0 then the iteration is not executed at all; command control skips to the closing angle bracket. If n is greater than 0, then the iteration is performed n times.

Search commands inside command loops are treated specially. If a search command which is not preceded by a colon modifier is executed within a command loop and the search fails, a warning message is printed [on TECO-11], the command loop is exited immediately and the command following the right angle bracket of the loop is the next command to be executed. If an unmodified search command in a command loop is immediately followed by a semicolon, it is treated as if it were a colon-modified search (see Section 5.13).

### 5.13 Branching Commands

TECO provides an unconditional branch command and a set of conditional execution commands. To branch within a command string, you must be able to name locations inside the string. TECO permits location tags of the form:

!tag!

to be placed between any two commands in a command string. The name "tag" will be associated with this location when the command string is executed. Tags may contain any number of ASCII characters and any character except an exclamation mark. (When using the @ form of this command, any character except the delimiter is legal.) Since tags are ignored by TECO except when a branch command references the tagged location, they may also be used as comments within complicated command strings.

The unconditional branch command is the O command which has the form:

Otag`

where "tag" is a location named elsewhere in the command string and "" signifies a <DELIM>. When an O command is executed, the next command to be executed will be the one that follows the tag referenced by the O command. Command execution continues normally from this point.

Use of the O command is subject to two restrictions. First, if an O command is stored in a Q-register as part of a command string which is to be executed by an M command, the tag referenced by the O command must reside in the same Q-register.

Second, an O command which is inside a command loop may not branch to a tagged location preceding the command loop. However, it is always possible to branch out of a command loop to a location which follows the command loop and then branch to the desired tag.

The string argument in the O command has the same format as the string arguments in the search and E commands. String build characters such as ^EQq can be embedded within the string in TECO-11. Also, in TECO-11 and TECO-10, the O command may be @-sign modified. In that case, the syntax of the command would be @O/tag/ where / represents any delimiting character that does not appear within the tag.

Branching into a conditional poses no problems, but branching into a command loop will causes unpredictable results.

Although tags may contain any sequence of ASCII characters, good programming practice dictates that tags should not contain unusual characters (such as space, comma, ESCAPE, etc.) and that they should be mnemonic for the piece of code to which they refer.

There are many other branching commands. Most of these are considerably faster than the O command and should be used wherever convenient. They are all described in the table below.

**Table 5–25 Branching Commands**

Command	Function
Otag`	This command causes TECO to branch to the first occurrence of the specified label (tag) in the current macro level. In TECO-8 and TECO-11, branching to the left of the start of the current iteration is not permitted, and this command will only look for an occurrence of the specified tag following the < of the current iteration, if you are in an iteration. In any case, branching out of an iteration is poor programming practice. Command execution resumes at the first character after the delimiter terminating the specified tag. Using this syntax, any character except <ESC> is permitted in the tag specification. The usual string build characters are permitted when specifying the tag.
@O/tag/	Equivalent to Otag` except that a delimiter (shown here as /) is used before and after the specified tag. Any character other than that delimiter is permitted inside the tag. The usual string build characters are permitted when specifying the tag. [Not in TECO-8]

(continued on next page)

## Command Descriptions

### 5.13 Branching Commands

**Table 5–25 (Cont.) Branching Commands**

Command	Function
<code>nOtag0,tag1, tag2,...`</code>	This command causes TECO to branch to the tag specified by the nth tag in the accompanying list. The string argument to this command consists of a sequence of tags separated by commas. The tags may contain any characters other than comma or <ESC>; however, good programming practice suggests that the tags should consist only of letters and digits. There must be no intervening spaces since these would be considered part of the tag. If n is out of range, or if n selects a null tag, then command execution continues with the first command following the <DELIM> that delimits this command. (A null tag would be signified in the list by two adjacent commas.) [TECO-11 only]
<code>n@O/tag0,tag1, tag2,.../</code>	Same as the preceding command except that the list of tags is bracketed by a delimiter shown here as "/". The delimiter can be any character that does not appear within the list of tags. In particular, using comma for the delimiter would not be very useful. [TECO-11 only]
<code>;</code>	This command causes TECO to branch out of the current iteration, if the immediately preceding search (or search and replace) command failed. In that case, control resumes at the character following the matching > at the end of the current iteration. On the other hand, if the preceding search succeeded, command execution continues with the character following the ;. If this command is encountered from outside of an iteration (in the current macro level), then the ?SNI error message is issued.
<code>n;</code>	This command causes TECO to branch out of the current iteration if the value of n is greater than or equal to 0. In that case, command execution resumes at the character following the matching > at the end of the current iteration. On the other hand, if n is less than 0, command execution continues with the character following the ;. If this command is encountered from outside of an iteration (in the current macro level), then the ?SNI error message is issued.
<code>::</code>	This command causes TECO to branch out of the current iteration if the immediately preceding search (or search and replace) command succeeded. In that case, control resumes at the character following the matching > at the end of the current iteration. On the other hand, if the preceding search failed, command execution continues with the character following the ;. If this command is encountered from outside of an iteration (in the current macro level), then the ?SNI error message is issued. [TECO-11 only]
<code>n::</code>	This command causes TECO to branch out of the current iteration if the value of n is less than 0. In that case, command execution resumes at the character following the matching > at the end of the current iteration. On the other hand, if n is greater than or equal to 0, command execution continues with the character following the ;. If this command is encountered from outside of an iteration (in the current macro level), then the ?SNI error message is issued. [TECO-11 only]

(continued on next page)

**Table 5–25 (Cont.) Branching Commands**

Command	Function
'	This "command" is actually part of the syntax of TECO conditionals. It has no affect if "executed" other than to signify termination of the current conditional level. If an argument is specified to this command, the result is not defined. (Arguments pass through this command on TECO-11.) Conditionals are described in Section 5.14.
	This "command" is actually part of the syntax of TECO conditionals. If executed, it causes control to branch to the end of the conditional. Command execution resumes with the character following the ' that ends the current conditional with the ELSE clause being skipped.
>	This "command" is actually part of the syntax of TECO iterations. If executed, it causes TECO to bump the current iteration count by 1 and test to see if the resulting count is equal to the maximum count permitted for the iteration (specified as an argument before the matching <). If the iteration count has not expired, then control returns to the command following the < at the beginning of the current iteration. If the iteration has expired, then command execution continues with the character following this >. If this command is encountered outside of an iteration (within the current macro level), then the ?BNI error message (or its equivalent) is issued.
F>	This command causes TECO to branch (flow) to the end of the current iteration. TECO effectively resumes execution at the matching >. The iteration count is tested as usual. If it has not expired, control returns back to the start of the iteration with the count having been incremented by 1. If the count was up, the iteration is exited and control continues with the first command after the >. If this command is encountered outside of an iteration, it has the same effect as the <DELIM><DELIM> command. [TECO-11 only]
F<	This command causes TECO to branch (flow) to the start of the current iteration. TECO effectively resumes execution at the first command following the < at the beginning of the current iteration. The iteration count is not affected. If this command is issued outside of an iteration, it causes TECO to branch back to the start of the current command string (in the current macro level). [TECO-11 only]
F'	This command causes TECO to branch (flow) to the end of the current conditional. TECO effectively resumes execution at the first command following the ' at the end of the current conditional. Numeric arguments are eaten up by this command. If this command is issued while not in a conditional, the ?MAP error (or its equivalent) is issued. [TECO-11 only]

(continued on next page)



## Command Descriptions

### 5.13 Branching Commands

Table 5–25 (Cont.) Branching Commands

Command	Function
F	This command causes TECO to branch (flow) to the else clause of the current conditional. TECO effectively resumes execution at the first command following the   at the end of the current THEN clause. If the current conditional has no ELSE clause, or if an unmatched ' is encountered before an unmatched  , then control resumes at the command following the '. Numeric arguments are eaten up by this command. If this command is issued while not in a conditional, the ?MAP error (or its equivalent) is issued. Well-structured programs should not need to use this command. [TECO-11 only]
\$\$	The <ESC><ESC> command causes TECO to exit from the current macro level. If this command is issued from top level (not from within a macro), then the command string execution is terminated and TECO returns to prompt level. Note that the second <ESC> must be a true ESCape and may not be a ^[. Also, note that both ESCapes must be true TECO commands and not part of the syntax of some previous command. That is, the first <ESC> does not count if it is the delimiting ESCape of a string. Finally, note that these must be real ESCape characters, and not the user-designated <DELIM> character (which TECO understands as an ESCape only when typed at the terminal).
n\$\$	This command causes TECO to exit from the current macro level, returning the number n as a value. This value will be used as the numeric argument to the first command following the macro call.
m,n\$\$	This command causes TECO to exit from the current macro level, returning the pair of values m and n as arguments to the first command following the macro call. Good programming practice dictates that all ways of exiting a macro return the same number of arguments.
^C	The <CTRL/C> (Caret-C) command when executed as a TECO command, causes command execution to stop and control return to TECO's prompt. No clean-up of push-down lists, flag settings, etc. is done. This command lets a macro abort TECO's command execution. [On TECO-8 and TECO-10, this command causes control to return to the operating system.] [On TECO-11, this command returns to the operating system if executed from the top level.] Consult the appendices for specific details concerning your operating system.
^C<CTRL/C>	This command causes TECO to unconditionally abort and control exits from TECO. Control returns to the operating system. The second <CTRL/C> must be a true <CTRL/C> and may not be a Caret-C.

### 5.14 Conditional Execution Commands

All conditonal execution commands are of the form:

n"X command-string '

or

n"X then-command-string | else-command-string '



## Command Descriptions

### 5.14 Conditional Execution Commands

In the first form of the command, "n" is a numeric argument on which the decision is based, "X" is any of the conditional execution commands listed in table 5-14, and "command string" is the command string which will be executed if the condition is satisfied. The numeric argument is separated from the conditional execution command by a double quote (") and the command string is terminated with an apostrophe ('). If the condition is not satisfied, the command string will not be executed; execution will continue with the first command after the apostrophe.

In the second form of the command, two command strings are specified. The first one is executed if the condition is satisfied and the second is executed if the condition is not satisfied. Only one of the command strings will be executed. After execution of the appropriate command string, control will continue with the first command after the apostrophe (unless the command string caused a branch out of the conditional to occur), since execution of the vertical bar command (|) causes TECO to scan to the next matching apostrophe.

Conditional commands are similar to the IF-THEN-ELSE constructs that you find in other structured programming languages, although none can match the brevity and elegance of TECO's implementation. Nonetheless, you must use these facilities wisely. Good programming practice dictates that a branch into the range of a conditional (from outside that range) should not occur.

Conditional execution commands may be nested in the same manner as iteration commands. That is, the command string which is to be executed if the condition on n is met may contain conditional execution commands, which may, in turn, contain further conditional execution commands.

**Table 5–26 Conditional Execution Commands**

Command	Function
n"A	Execute the following command string if n equals the ASCII code for an alphabetic character (upper or lower case A to Z).
n"C	Execute the following command string if n is the ASCII code of any character that is a symbol constituent. This is usually one of the upper or lower case letters A to Z, one of the digits 0 to 9, or period, or dollar sign, but may include additional characters on some operating systems. Consult the appropriate appendix.
n"D	Execute the following command string if n equals the ASCII code for a digit (0 to 9).
n"E	Execute the following command string if n is equal to zero.
n"F	Execute the following command string if n is FALSE. Equivalent to n"E.
n"G	Execute the following command string if n is greater than zero.
n"L	Execute the following command string if n is less than zero.
n"N	Execute the following command string if n is not equal to zero.
n"R	Execute the following command string if n equals the ASCII code for an alphanumeric (upper or lower case A to Z or 0 to 9).
n"S	Execute the following command string if n is SUCCESSFUL. Equivalent to n"L.

(continued on next page)

## Command Descriptions

### 5.14 Conditional Execution Commands

**Table 5–26 (Cont.) Conditional Execution Commands**

Command	Function
n"T	Execute the following command string if n is TRUE. Equivalent to n"L.
n"U	Execute the following command string if n is UNSUCCESSFUL. Equivalent to n"E.
n"V	Execute the following command string if n equals the ASCII code for a lower case alphabetic character (lower case A to Z). [Not in TECO-8]
n"W	Execute the following command string if n equals the ASCII code for an upper case alphabetic character (upper case A to Z). [Not in TECO-8]
n"<	Identical to n"L
n">	Identical to n"G
n"=	Identical to n"E

### 5.15 Retrieving Environment Characteristics

The following TECO commands return values of interest to users who want information about their current job, the operating system, their terminal, etc.

All negative EJ commands return an operating system dependent value. Consult the appendices for operating system unique commands.

**Table 5–27 Retrieving Environment Characteristics**

Command	Function
---------	----------

(continued on next page)

## Command Descriptions

### 5.15 Retrieving Environment Characteristics

**Table 5–27 (Cont.) Retrieving Environment Characteristics**

Command	Function																														
-1EJ	<p>Return a number representing the computer and operating system upon which TECO is currently running. This value has the form 256m+n where m is a number representing the computer in use and n is a number representing the operating system that is running. Current values of m and n are:</p> <table> <tr> <th>Computer (m)</th><th>Operating System (n)</th></tr> <tr> <td>0</td><td>PDP-11</td></tr> <tr> <td></td><td>0 RSX-11D</td></tr> <tr> <td></td><td>1 RSX-11M</td></tr> <tr> <td></td><td>2 RSX-11S</td></tr> <tr> <td></td><td>3 IAS</td></tr> <tr> <td></td><td>4 RSTS/E</td></tr> <tr> <td></td><td>5 VAX/VMS (compatibility mode)</td></tr> <tr> <td></td><td>6 RSX-11M+</td></tr> <tr> <td></td><td>7 RT-11</td></tr> <tr> <td>1</td><td>PDP-8</td></tr> <tr> <td>2</td><td>DEC-10</td></tr> <tr> <td>3</td><td>DEC-20</td></tr> <tr> <td>4</td><td>VAX-11</td></tr> <tr> <td></td><td>0 VAX/VMS (native mode)</td></tr> </table>	Computer (m)	Operating System (n)	0	PDP-11		0 RSX-11D		1 RSX-11M		2 RSX-11S		3 IAS		4 RSTS/E		5 VAX/VMS (compatibility mode)		6 RSX-11M+		7 RT-11	1	PDP-8	2	DEC-10	3	DEC-20	4	VAX-11		0 VAX/VMS (native mode)
Computer (m)	Operating System (n)																														
0	PDP-11																														
	0 RSX-11D																														
	1 RSX-11M																														
	2 RSX-11S																														
	3 IAS																														
	4 RSTS/E																														
	5 VAX/VMS (compatibility mode)																														
	6 RSX-11M+																														
	7 RT-11																														
1	PDP-8																														
2	DEC-10																														
3	DEC-20																														
4	VAX-11																														
	0 VAX/VMS (native mode)																														
0EJ	Returns a value equal to your job number. On single-user systems, this is always a 0.																														
1EJ	Returns a value equal to your terminal keyboard number (the keyboard you detached from if you are running detached). On single-terminal systems, this is always a 0.																														
2EJ	Returns a value equal to your operating system's user identification number. This may be called your UIC, PPN, Group, etc. under various operating systems. Consult the appendices for more information.																														

**Table 5–28 Setting Environment Information**

Command	Function
n,1EJ	Set the terminal number to receive output. This will not affect terminal input. Your job will remain attached to, or detached from, your terminal, whichever it was before. Output will only occur if the specified terminal is ASSIGNed with a monitor ASSIGN command (you may ^C, issue that command and continue) and if your job has POKE privileges. This command also sets the terminal to be reattached if the set detach flag (64&ET) is cleared. The reattaching operation requires [1,2] or JACCT privileges. [TECO-10 only]
n,2EJ	Sets your [p,pn] to n where n has the same format as the number returned by the 2EJ command. Issuance of this command requires the appropriate privileges. [TECO-10 only]

## 5.16 Mode Control Flags

TECO has flags which control various aspects of its operation. You can find a flag's current setting by executing its command name without an argument; the current setting of the flag is returned as a value. A flag may be set to a specific value by executing its command name preceded by a numerical argument; the flag is set to the value of the argument.

The following table describes the commands that set and clear flags; <flag> represents any of the flags listed below.

**Table 5–29 Flag Manipulation Commands**

Command	Function
<flag>	Return value of flag.
n<flag>	Set value of flag to n.
m,n<flag>	In the flag, turn off those bits specified by m and turn on those bits specified by n.
0,n<flag>	Turn on the bits in the flag specified by n.
m,0<flag>	Turn off the bits in the flag specified by m.

The flags have the following functions:

**Table 5–30 Mode Control Flags**

Command	Function				
ED	The edit level flag, a bit-encoded word that controls TECO's behavior in various respects. Any combination of the individual bits may be set as the user sees fit. The bits have the following functions:				
<table><tr><th>Bit</th><th>Function</th></tr><tr><td>ED&amp;1</td><td>Allow caret (^) in search strings. If this bit is clear, a caret (^) in a search string modifies the immediately following character to become a control character. When this bit is set, a caret in a search string is simply the literal character caret. If you are editing a file that contains many caret characters (e.g., a RUNOFF file with case control), you will want to set this bit. (For control of upper/lower case matching in search strings, see the ^X flag.)</td></tr></table>		Bit	Function	ED&1	Allow caret (^) in search strings. If this bit is clear, a caret (^) in a search string modifies the immediately following character to become a control character. When this bit is set, a caret in a search string is simply the literal character caret. If you are editing a file that contains many caret characters (e.g., a RUNOFF file with case control), you will want to set this bit. (For control of upper/lower case matching in search strings, see the ^X flag.)
Bit	Function				
ED&1	Allow caret (^) in search strings. If this bit is clear, a caret (^) in a search string modifies the immediately following character to become a control character. When this bit is set, a caret in a search string is simply the literal character caret. If you are editing a file that contains many caret characters (e.g., a RUNOFF file with case control), you will want to set this bit. (For control of upper/lower case matching in search strings, see the ^X flag.)				

(continued on next page)

Table 5–30 (Cont.) Mode Control Flags

Command	Function
ED&2	Allow all Y and _ commands. If this bit is set, the Y (Yank) command and _ (underscore or backarrow) command work unconditionally as described earlier in the manual. If clear, the behavior of the Y and _ commands are modified as follows: If an output file is open and text exists in the text buffer, the Y or _ command will produce an error message and the command will be aborted leaving the text buffer unchanged. Note that if no output file is open the Y and _ commands act normally. Furthermore, if the text buffer is empty the Y command can be used to bring in a page of text whether or not an output file is open (HKY will always work). The _ command will succeed in bringing one page of text into an empty text buffer but will fail to bring in successive pages if an output file is open.
ED&4	When this bit is clear, TECO will try to expand memory as much as it can in order to try to fit entire pages into memory when requested to do so. If this bit is set, arbitrary memory expansion will not occur. In that case, TECO will expand memory only on the A command and not on the Y, P, or N commands. This bit is always set in TECO-10 and has no significance in TECO-8 or in TECO-11 on RT-11.
ED&8	Reserved for future use by TECO-8.
ED&16	Allow failing searches to preserve dot. If this bit is set, then whenever a search fails, the original location of the text buffer pointer will be preserved. If this bit is clear, then failing searches (other than bounded searches) leave the text buffer pointer at pointer position 0 after they fail. [not in TECO-8]
ED&32	Enable immediate ESCape-sequence commands. If this bit is set, TECO will recognize an ESCape-sequence key pressed immediately after the prompting asterisk as an immediate command. See Section 4.4 for a description of immediate ESCape-sequence commands.  If this bit is clear (the default case), TECO will treat an ESCape coming in immediately after the asterisk prompt as a <DELIM> That is, TECO will hear a discrete <ESC> character: an ESCape sequence will therefore be treated not as a unified command, but as a sequence of characters. [TECO-11 only]

(continued on next page)

## Command Descriptions

### 5.16 Mode Control Flags

Table 5–30 (Cont.) Mode Control Flags

Command	Function
ED&64	Only move dot by one on multiple occurrence searches. If this bit is clear, TECO treats nStext` exactly as n<1Stext`. That is, skip over the whole matched search string when proceeding to the nth search match. For example, if the text buffer contains only A's, the command 5SAA` will complete with dot equal to ten (10). If this bit is set, TECO increments dot by one each search match. In the above example, dot would become five (5). [TECO-11 only]
ED&128	Automatic refresh inhibit. If scroll mode is enabled (that is, if 7:W is non-zero), TECO normally refreshes the text buffer display just before printing its * prompt. Setting the 128 bit in ED inhibits this automatic refresh. (Inhibiting the automatic refresh may be useful when running over a slow terminal line; see also the ^W immediate command.) [TECO-11 only]
	The initial value of ED&1 is system dependent (See appendices). The initial value of the other bits in the ED flag is 0.
EE	This flag is initially zero, and (unless the 8192 ET bit is set) TECO recognizes only the ESCape as its <DELIM> character from the terminal. If it is desired to use another character as an ESCape surrogate (as when working from a terminal lacking an ESCape key), the ASCII value of that other character may be set in the EE flag. Note that when an ESCape surrogate is set, the designated character (when typed at the terminal) is received by TECO as an ESCape—that character is no longer directly available at the keyboard. When an ESCape surrogate is active, an ESCape received by TECO echos as accent grave; when none is active, an ESCape received by TECO echos as dollar sign. This feature is provided for the benefit of newer terminals which lack an ESCape key. (8192ET and 96EE both set ` as ESCape surrogate. EE, however, can be used to designate a character other than accent grave.) [RSTS/E only]

(continued on next page)

Table 5–30 (Cont.) Mode Control Flags

Command	Function						
EH	<p>The help level flag, which controls the printing of error messages and failed commands. (See also the / command.)</p> <table> <tr> <th>Bits</th><th>Function</th></tr> <tr> <td>EH&amp;3</td><td> <p>The low two bits of EH (value range 0 through 3) control the printing of TECO error messages as follows (assuming the low two bits have value m):</p> <p>If m is equal to 1, error messages are output in abbreviated form ("?XXX"). If m is equal to 2, error messages are output in normal form ("?XXX Message"). If m is equal to 3, error messages are output in long or "War and Peace" form, that is, a paragraph of informative material is typed following the normal form of the error message. In TECO-11, m=3 is implemented only on VAX/VMS. (See section G.13 for a description of HELP facilities available on VAX/VMS.) In other TECO-11's, case m=3 is equivalent to m=2.</p> </td></tr> <tr> <td>EH&amp;4</td><td> <p>If this bit of EH is set, the failing command is also output up to and including the failing character in the command followed by a question mark. (Just like TECO's response to the typing of a question mark immediately after an error.) This bit is not supported by TECO-10.</p> </td></tr> </table>	Bits	Function	EH&3	<p>The low two bits of EH (value range 0 through 3) control the printing of TECO error messages as follows (assuming the low two bits have value m):</p> <p>If m is equal to 1, error messages are output in abbreviated form ("?XXX"). If m is equal to 2, error messages are output in normal form ("?XXX Message"). If m is equal to 3, error messages are output in long or "War and Peace" form, that is, a paragraph of informative material is typed following the normal form of the error message. In TECO-11, m=3 is implemented only on VAX/VMS. (See section G.13 for a description of HELP facilities available on VAX/VMS.) In other TECO-11's, case m=3 is equivalent to m=2.</p>	EH&4	<p>If this bit of EH is set, the failing command is also output up to and including the failing character in the command followed by a question mark. (Just like TECO's response to the typing of a question mark immediately after an error.) This bit is not supported by TECO-10.</p>
Bits	Function						
EH&3	<p>The low two bits of EH (value range 0 through 3) control the printing of TECO error messages as follows (assuming the low two bits have value m):</p> <p>If m is equal to 1, error messages are output in abbreviated form ("?XXX"). If m is equal to 2, error messages are output in normal form ("?XXX Message"). If m is equal to 3, error messages are output in long or "War and Peace" form, that is, a paragraph of informative material is typed following the normal form of the error message. In TECO-11, m=3 is implemented only on VAX/VMS. (See section G.13 for a description of HELP facilities available on VAX/VMS.) In other TECO-11's, case m=3 is equivalent to m=2.</p>						
EH&4	<p>If this bit of EH is set, the failing command is also output up to and including the failing character in the command followed by a question mark. (Just like TECO's response to the typing of a question mark immediately after an error.) This bit is not supported by TECO-10.</p>						
	<p>The initial value of the EH flag is 0 which is equivalent to a value of 2.</p>						
EO	<p>Setting the value of the EO flag to n allows features that were peculiar to that version of TECO to work. [TECO-10 only]</p>						
ES	<p>The search verification flag, which controls the text typed out after searches.</p> <p>If n is equal to 0, nothing is typed out after searches. If n is -1, the current line is typed out when a successful search at top level is completed (i.e., a V command is done automatically). If n is between 1 and 31, the current line is typed out with a line feed immediately following the position of the pointer to identify its position. If n is between 32 and 126, the current line is typed out with the ASCII character corresponding to the value of n immediately following the position of the pointer to identify its position. If you want to see more than one line of type out, use the form m*256+n. The n is the same as above. The m is the number of lines of view. For example, 3*256+^^! would give two lines on either side of the found line, and the found line with the character "!" at the pointer's position. The ES flag does not apply to searches executed inside iterations or macros; lines found inside iterations or macros are never typed out. [Not in TECO-8]</p> <p>The initial value of ES is 0.</p>						

(continued on next page)

## Command Descriptions

### 5.16 Mode Control Flags

**Table 5–30 (Cont.) Mode Control Flags**

Command	Function																
ET	<p>The ET flag is a bit-encoded word controlling TECO's treatment of the terminal. Any combination of the individual bits may be set. The bits provide the following functions, when set:</p> <table> <tr> <th>Bit</th><th>Function</th></tr> <tr> <td>ET&amp;1</td><td>Type out in image mode. Setting this bit inhibits all of TECO's type out conversions. All characters are output to the terminal exactly as they appear in the buffer or ^A command. For example, the changing of control characters into the "caret/character" form, and the conversion of &lt;ESCAPE&gt; to ` (accent grave) or to \$ (dollar sign) are suppressed. This mode is useful for driving displays. It should be used with caution, especially if you are talking to TECO over a dial-up line.</td></tr> <tr> <td>ET&amp;2</td><td>Process DELETes and &lt;CTRL/U&gt;s in "scope" mode. Scope mode processing uses the cursor control features of CRT type terminals to handle character deletion by actually erasing characters from the screen.</td></tr> <tr> <td>ET&amp;4</td><td>Read lower case. TECO normally converts all lower case alphabets to upper case on input. Setting this bit causes lower case alphabets to be input as lower case. TECO commands and file specifiers may be typed in either upper or lower case. For the purpose of searches, however, upper and lower case may be treated as different characters. (See ^X flag).</td></tr> <tr> <td>ET&amp;8</td><td>Read without echo for ^T commands. This allows data to be read by the ^T command without having the characters echo at the terminal. Normal command input to TECO will echo.</td></tr> <tr> <td>ET&amp;16</td><td>Cancel &lt;CTRL/O&gt; on type out. Setting this bit will cancel any outstanding &lt;CTRL/O&gt; when the next type out occurs. After TECO has canceled the &lt;CTRL/O&gt;, it will automatically clear the bit.</td></tr> <tr> <td>ET&amp;32</td><td>Read with no wait. This enables the ^T command to test if a character is available at the user terminal. If a character has been typed, ^T returns the value of the character as always. If no character has been typed, ^T immediately returns a value of -1 and execution continues without waiting for a character.</td></tr> <tr> <td>ET&amp;64</td><td>Detach flag (See appendices).</td></tr> </table>	Bit	Function	ET&1	Type out in image mode. Setting this bit inhibits all of TECO's type out conversions. All characters are output to the terminal exactly as they appear in the buffer or ^A command. For example, the changing of control characters into the "caret/character" form, and the conversion of <ESCAPE> to ` (accent grave) or to \$ (dollar sign) are suppressed. This mode is useful for driving displays. It should be used with caution, especially if you are talking to TECO over a dial-up line.	ET&2	Process DELETes and <CTRL/U>s in "scope" mode. Scope mode processing uses the cursor control features of CRT type terminals to handle character deletion by actually erasing characters from the screen.	ET&4	Read lower case. TECO normally converts all lower case alphabets to upper case on input. Setting this bit causes lower case alphabets to be input as lower case. TECO commands and file specifiers may be typed in either upper or lower case. For the purpose of searches, however, upper and lower case may be treated as different characters. (See ^X flag).	ET&8	Read without echo for ^T commands. This allows data to be read by the ^T command without having the characters echo at the terminal. Normal command input to TECO will echo.	ET&16	Cancel <CTRL/O> on type out. Setting this bit will cancel any outstanding <CTRL/O> when the next type out occurs. After TECO has canceled the <CTRL/O>, it will automatically clear the bit.	ET&32	Read with no wait. This enables the ^T command to test if a character is available at the user terminal. If a character has been typed, ^T returns the value of the character as always. If no character has been typed, ^T immediately returns a value of -1 and execution continues without waiting for a character.	ET&64	Detach flag (See appendices).
Bit	Function																
ET&1	Type out in image mode. Setting this bit inhibits all of TECO's type out conversions. All characters are output to the terminal exactly as they appear in the buffer or ^A command. For example, the changing of control characters into the "caret/character" form, and the conversion of <ESCAPE> to ` (accent grave) or to \$ (dollar sign) are suppressed. This mode is useful for driving displays. It should be used with caution, especially if you are talking to TECO over a dial-up line.																
ET&2	Process DELETes and <CTRL/U>s in "scope" mode. Scope mode processing uses the cursor control features of CRT type terminals to handle character deletion by actually erasing characters from the screen.																
ET&4	Read lower case. TECO normally converts all lower case alphabets to upper case on input. Setting this bit causes lower case alphabets to be input as lower case. TECO commands and file specifiers may be typed in either upper or lower case. For the purpose of searches, however, upper and lower case may be treated as different characters. (See ^X flag).																
ET&8	Read without echo for ^T commands. This allows data to be read by the ^T command without having the characters echo at the terminal. Normal command input to TECO will echo.																
ET&16	Cancel <CTRL/O> on type out. Setting this bit will cancel any outstanding <CTRL/O> when the next type out occurs. After TECO has canceled the <CTRL/O>, it will automatically clear the bit.																
ET&32	Read with no wait. This enables the ^T command to test if a character is available at the user terminal. If a character has been typed, ^T returns the value of the character as always. If no character has been typed, ^T immediately returns a value of -1 and execution continues without waiting for a character.																
ET&64	Detach flag (See appendices).																

(continued on next page)



**Table 5–30 (Cont.) Mode Control Flags**

Command	Function
ET&128	"Abort-on-error" bit. Initially set, when TECO starts up; cleared each time TECO issues its asterisk prompt. When this bit is set: 1) all informational messages are suppressed, 2) any <CTRL/C> causes the immediate termination of TECO, and 3) any error causes the termination of TECO after the error message is printed.
ET&256	If this bit is set, all lines output to the terminal are truncated to the terminal's width if needed. (RSTS/E, RSX-11, and VAX/VMS only. On VAX/VMS, this bit reflects and can change the state of the terminal characteristic WRAP/NOWRAP; TECO restores the terminal characteristic to its original state upon exit.)
ET&512	If this bit is set, the scope "WATCH" feature of TECO is present and your terminal is a scope type terminal. This bit is a read-only bit; its state cannot be altered. (See Section 5.17.)
ET&1024	If this bit is set, the refresh scope "WATCH" feature of TECO is present and a refresh scope is available. This bit is a read-only bit; its state cannot be altered. (See Section 5.17.)
ET&4096	<p>This bit reflects and can change the state of the terminal characteristic (maintained by the operating system) recording whether the terminal is capable of handling eight-bit character codes. Upon entry to TECO, this bit reflects the characteristic as recorded with the operating system at that time. If you change this bit, the operating system is directed to change its recording of the characteristic to match.</p> <p>Because the data manipulated (edited) by TECO can consist of all 256 possible byte codes, the way data characters are displayed (typed out) at the terminal varies depending upon the setting of the 4096 ET bit.</p> <p>If the bit is clear (meaning that TECO understands your terminal to be 7-bit), valid DEC multi-national codes are typed out as &lt;xy&gt; (where xy is the corresponding LK201 keyboard compose sequence).</p> <p>If the bit is set (meaning that TECO understands your terminal to be 8-bit), valid DEC multi-national codes are typed out as themselves.</p> <p>In either case, unprintable or illegal codes in the 128 to 255 range are typed out as [ab] (where ab is the corresponding hexadecimal code). The scope watch feature (screen buffer display, controlled by the W commands) always uses the &lt;xy&gt; and [ab] notations.</p>

(continued on next page)

## Command Descriptions

### 5.16 Mode Control Flags

Table 5–30 (Cont.) Mode Control Flags

Command	Function
ET&8192	<p>Accent grave as ESCape surrogate. [TECO-11 only]</p> <p>If this bit is set, TECO recognizes the ` (accent grave) character as an ESCAPE surrogate. That is, an ` character typed at the terminal will be recognized as a command &lt;DELIM&gt; character, and passed to TECO as an ESCape. (This interpretation applies only to &lt;DELIM&gt;s typed at the terminal; ESCape characters must still be used in macros and indirect files.) When an ESCape surrogate is set, an ESCape received by TECO echos as accent grave; when none is set, an ESCape received by TECO echos as dollar sign. This feature is provided for the benefit of certain newer terminals which lack an ESCape key. (See also the Introduction, and the EE flag.)</p>
ET&32768	<p>If this bit is set and a &lt;CTRL/C&gt; is typed, the bit is turned off, but execution of the current command string is allowed to continue. This allows a TECO macro to detect typed &lt;CTRL/C&gt;s. In TECO-8, this bit is the 2048's bit rather than the 32768's bit.</p>
	<p>The initial setting of ET is operating system dependent (See appendices). In addition, some of the ET bits are automatically turned off by certain error conditions.</p>
EU	<p>The upper/lower case flag.</p> <p>If n is -1, no case flagging of any type is performed on type out, lower case characters are output as lower case characters. If n is 0, lower case characters are flagged by outputting a ' (quote) before the lower case character and the lower case character is output in upper case; upper case characters are unchanged. If n is +1, upper case characters are flagged by outputting a ' (quote) before each one and then the upper case character is output; lower case characters are output as their upper case equivalents.</p> <p>The initial value of the EU flag is -1 if TECO can tell from the operating system that the user's terminal supports display of lower case characters; otherwise the initial value is 0. Consult the appendices for more details.</p>
EV	<p>The edit verify flag is decoded just like the ES flag. Just before TECO prints its prompting *, the EV flag is checked. If it is non-zero the lines to be viewed are printed on the terminal.</p>
	<p>The initial value of the EV flag is 0. [TECO-11 only]</p>
^X	<p>The search mode flag. [Not in TECO-8]</p> <p>If ^X is 0, the text argument in a search command will match text in the text buffer independent of case in either the search argument or the text buffer. The lower case alphabets match the upper case alphabets, and "", "{", "   ", "}", "~" match "@", "[", "\", "]", "^" respectively. In addition, DEC multinational characters having diacritical marks are regarded as matching the corresponding characters without diacritical marks.</p>

(continued on next page)

Table 5–30 (Cont.) Mode Control Flags

Command	Function
	<p>If ^X is -1, the search will succeed only if the text argument is identical to text in the text buffer. (This means that lower case does NOT match upper case, nor does a character having a diacritical mark match one without).</p> <p>The initial value of the ^X flag is 0. (For control of caret or uparrow treatment in search strings, see the ED&amp;1 bit above.)</p>

## 5.17 Scope Commands

The W command (scope "WATCH") is present in most implementations of TECO. There are two different variations of the W command. Neither, one, or both may be present. ET flag Bits 9 and 10 indicate which variation(s) are configured and can be used.

### 5.17.1 Video Terminal Scope Commands

If the VT support is present and your terminal is a video terminal (such as a VT05, VT52, or VT100), ET flag Bit 9 (value 512) will be on.

Table 5–31 Video Terminal Watch Commands

Command	Function
-1W	Refresh the terminal's screen to show the contents of the text buffer.
-nW	Tell the video terminal screen refresher that the top n-1 lines of the screen have been altered. The screen refresher will completely redraw the top n-1 lines of the screen upon the next -1W command.
nW	Place the default cursor line at line n of the screen. The initial default cursor line is line 16. This command makes the window support forget the screen image and any special associated modes (SEEALL, MARK, HOLD).
0W	Equivalent to "16W".
W	Forget screen image and special scope modes.
-1000W	Forget that output was done. Normally, if the user outputs to the terminal with a command such as T, n^T, or ^A, TECO will believe that the window needs updating, and upon the next -1W command, TECO will refresh the entire window display. Issuing the -1000W command informs TECO that the output command did not destroy the window. [TECO-11 only]

The :W commands are used to interrogate and set (see m,n:W below) video terminal status information, as well as implement some of the more advanced features of the video terminal "WATCH" functions.

## Command Descriptions

### 5.17 Scope Commands

**Table 5–32 Video Terminal Status Commands**

Command	Function																
0:W	<p>Return a number representing the type of scope in use as the editing terminal. Current values are:</p> <table> <tr> <th>value</th><th>scope type</th></tr> <tr> <td>0</td><td>VT52</td></tr> <tr> <td>1</td><td>VT61 [TECO-10 only]</td></tr> <tr> <td>2</td><td>VT100 in VT52 mode</td></tr> <tr> <td>4</td><td>VT100 in ANSI mode</td></tr> <tr> <td>6</td><td>VT05</td></tr> <tr> <td>8</td><td>VT102 in ANSI mode</td></tr> <tr> <td>10</td><td>VK100 (GIGI)</td></tr> </table>	value	scope type	0	VT52	1	VT61 [TECO-10 only]	2	VT100 in VT52 mode	4	VT100 in ANSI mode	6	VT05	8	VT102 in ANSI mode	10	VK100 (GIGI)
value	scope type																
0	VT52																
1	VT61 [TECO-10 only]																
2	VT100 in VT52 mode																
4	VT100 in ANSI mode																
6	VT05																
8	VT102 in ANSI mode																
10	VK100 (GIGI)																
:W	Equivalent to 0:W																
1:W	<p>Return or set the horizontal size of the user's editing scope. This number represents the number of character positions available horizontally along the face of the scope.</p> <p>If this value is changed (see m,n:W below), the operating system is automatically informed of the terminal's new width setting, and the appropriate control sequence is issued to physically change the terminal's screen mode (80 versus 132 column mode) if needed.</p>																
2:W	Return or set the vertical size of the user's editing scope. This number represents the number of lines of text that can appear on the screen of the terminal. This number is affected by use of the m,7:W command (set scrolling region).																
3:W	Return or set SEEALL mode. 0 represents off and -1 represents on. In SEEALL mode, a visible indication is shown in the window for every character, including characters that normally don't print.																
4:W	Return or set "mark" status of window support. 0 means that no mark has been set. A value of n means that a mark has been set at buffer ("dot") position n-1. This status is used by software that uses the window support and by the support itself in the case of scopes that support reverse video.																
5:W	Return or set the hold mode indicator. 0 means off, -1 means hold whole screen, and a positive value, n, means hold all but top and bottom n lines. If hold mode is on, then scrolling is inhibited until the cursor is about to run off either end of the screen. This makes the window display more palatable on terminals on a slow line. If hold mode is on, the window support will scroll the window as necessary in an attempt to keep the cursor centered.																
6:W	Returns buffer pointer position of character that was in the upper left hand corner of the window as of the last -1W command.																

(continued on next page)

Table 5–32 (Cont.) Video Terminal Status Commands

Command	Function
7:W	<p>Return or set the number of lines in the scrolling (command dialogue) portion of the screen. If n is zero, then split screen scrolling is disabled. When split screen scrolling is enabled, n lines (as specified) are reserved at the bottom of the screen to scroll the terminal interaction. The remainder of the screen is used as a display window into the text buffer, and is automatically updated by TECO immediately before each command prompt. This feature functions only on terminals capable of split screen scrolling, such as members of the VT100 and VT200 families.</p> <p>The value of n must be greater than 1, and scope height minus n must be greater than 9.</p> <p>Whenever the scrolling region's size is modified (that is, whenever an m,7:W command is executed), TECO alters the scope's height (2:W) accordingly. For example, if 2:W is currently returning a value of 24, then after a 5,7:W command 2:W will return a value of 19. Executing a 0,7:W will restore 2:W to 24.</p>
m,n:W	<p>Sets the entity represented by n:W to m and returns a value. If the new setting has been accepted, the returned value is m. Elsewise, the returned value is either the old value associated with n:W or whatever new setting was actually set. In all cases, the returned value reflects the new current setting. Specific operating systems may put restrictions on the valid values for m.</p> <p>Note that each m,n:W command returns a value, even if your only intent is to set something. Good programming practice suggests following any command which returns a value with &lt;DELIM&gt; or ^  if you don't intend that value to be passed to the following command.</p>

### 5.17.2 Refresh Scope Commands

If refresh scope support is present and a refresh scope is available (such as a VS60 or a VR12), bit value 1024 of the ET flag will be on.

Table 5–33 Refresh Scope Watch Commands

Command	Function
W	Update the refresh scope screen to reflect the contents of the text buffer surrounding the text pointer ("dot").
0W	Turn off the refresh scope display.
nW	Set the number of lines to be displayed around the text pointer to n.

## 5.18 Programming Aids

In addition to the command string editing capabilities described in Chapter 4, TECO includes various features to facilitate programming. These are described in the following sections.

## Command Descriptions

### 5.18 Programming Aids

#### 5.18.1 Text Formatting

The characters carriage return, line feed, and space are ignored in command strings, except when they appear as part of a text argument. Numeric values are not affected. (Inserting a space between digits within a digit string may cause unpredictable results). These characters may be inserted between any two TECO commands to lend clarity to a long command string. The carriage return/line feed combination is particularly useful for typing command strings which are too long to fit on a single line.

If the character form feed is encountered in a command string and it is not part of a text argument, a form feed is output to the terminal. This can be used to format terminal output. On TECO-10, execution of the form feed command will clear the screen if TECO is in scope command string editing mode (2&ET on).

#### 5.18.2 Comments

One of the most powerful features of TECO is its ability to store very long command strings so that a given sequence of commands may be executed whenever needed. Long command strings may be thought of as editing programs and, like any other type of program, they should be documented by means of comments.

Comments may be inserted between any two commands by using a tag construction of the form:

!THIS IS A COMMENT!

Comments may contain any number of characters and any characters except the special characters. Thus a long TECO macro might look like:

```
TECO commands !This comment describes line 1!
TECO commands !This comment describes line 2!
more commands
more commands !end of comment string!
```

Do not use <TAB> characters to format long command strings! Only <space>, <CR>, and <LF> can be used to format command strings since <TAB> is an insertion command.

Good TECO code is well structured and adequately commented. Unfortunately, massive comments in a TECO macro tend to slow execution, especially if they appear within text scanned by GOTOs or unsatisfied conditionals. Unless speed is not a goal, it is common practice in larger TECO programs to strip out comments before loading up TECO macros. Thus the TECO program can be adequately commented, yet still run efficiently.

A large TECO program can start by placing a comment stripper in a Q-register, say Q-register C. Then it can successively put subroutines (macro text) into the text buffer, do an MC, and load the appropriate Q-register with the resulting text buffer, until all the subroutines have been loaded. Finally, Q-register C can be zeroed and the program started.

In order for you to strip the comments without losing essential tags, you must make a convention for the format of your comments so that your comment stripper can distinguish them from tags. There are two common conventions. In one, the first character in every comment after the initial ! is some distinctive character, such as \*. In the other, all tags start in the left margin and all comments are embedded within the text. Any large comment that wants to be

on a line by itself starts with a <space> before the !. Both methods allow for readable code and easy comment stripping.

### 5.18.3 Messages

The <CTRL/A> command may be used to print out a statement at any point during the execution of a command string. The <CTRL/A> command has the general form:

^Atext<CTRL/A>

or

@^A/text/

The first ^A is the actual command, which may be entered by striking the control key and the A key simultaneously or by typing a caret (uparrow) followed by an A character. The second <CTRL/A> character of the first form shown is the command terminator, which must be entered by typing the control key and the A key simultaneously. In the second form, the second occurrence of the delimiting character (shown as slash in the example) terminates the message. Upon execution, this command causes TECO to print the specified message at the terminal.

The ^Amessage<CTRL/A> command is particularly useful when it precedes a command whose numeric argument contains ^T or ^F characters. The message may contain instructions notifying the user as to what sort of input is required.

### 5.18.4 Tracing

A question mark entered between any two commands in a command string causes TECO to print all subsequent commands at the terminal as they are executed. Commands will be printed as they are executed until another question mark character is encountered or the command string terminates.

### 5.18.5 Convenience Characters

In addition to the characters mentioned in Section 5.18.1, there are several characters which have no special meaning to TECO but which may be used to help format your TECO programs and command strings. Judicious use of these commands will make your program easier to read and maintain. These characters are described in the table below:

**Table 5–34 Convenience Characters**

Character	Meaning
<NULL>	A null (ASCII 0) encountered as a TECO command will be ignored. Numeric values are not affected. A null read in from an input file will be discarded (except under RSX-11 and VAX/VMS). A null typed in from a terminal will be ignored.

(continued on next page)



## Command Descriptions

### 5.18 Programming Aids

**Table 5–34 (Cont.) Convenience Characters**

Character	Meaning
<ESCAPE>	An ESCape that is executed as a TECO command (as distinct from an immediate action command or an ESCape that is part of the syntax of some other TECO command) is ignored by TECO. However, any pending numeric values are discarded. This command is useful for discarding the value returned from a command (such as n%q or m,n:W) when you don't want that value to affect execution of the following command.
^[	Same as <ESCAPE>. Like any other TECO command that is a control character, ESCape may be entered in up-arrow mode. Sometimes easier to type, ^[ is also useful on systems whose line-printer spoolers do not visibly print the ESCape character.  Any pending numeric values are discarded. This command is useful for discarding the value returned from a command (such as n%q or m,n:W) when you don't want that value to affect execution of the following command.
<ALTMODE>	Some older terminals may have no ESCape key, but instead a key labelled ALTMODE or PREFIX which sends TECO a character whose ASCII value is 175 or 176. In such a case, TECO will treat these characters as if they were typed in as an ESCape (octal 33), provided lower to upper case conversion is enabled.
\$	(dollar sign) Same as <ESCAPE>, but as a command only, not as a string terminator. [TECO-10 only]

Note that <TAB> and <FORM FEED> are valid TECO commands and must not be used as aids to formatting TECO programs.

#### 5.18.6 Memory Expansion

The nEC command can be used to make TECO reclaim lost space after it had expanded memory usage. nEC tells TECO to expand or contract until it uses nK words of memory. If this is not possible, then TECO's memory usage does not change. The 0EC command tells TECO to shrink back to its original size (use the least amount of memory possible). [TECO-10 only]

#### 5.18.7 Case Control

The <CTRL/V> and <CTRL/W> TECO commands are used to specify automatic case control for alphabetic characters typed into strings.

**Table 5–35 Case Control Characters**

Character	Meaning
^V	<CTRL/V> puts TECO into lower case conversion mode. In this mode, all alphabetic characters in string arguments are automatically changed to lower case. This mode can be overridden by explicit case control within the search string. This command makes all strings behave as if they began with a ^V^V. [TECO-10 only]

(continued on next page)



Table 5–35 (Cont.) Case Control Characters

Character	Meaning
<code>^W</code>	<CTRL/W> puts TECO into upper case conversion mode. In this mode, all alphabetic characters in string arguments are automatically changed to upper case. This mode can be overridden by explicit case control within the search string. This command makes all strings behave as if they began with <code>^W^W</code> . [TECO-10 only]
<code>0^V</code>	Returns TECO to its original mode. No special case conversion occurs within strings except those case conversions that are explicitly specified by <code>^V</code> and <code>^W</code> string build constructs located within the string. [TECO-10 only]
<code>0^W</code>	Same as <code>0^V</code> . [TECO-10 only]

## 5.19 Manipulating Large Pages

TECO is designed to operate most efficiently when editing files that contain no more than several thousand characters per page. (TECO storage includes Q-register storage and buffer space. The size of the text storage area is dynamic and depends on the amount of available memory.) If any page of an input file is too large to fit in the text area, the TECO input commands will terminate reading that page into memory when the first line feed is encountered after a point that the buffer is 3/4 full. (See appendices for details.) You can make room by positioning the pointer past a section of text at the beginning of the buffer and moving that section out of the buffer with the commands:

`0,.PW0,.K`

It is sometimes advantageous to restrict the amount of the file that is present in the buffer. For example, each insert and delete command must move the entire text that is beyond the point of insertion or deletion. An operation that does many small inserts or deletes may therefore run extremely slowly if the text buffer is large. Such an operation can be sped up substantially by reading the input file with `n:A` commands and explicitly writing the processed text.

## 5.20 Techniques and Examples

The most elementary TECO application, described in Chapter 1 of this manual, is creating and editing ASCII files on-line. The user enters short command strings, often consisting of a single command, and proceeds from task to task until the file is completely edited.

Since every editing job is simply a long sequence of TECO commands, you may accomplish an entire job with one long command string made up of all the short command strings placed end to end with the intervening double <DELIM> characters removed. A long command string that performs a certain editing task can be considered a TECO "editing program". Editing programs may be written (using TECO) and stored in the same manner as any other ASCII file. Whenever the program is needed, it may be read into the buffer as text, stored in a Q-register, and executed by an `Mq` command.

## Command Descriptions

### 5.20 Techniques and Examples

For more complex editing jobs, you may want to write and maintain a collection of specialized "editing subroutines." TECO subroutines can perform such elementary functions as replacing every occurrence of two or more consecutive spaces with a tabulation character, for example, or ensuring that words are not hyphenated across a page boundary. When an editing problem arises, you can load the right combination of subroutines into various Q-registers, augment them with additional commands if necessary, and call them by a "mainline" command string.

Editing subroutines are essentially macros; that is, sequences of commands which perform commonly required editing functions. The most powerful application of TECO is the creation and use of a macro library. As you perform an editing job, look for sequences of operations which might be required in future editing assignments. Load all of the TECO commands required to perform such an operation into a Q-register. When the job is finished, write the contents of the Q-register onto an output file (via the buffer) and save it in the macro library. The nMq and m,nMq commands, which were designed to facilitate use of macros, permit run-time numeric arguments to be passed to a macro.

TECO macros can preserve the user's radix, flag values, etc. By using the Q-register push-down list, the macro can save and then restore values and/or text. For example:

#### Example 5-1 Saving and Restoring values and/or text

```
[0 [1 [2      ! Save contents of Q-registers 0, 1 and 2 !
+0U0         ! Put any calling argument into Q-register 0 !
10U1         ! Put a 10 (if radix is decimal) or 8 (if radix is
              ! octal) into Q-register 1 !
^D           ! Ensure that the current radix is now decimal !
E0U2         ! Save the case flagging flag !
-1EU        ! Ensure no case flagging !
Q0"E 3U0 '   ! Default calling argument to 3 !
.
.
.
Q2EU        ! Restore the case flagging flag !
10-Q1"N ^O ' ! Restore radix as octal if needed !
]2 ]1 ]0     ! Restore contents of Q-registers 2, 1, and 0 !
```

The EI command is particularly useful for executing macros from a library, since with it they may be read without disturbing the current input file. This makes it unnecessary to plan in advance which macros might be needed; it also saves Q-register storage space. You can retrieve two kinds of TECO command files with an EI command: a file containing a TECO command that loads the macro into a Q-register for later use, or a file containing just the macro (which must be retrieved with EI each time it is used).

The following examples are intended to illustrate some of the techniques discussed above. It would not be practical to include examples of the use of every TECO command, since most of the commands apply to many diverse situations. Instead, you are encouraged to experiment with the individual commands on scratch files.

Assume that there is a file named PROG.DAT on the system disk and that this file contains data in the following form:

```
AB <FF> CD <FF> EF <FF> GH <FF> IJ <FF> KL <FF> MN <FF> OP
```

## Command Descriptions

### 5.20 Techniques and Examples

where each of the letters A, B, C etc., represents 20 lines of text and <FF> represents a form feed character. The user intends to rearrange the file so that it appears in the following format:

AOB <FF> D <FF> MN <FF> EF <FF> ICJ <FF> KL <FF> P <FF> GH

The following sequence of commands will achieve this rearrangement. (Search command arguments are not listed explicitly.)

#### Example 5-2 Splitting, Merging, And Rearranging Files

```
Start TECO.
* 2ED`` Allow all Y commands.
* EBPROG.DAT`Y`` Specify input file and get first page.
* NC`` Search for a character string in C, writing A
and B on the output file.
* J20X1`` Save all of C in Q-register 1.
* 20K`` Delete C from the buffer.
* NG`` Search for a character string in G, writing
D, E, and F on the output file.
* HX2`` Save G and H in Q-register 2.
* Y`` Delete GH from the buffer and read IJ.
* 20L`` Move the pointer to the beginning of J.
* G1`` Insert C, which was stored in Q-register 1.
* NM`` Search for a character string in M, writing
ICJ and KL on the output file.
* HX1`` Save MN in Q-register 1 (the previous
contents is overwritten).
* Y`` Delete MN and read OP
* J20X3`` Save all of O in Q-register 3.
* 20K`` Delete O from the buffer.
* PWHK`` Write P onto the output file, appending a
form feed, and clear the text buffer.
* G2`` Bring GH into the buffer from Q-register 2.
* HPEF`` Write GH on the output file and close it.
* EBPROG.DAT`Y`` Open the partially revised file.
* 20L`` Move the pointer to the beginning of B.
* G3`` Insert all of O from Q-register 3.
* ND`` Search for a character string in D writing
AOB on the output file.
* PWHK`` Write D on the output file and clear buffer.
* G1`` Bring all of MN from Q-register 1 into the
buffer.
* EX`` Write MN onto the output file, then close the
file and exit.
```

At this point the file has been rearranged in the desired format. Of course, this rearrangement could have been accomplished in fewer steps if the commands listed above had been combined into longer command strings. Note that the asterisks shown at the left margin in this example are generated by TECO, and not typed by the user.

Assume, now, that the same input file, containing data in the form:

AB <FF> CD <FF> EF <FF> ... <FF> OP

is to be split into two separate files, with the first file containing AB <FF> CD and the second file containing KL <FF> M, while the rest of the data is to be discarded. The following commands could be used to achieve this rearrangement:

(continued on next page)

## Command Descriptions

### 5.20 Techniques and Examples

#### Example 5–2 (Cont.) Splitting, Merging, And Rearranging Files

```
Start TECO.
* 2ED`` Allow all Y commands.
* ERFILE`EWFIL1`` Open the input file and the first output
file.
* Y`` Read AB into the buffer.
* P`` Write AB <FF> onto the output file and read
CD into the buffer.
* HPEF`` Write CD onto the output file (without
appending a form feed), and close the first
output file.
* _K`` Search for a character string in K. After
this command has been executed, the buffer
will contain KL. No output is generated.
* EWFIL2`P`` Open the second output file and write KL onto
it. Read MN into the buffer.
* 20L0,.P`` Move the pointer to the end of M, then write
M onto the output file.
* EF`` Close the output file.
* HKEX`` Clear the buffer and exit.
```

As a final example of file manipulation techniques, assume that there are two files. One file is MATH.ONE, which contains information in the form:

AB <FF> CD <FF> EF <FF> GH <FF> IJ <FF> KL

and the other is MATH.TWO, which contains:

MN <FF> OP <FF> QR

If both of these files are stored on DK1, the following sequence of commands may be used to merge the two files into a single file, MATH.NEW, which contains all of MATH.TWO followed by the latter half of file MATH.ONE in the following format:

MN <FF> OP <FF> QR <FF> GH <FF> IJ <FF> KL

```
Start TECO.
* 2ED`` Allow all Y commands.
* ERDK1:MATH.TWO`` Open the first input file.
* EWMATH.NEW`` Open the output file on the default device.
* Y`` Read MN into the text buffer.
* NR`` Search for a character string in R, writing
MN and OP onto the output file.
* PW`` Write QR onto the output file, appending a
form feed.
* ERDK1:MATH.ONE`` Open the second input file.
* HKY`` Read AB into the buffer. QR is over-written.
* _G`` Search for a character string in G, deleting
AB, CD, and EF, leaving GH in the buffer.
* NK`` Search for a character string in K, writing
GH and IJ on the output file, leaving KL in
the buffer
* HPEFHKEX`` Write KL onto the output file (without
appending a form feed) and close the file,
then exit.
```

## Command Descriptions

### 5.20 Techniques and Examples

Assume that TECO is running and that the buffer contains many short lines of text beginning with an alphabetic character at the left margin (i.e., immediately following a line feed). The lines might consist of names in a roster, for example, or entries in an index. The following command string will rearrange the lines into rough alphabetical order, grouping all lines which begin with the character "A" at the beginning of the page, followed by all lines with "B", and so on. Note that the algorithm could be extended to place the entries in strict alphabetical order by having it loop back to perform the same sorting operation on successive characters in each line.

#### Example 5-3 Alphabetizing by Interchange Sort

!START! J 0AUA	!Load first character of first line into Q-register A !
!CONT! L 0AUB	!Load first character of next line into Q-register B !
QA-QB"G XA K -L GA 1UZ '	!If A>B, switch the lines and set a flag (Q-register Z) !
QBUA	!Load B into A !
L Z-."G -L @O/CONT/ '	!Loop back if there is another line in the buffer !
QZ"G 0UZ @O/START/ '	!Repeat if a switch was made on the last pass !

The same algorithm can be coded in a more structured way as follows:

0UZ	!clear repeat flag!
<J 0AUA L	!Load first character of first line into Q-register A !
<0AUB	!Load first character of next line into Q-register B !
QA-QB"G XA K -L GA -1UZ '	!If A>B, switch the lines and set a flag !
QBUA	!Load B into A !
L .-Zi>	!Loop back if there is another line in the buffer !
QZi>	!Repeat if a switch was made on the last pass !

This example is a bit shorter and does not use any GOTOs. It will also run somewhat faster.



# A

---

## Octal, Decimal & Hexadecimal ASCII Character Set

This appendix contains the ASCII character set table as defined by the Digital Multi-national Character set (MCS).

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[00]	000	0.	<NUL>	Null
[01]	001	1.	<SOH>	^A, Control-A, Start-of-Header
[02]	002	2.	<STX>	^B, Control-B, Start-of-Text
[03]	003	3.	<ETX>	^C, Control-C, End-of-Text
[04]	004	4.	<EOT>	^D, Control-D, End-of-Transmission
[05]	005	5.	<ENQ>	^E, Control-E, Enquiry
[06]	006	6.	<ACK>	^F, Control-F, Acknowledge
[07]	007	7.	<BEL>	^G, Control-G, Bell
[08]	010	8.	<BS>	^H, Control-H, Backspace
[09]	011	9.	<TAB>	^I, Control-I, Horizontal tab
[0A]	012	10.	<LF>	^J, Control-J, Line feed
[0B]	013	11.	<VT>	^K, Control-K, Vertical tab
[0C]	014	12.	<FF>	^L, Control-L, Form feed
[0D]	015	13.	<CR>	^M, Control-M, Carriage return
[0E]	016	14.	<SO>	^N, Control-N, Shift Out
[0F]	017	15.	<SI>	^O, Control-O, Shift In
[10]	020	16.	<DLE>	^P, Control-P, Data Link Escape
[11]	021	17.	<DC1>	^Q, Control-Q, Device Control 1, <XON>
[12]	022	18.	<DC2>	^R, Control-R, Device Control 2
[13]	023	19.	<DC3>	^S, Control-S, Device Control 3, <XOFF>
[14]	024	20.	<DC4>	^T, Control-T, Device Control 4
[15]	025	21.	<NAK>	^U, Control-U, Negative Acknowledge
[16]	026	22.	<SYN>	^V, Control-V, Synchronization
[17]	027	23.	<ETB>	^W, Control-W, End-of-Text-Block
[18]	030	24.	<CAN>	^X, Control-X, Cancel
[19]	031	25.	<EM>	^Y, Control-Y, End-of-Medium
[1A]	032	26.	<SUB>	^Z, Control-Z, Substitute
[1B]	033	27.	<ESC>	^[, Control-[, Escape
[1C]	034	28.	<FS>	^\\, Control-\\, File Separator
[1D]	035	29.	<GS>	^], Control-], Group Separator
[1E]	036	30.	<RS>	^^, Control-^, Record Separator
[1F]	037	31.	<US>	^_, Control-_, Unit Separator

(continued on next page)



## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[20]	040	32.		Space
[21]	041	33.	!	Exclamation point
[22]	042	34.	"	Double quotation mark
[23]	043	35.	#	Number sign
[24]	044	36.	\$	Dollar sign
[25]	045	37.	%	Percent sign
[26]	046	38.	&	Ampersand
[27]	047	39.	'	Single quote, apostrophe
[28]	050	40.	(	Left parenthesis
[29]	051	41.	)	Right parenthesis
[2A]	052	42.	*	Asterisk
[2B]	053	43.	+	Plus
[2C]	054	44.	,	Comma
[2D]	055	45.	-	Hyphen, minus
[2E]	056	46.	.	Period, decimal point
[2F]	057	47.	/	Slash, slant, solidus
[30]	060	48.	0	Digit 0
[31]	061	49.	1	Digit 1
[32]	062	50.	2	Digit 2
[33]	063	51.	3	Digit 3
[34]	064	52.	4	Digit 4
[35]	065	53.	5	Digit 5
[36]	066	54.	6	Digit 6
[37]	067	55.	7	Digit 7
[38]	070	56.	8	Digit 8
[39]	071	57.	9	Digit 9
[3A]	072	58.	:	Colon
[3B]	073	59.	;	Semicolon
[3C]	074	60.	<	Left angle, less than
[3D]	075	61.	=	Equals
[3E]	076	62.	>	Right angle, greater than
[3F]	077	63.	?	Question mark

(continued on next page)

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[40]	100	64.	@	Commercial at
[41]	101	65.	A	Upper case A
[42]	102	66.	B	Upper case B
[43]	103	67.	C	Upper case C
[44]	104	68.	D	Upper case D
[45]	105	69.	E	Upper case E
[46]	106	70.	F	Upper case F
[47]	107	71.	G	Upper case G
[48]	110	72.	H	Upper case H
[49]	111	73.	I	Upper case I
[4A]	112	74.	J	Upper case J
[4B]	113	75.	K	Upper case K
[4C]	114	76.	L	Upper case L
[4D]	115	77.	M	Upper case M
[4E]	116	78.	N	Upper case N
[4F]	117	79.	O	Upper case O
[50]	120	80.	P	Upper case P
[51]	121	81.	Q	Upper case Q
[52]	122	82.	R	Upper case R
[53]	123	83.	S	Upper case S
[54]	124	84.	T	Upper case T
[55]	125	85.	U	Upper case U
[56]	126	86.	V	Upper case V
[57]	127	87.	W	Upper case W
[58]	130	88.	X	Upper case X
[59]	131	89.	Y	Upper case Y
[5A]	132	90.	Z	Upper case Z
[5B]	133	91.	[	Opening bracket
[5C]	134	92.	\	Backslash, reverse slant
[5D]	135	93.	]	Closing bracket
[5E]	136	94.	^	Circumflex, caret, uparrow
[5F]	137	95.	_	Underline, underscore

(continued on next page)

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[60]	140	96.	`	Accent grave
[61]	141	97.	a	Lower case a
[62]	142	98.	b	Lower case b
[63]	143	99.	c	Lower case c
[64]	144	100.	d	Lower case d
[65]	145	101.	e	Lower case e
[66]	146	102.	f	Lower case f
[67]	147	103.	g	Lower case g
[68]	150	104.	h	Lower case h
[69]	151	105.	i	Lower case i
[6A]	152	106.	j	Lower case j
[6B]	153	107.	k	Lower case k
[6C]	154	108.	l	Lower case l
[6D]	155	109.	m	Lower case m
[6E]	156	110.	n	Lower case n
[6F]	157	111.	o	Lower case o
[70]	160	112.	p	Lower case p
[71]	161	113.	q	Lower case q
[72]	162	114.	r	Lower case r
[73]	163	115.	s	Lower case s
[74]	164	116.	t	Lower case t
[75]	165	117.	u	Lower case u
[76]	166	118.	v	Lower case v
[77]	167	119.	w	Lower case w
[78]	170	120.	x	Lower case x
[79]	171	121.	y	Lower case y
[7A]	172	122.	z	Lower case z
[7B]	173	123.	{	Opening brace
[7C]	174	124.		Vertical line
[7D]	175	125.	}	Closing brace
[7E]	176	126.	~	Tilde, overline, high line
[7F]	177	127.	<DEL>	Delete, rubout

(continued on next page)

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[80]	200	128.		
[81]	201	129.		
[82]	202	130.		
[83]	203	131.		
[84]	204	132.	<IND>	Index
[85]	205	133.	<NEL>	Next Line
[86]	206	134.	<SSA>	Start Selected Area
[87]	207	135.	<ESA>	End Selected Area
[88]	210	136.	<HTS>	Horizontal Tab Set
[89]	211	137.	<HTJ>	
[8A]	212	138.	<VTS>	
[8B]	213	139.	<PLD>	
[8C]	214	140.	<PLU>	
[8D]	215	141.	<RI>	Reverse Index
[8E]	216	142.	<SS2>	Single Shift 2
[8F]	217	143.	<SS3>	Single Shift 3
[90]	220	144.	<DCS>	Device Control String
[91]	221	145.	<PU1>	
[92]	222	146.	<PU2>	
[93]	223	147.	<STS>	Set Transmit State
[94]	224	148.	<CCH>	
[95]	225	149.	<MW>	
[96]	226	150.	<SPA>	Start Protected Area
[97]	227	151.	<EPA>	End Protected Area
[98]	230	152.		
[99]	231	153.		
[9A]	232	154.		
[9B]	233	155.	<CSI>	Control Sequence Introducer
[9C]	234	156.	<ST>	String Terminator
[9D]	235	157.	<OSC>	Operating System Command
[9E]	236	158.	<PM>	Privacy Message
[9F]	237	159.	<APC>	Application Program Command

(continued on next page)

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[A0]	240	160.		-
[A1]	241	161.	¡	Inverted exclamation mark
[A2]	242	162.	¢	Cent sign
[A3]	243	163.	£	Pound sterling sign
[A4]	244	164.		reserved for future
[A5]	245	165.	¥	Yen
[A6]	246	166.		reserved for future
[A7]	247	167.	§	Section sign
[A8]	250	168.	¨	General currency sign
[A9]	251	169.	©	Copyright sign
[AA]	252	170.	ª	Feminine ordinal indicator
[AB]	253	171.	«	Angle quotation mark left
[AC]	254	172.		reserved for future
[AD]	255	173.		reserved for future
[AE]	256	174.		reserved for future
[AF]	257	175.		reserved for future
[B0]	260	176.	°	Degree sign
[B1]	261	177.	±	Plus/minus sign
[B2]	262	178.	<sup>2</sup>	Superscript 2
[B3]	263	179.	<sup>3</sup>	Superscript 3
[B4]	264	180.		reserved for future
[B5]	265	181.	μ	Micro sign, Greek mu
[B6]	266	182.	¶	Paragraph sign, pilcrow
[B7]	267	183.	·	Middle dot
[B8]	270	184.		reserved for future
[B9]	271	185.	<sup>1</sup>	Superscript 1
[BA]	272	186.	º	Masculine ordinal indicator
[BB]	273	187.	»	Angle quotation mark right
[BC]	274	188.	¼	Fraction one quarter
[BD]	275	189.	½	Fraction one half
[BE]	276	190.		reserved for future
[BF]	277	191.	¿	Inverted question mark

(continued on next page)

## Octal, Decimal & Hexadecimal ASCII Character Set

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[C0]	300	192.	À	A with accent grave
[C1]	301	193.	Á	A with accent aigu (acute)
[C2]	302	194.	Â	A with circumflex
[C3]	303	195.	Ã	A with tilde
[C4]	304	196.	Ä	A with umlaut (diæresis)
[C5]	305	197.	Å	A with ring
[C6]	306	198.	Æ	AE ligature
[C7]	307	199.	Ç	C with cedilla
[C8]	310	200.	È	E with accent grave
[C9]	311	201.	É	E with accent aigu (acute)
[CA]	312	202.	Ê	E with circumflex
[CB]	313	203.	Ë	E with umlaut (diæresis)
[CC]	314	204.	Ì	I with accent grave
[CD]	315	205.	Í	I with accent aigu (acute)
[CE]	316	206.	Î	I with circumflex
[CF]	317	207.	Ï	I with umlaut (diæresis)
[D0]	320	208.		reserved for future
[D1]	321	209.	Ñ	N with tilde
[D2]	322	210.	Ò	O with accent grave
[D3]	323	211.	Ó	O with accent aigu (acute)
[D4]	324	212.	Ô	O with circumflex
[D5]	325	213.	Õ	O with tilde
[D6]	326	214.	Ö	O with umlaut (diæresis)
[D7]	327	215.	×	OE ligature
[D8]	330	216.	Ø	O (letter) with slash
[D9]	331	217.	Ù	U with accent grave
[DA]	332	218.	Ú	U with accent aigu (acute)
[DB]	333	219.	Û	U with circumflex
[DC]	334	220.	Ü	U with umlaut (diæresis)
[DD]	335	221.	Ý	Y with umlaut (diæresis)
[DE]	336	222.		reserved for future
[DD]	337	223.	ß	German sharp s

(continued on next page)

**Table A–1 (Cont.) Hexadecimal, Octal & Decimal ASCII Character Set**

Hex	Oct	Dec	Char	Description
[E0]	340	224.	à	a with accent grave
[E1]	341	225.	á	a with accent aigu (acute)
[E2]	342	226.	â	a with circumflex
[E3]	343	227.	ã	a with tilde
[E4]	344	228.	ä	a with umlaut (diæresis)
[E5]	345	229.	å	a with ring
[E6]	346	230.	æ	ae ligature
[E7]	347	231.	ç	c with cedilla
[E8]	350	232.	è	e with accent grave
[E9]	351	233.	é	e with accent aigu (acute)
[EA]	352	234.	ê	e with circumflex
[EB]	353	235.	ë	e with umlaut (diæresis)
[EC]	354	236.	ì	i with accent grave
[ED]	355	237.	í	i with accent aigu (acute)
[EE]	356	238.	î	i with circumflex
[EF]	357	239.	ï	i with umlaut (diæresis)
[F0]	360	240.		reserved for future
[F1]	361	241.	ñ	n with tilde
[F2]	362	242.	ò	o with accent grave
[F3]	363	243.	ó	o with accent aigu (acute)
[F4]	364	244.	ô	o with circumflex
[F5]	365	245.	õ	o with tilde
[F6]	366	246.	ö	o with umlaut (diæresis)
[F7]	367	247.	÷	oe ligature
[F8]	370	248.	ø	o (letter) with slash
[F9]	371	249.	ù	u with accent grave
[FA]	372	250.	ú	u with accent aigu (acute)
[FB]	373	251.	û	u with circumflex
[FC]	374	252.	ü	u with umlaut (diæresis)
[FD]	375	253.	ý	y with umlaut (diæresis)
[FE]	376	254.		reserved for future
[FF]	377	255.	-	-





## Error Messages

TECO error messages consist of a three letter message preceded by a question mark (?) or preceded by ?TEC. A short description of the error optionally follows (dependent on the current value of the EH flag). Typing ? (question mark) immediately after an error message printout causes the command string to be printed up to and including the character which causes the error message. Typing \*q (asterisk, Q-register name) immediately after an error message printout saves the entire command string in the specified Q-register. This is especially useful for recovering mistyped insert commands. Both the ? and \*q facilities may be used when an error occurs.

TECO-11 also produces two warning messages. These messages do not abort the command and execution continues.

%Superseding existing file	Indicates that the file to be created as the result of an EW command already exists. If the output file is closed the old copy of the file will be deleted. The EK command may be used to "take back" the EW command.
%Search fail in iter	Indicates that a search command has failed inside iteration brackets. A ; (semi-colon) command immediately following the search command can typically be used to suppress this message. After printing the message, the iteration is terminated, i.e., TECO simulates a 0; command.

These error messages are listed alphabetically by their three-letter code. In general, these three-letter codes have the same meaning on all implementations, although not all error messages are produced by each implementation. The one-line error message given here is a paraphrasing of the message given, which may differ slightly from system to system.

?ARG, Improper Arguments

**Explanation:** Three arguments are given (a,b,c or H,c).

?BNI, > not in iteration

**Explanation:** There is a close angle bracket not matched by an open angle bracket somewhere to its left. (Note: an iteration in a macro stored in a Q-register must be complete within the Q-register.)

?CCL, CCL.SV not found or EG argument too long

**Explanation:** The EGcommand` command on OS/8 was unable to locate SYS:CCL.SV or the specified command has more than 46 characters.

## Error Messages

?CON, Confused use of conditionals

**Explanation:** Conditionals, parenthesized arguments, and iterations must be properly nested. The user probably used some construct like: N"E...(' where an iteration or parenthesized argument is begun in a conditional but not terminated in the same conditional.

?CPQ, Can't pop into Q-register

**Explanation:** A ] command has been executed and there is nothing saved on the Q-register push down list.

?DEV, Invalid device

**Explanation:** A file specification string in an E command contains an unknown device name.

?DTB, Delete too big

**Explanation:** An nD command has been attempted which is not contained within the current page.

?ERR, RSTS/E error message (RSTS/E only)

**Explanation:** Some RSTS/E monitor call failed. The error message text explains the error.

?FER, File Error

**Explanation:** The file specified in an ER, EW or EB command was not found.

?FNF, File not found "filespec"

**Explanation:** The requested input file could not be located. If this occurred within a macro the colon modified ER or EB command may be necessary.

?FUL, Output Command would have overflowed output device

**Explanation:** The page of text currently in the text buffer will not fit in the open output file. Until enough free space can be obtained on the output device the file may have to be split. An EF command to close the current output file, followed by a new EW command to a temporary file may be used. The files should be concatenated when the space problem is alleviated.

?ICE, Illegal ^E Command in Search Argument

**Explanation:** A search argument contains a ^E command that is either not defined or incomplete. The only valid ^E commands in search arguments are: ^EA, ^ED, ^EV, ^EW, ^EL, ^ES, ^E<NNN>, and ^E[A,B,C,...].

?IEC, Illegal character "x" after E

**Explanation:** An invalid E command has been executed. The E character must be followed by an alphabetic to form a legal E command (i.e., ER or EX).

?IFC, Illegal character "x" after F

**Explanation:** An invalid F command has been executed.

?IFN, Illegal character "x" in filename

**Explanation:** The filespec as an argument to one of the E commands is unacceptable to the system. The file specification must be appropriate to the system in use.

?IIA, Illegal insert arg

**Explanation:** A command of the form "nItext`" was attempted. Combining character (nI`) and text (Itext`) insertions is illegal.

?ILL, Illegal command "x"

**Explanation:** An attempt has been made to execute an invalid TECO command.

?ILN, Illegal number

**Explanation:** An 8 or 9 has been entered when the radix of TECO is set to octal.

?INP, Input error

**Explanation:** The system has reported an error attempting to read the current input file. The text buffer may be corrupt. This operation may be retried, but if the error persists, you may have to return to a backup file.

?IPA, Negative or 0 argument to P

**Explanation:** The argument preceding a P or PW command is negative or 0.

?IQC, Illegal " character

**Explanation:** One of the valid " commands did not follow the ". Refer to Section 5.14, Conditional Execution Commands, for the legal set of commands.

?IQN, Illegal Q-register name "x"

**Explanation:** An illegal Q-register name was specified in one of the Q-register commands.

?IRA, Illegal radix argument to ^R

**Explanation:** The argument to a ^R radix command must be 8, 10, or 16.

?ISA, Illegal search arg

**Explanation:** The argument preceding a search command is 0. This argument must not be 0.

?ISS, Illegal search string

**Explanation:** One of the search string special characters (^Q, ^V, ^W, etc.) would have modified the search string delimiter (usually ESCAPE).

?IUC, Illegal character "x" following ^

**Explanation:** The character following an ^ must have ASCII value between 100 and 137 inclusive or between 141 and 172 inclusive.

?MAP, Missing '

**Explanation:** Every conditional (opened with the " command) must be closed with the ' command.

?MEM, Memory overflow

**Explanation:** Insufficient memory available to complete the current command. Make sure the Q-register area does not contain much unnecessary text. Breaking up the text area into multiple pages might be useful. (See Section 5.19.)

## Error Messages

?MLA, Missing Left Angle Bracket

**Explanation:** There is a right angle bracket that has no matching left angle bracket. An iteration must be complete within the macro or command.

?MLP, Missing (

**Explanation:** There is a right parenthesis that is not matched by a corresponding left parenthesis.

?MRA, Missing Right Angle Bracket

**Explanation:** There is a left angle bracket that has no matching right angle bracket. An iteration must be complete within the macro or command.

?MRP, Missing )

**Explanation:** There is a right parenthesis that is not matched by a corresponding left parenthesis.

?MSC, Missing Start of Conditional

**Explanation:** A ' command (end of conditional) was encountered. Every ' command must be matched by a preceding " (start of conditional) command.

?NAB, No arg before ^\_

**Explanation:** The ^\_ command must be preceded by either a specific numeric argument or a command that returns a numeric value.

?NAC, No arg before ,

**Explanation:** A command has been executed in which a , is not preceded by a numeric argument.

?NAE, No arg before =

**Explanation:** The =, ==, or === command must be preceded by either a specific numeric argument or a command that returns a numeric value.

?NAP, No arg before )

**Explanation:** A ) parenthesis has been encountered and is not properly preceded by a specific numeric argument or a command that returns a numeric value.

?NAQ, No arg before "

**Explanation:** The " commands must be preceded by a single numeric argument on which the decision to execute the following commands or skip to the matching ' is based.

?NAS, No arg before ;

**Explanation:** The ; command must be preceded by a single numeric argument on which the decision to execute the following commands or skip to the matching > is based.

?NAU, No arg before U

**Explanation:** The U command must be preceded by either a specific numeric argument or a command that returns a numeric value.

?NCA, Negative argument to ,

**Explanation:** A comma was preceded by a negative number.

?NYA, Numeric argument with Y

**Explanation:** The Y command must not be preceded by either a numeric argument or a command that returns a numeric value.

?NYI, Not yet implemented

**Explanation:** A command was issued which tries to invoke a feature not available on this implementation of TECO.

?NFI, No file for input

**Explanation:** Before issuing an input command, such as Y, it is necessary to open an input file by use of a command such as ER or EB.

?NFO, No file for output

**Explanation:** Before issuing an output command such as N search or P it is necessary to open an output file by use of a command such as EW or EB.

?NPA, Negative or 0 argument to P

**Explanation:** A P command was preceded by a negative or 0 argument.

?NRO, No room for output

**Explanation:** The output device is too full to accept the requested output file.

?NYI, Not Yet Implemented

**Explanation:** A command was issued that is not yet implemented in this version of TECO.

?OFO, Output file already open

**Explanation:** A command has been executed which tried to create an output file, but an output file currently is open. It is typically appropriate to use the EC or EK command as the situation calls for to close the output file.

?OUT, Output error

**Explanation:** The system has reported an error attempting to do output to the output file. Make sure that output device did not become write locked. Use of the EF command (or EK if necessary) and another EW can be considered until the condition is fixed.

?PES, Attempt to Pop Empty Stack

**Explanation:** A ] command (pop off q-register stack into a q-register) was encountered when there was nothing on the q-register stack.

?PDO, Push-down list overflow

**Explanation:** The command string has become too complex. Simplify it.

?POP, Attempt to move Pointer Off Page with "x"

**Explanation:** A J, C or R command has been executed which attempted to move the pointer off the page. The result of executing one of these commands must leave the pointer between 0 and Z, inclusive. The characters referenced by a D or m,nX command must also be within the buffer boundary.

## Error Messages

?SNI, ; not in iteration

**Explanation:** A ; command has been executed outside of an open iteration bracket. This command may only be executed within iteration brackets.

?SRH, Search failure "text"

**Explanation:** A search command not preceded by a colon modifier and not within an iteration has failed to find the specified "text". After an S search fails the pointer is left at the beginning of the buffer. After an N or \_ search fails the last page of the input file has been input and, in the case of N, output, and the buffer is cleared. In the case of an N search it is usually necessary to close the output file and reopen it for continued editing.

?STL, String too long

**Explanation:** A search or file name string is too long. This is most likely the result of a missing <DELIM> after the string.

?TAG, Missing Tag !tag!

**Explanation:** The tag !tag! specified by an O command cannot be found. This tag must be in the same macro level as the O command referencing it.

?UTC, Unterminated command "x"

**Explanation:** This is a general error which is usually caused by an unterminated insert, search, or filespec argument, an unterminated ^A message, an unterminated tag or comment (i.e., unterminated ! construct), or a missing ' character which closes a conditional execution command.

?UTM, Unterminated macro

**Explanation:** This error is the same as the ?UTC error except that the unterminated command was executing from a Q-register (i.e., it was a macro). (Note: An entire command sequence stored in a Q-register must be complete within the Q-register.)

?XAB, Execution aborted

**Explanation:** Execution of TECO was aborted. This is usually due to the typing of <CTRL/C>.

?WLO, System Device Write-Locked

**Explanation:** TECO-8 needs to write on the system device when it is running in less than 16K (less than 20K if VT52 is present) so that it can later swap in overlays.

?YCA, Y command aborted

**Explanation:** An attempt has been made to execute an Y or \_ search command with an output file open, that would cause text in the text buffer to be erased without outputting it to the output file. The ED command (Section 5.16) controls this check.

?nnn, I/O Error or Directive Error (RSX-11 only)

**Explanation:** All errors from the executive and file system are reported in this format, where nnn is the decimal I/O or directive error status. The accompanying message is the corresponding message from the QIOSYM message file. A complete list of I/O and directive errors appears in appendices to the various Executive reference manuals and in the IAS/RSX-11 I/O Operations Reference Manual.





## Incompatible, Obsolete, and System-Specific Commands

This appendix describes commands that are peculiar to specific operating systems. These commands fall into many categories. Some are obsolete, and are kept around only as a convenience to the user. Others are so system specific or so obscure that it was felt best not to include them in the main body of the manual. Some are incompatible across operating systems. Some are new commands that have not become firmly established and may change in the future. In general, use these commands at your own risk and with the understanding that in future releases of TECO, these commands may change or go away completely. Implementors of TECO on other operating systems should contact the TECO SIG before implementing any of these features.

### C.1 Specific Features of TECO-11

Table C–1 TECO-11 and TECO-32 Commands

Command	Description
<code>m,nStext`</code>	Performs the same function as the <code>nS</code> command, but <code>m</code> serves a bound limit for the search. If the search string can be found without moving the pointer more than <code>ABS(m)-1</code> places, the search succeeds and the pointer is repositioned to immediately after the last character of the string. Otherwise, the pointer is left unchanged. The <code>^Q</code> operator, described below, is useful in conjunction with this command.  Note that <code>m,Stext`</code> is identical to <code>m,1Stext`</code> and <code>m,-Stext`</code> is identical to <code>m,-1Stext`</code> .
<code>m,-nStext`</code>	Performs the same function as the <code>m,nS</code> command, but searches in the reverse direction.
<code>0,nStext`</code>	Performs the same function as the <code>nS</code> command, except that the pointer position will remain unchanged on search string failure. (Essentially an unbounded search with no pointer movement on failure.)
<code>G*</code>	Get most recent filespec string. The asterisk represents TECO's filespec string area, which contains the fully expanded filespec of the last <code>E</code> command (see appendices). Copy the contents of the filespec string area into the buffer at the current position of the buffer pointer, leaving the pointer positioned after the last character copied.
<code>:G*</code>	Print the contents of the filespec buffer on the terminal.

(continued on next page)

## Incompatible, Obsolete, and System-Specific Commands

### C.1 Specific Features of TECO-11

Table C–1 (Cont.) TECO-11 and TECO-32 Commands

Command	Description																
G_	Get most recent search string. The underscore (backarrow) represents TECO's search string area. Copy the contents of the search string area into the buffer at the current position of the buffer pointer, leaving the pointer positioned after the last character copied.																
:G_	Print the contents of the search string buffer on the terminal.																
n^Q	n^QC is identical to nL. The n^Q command returns the number of characters between the buffer pointer and the nth line separator (both positive and negative). This command converts line oriented command argument values into character oriented argument values. Used after an expression.																
m,-256+n:W	<p>Inserts characters at "dot" until... Characters are read (echo off) from the terminal and inserted at "dot" until and according to the microcoded bits in n. The terminating character is not inserted.</p> <table> <tr> <th>bit</th><th>meaning</th></tr> <tr> <td>128</td><td>Return immediately if no typed characters</td></tr> <tr> <td>64</td><td>Terminate on any character</td></tr> <tr> <td>32</td><td>Don't keep screen updated (i.e., no -1W)</td></tr> <tr> <td>8</td><td>Treat m as terminating character(s)</td></tr> <tr> <td>4</td><td>Convert any alphabetic inserts to upper case</td></tr> <tr> <td>2</td><td>Terminate on &lt;TAB&gt;</td></tr> <tr> <td>1</td><td>Screen is initially O.K.</td></tr> </table> <p>Control characters (octal 0 through 37 and 177 except &lt;TAB&gt;) are always terminating characters. The returned value has the terminating character code in low byte (octal 0 through 177 or 377 for returned immediately). The sign bit is set if one or more inserts were done. The return value may be a character code that normally would have been inserted, but could not be for some reason (e.g., convert alphabetic inserts to upper case requested, but not supported in this implementation).</p> <p>The m of m,-256+n:W is always optional. If bit value 8 of n is set, m contains up to two additional termination character codes, one in the low 8 bits (low byte) and another in the high 8 bits (high byte). If only one extra termination character is desired, it is simply placed in m thus setting the high 8 bits to zero (which is already a termination character). If bit value 8 of n is not set, passing the m argument is undefined...</p>	bit	meaning	128	Return immediately if no typed characters	64	Terminate on any character	32	Don't keep screen updated (i.e., no -1W)	8	Treat m as terminating character(s)	4	Convert any alphabetic inserts to upper case	2	Terminate on <TAB>	1	Screen is initially O.K.
bit	meaning																
128	Return immediately if no typed characters																
64	Terminate on any character																
32	Don't keep screen updated (i.e., no -1W)																
8	Treat m as terminating character(s)																
4	Convert any alphabetic inserts to upper case																
2	Terminate on <TAB>																
1	Screen is initially O.K.																

## Incompatible, Obsolete, and System-Specific Commands

### C.1 Specific Features of TECO-11

**Table C-2 String Build Constructs**

Construct	Description
<CTRL/E>Q*	<CTRL/E>Q* indicates that the string stored in the filespec buffer is to be used in the position occupied by the ^EQ* in the search string.
<CTRL/E>Q_	<CTRL/E>Q_ indicates that the string stored in the search string buffer is to be used in the position occupied by the ^EQ_ in the search string.

## C.2 Specific Features of RT-11

[For "RT-11 Operating Characteristics", refer to Appendix D.]

## C.3 Specific Features of RSTS/E

[For "RSTS/E Operating Characteristics", refer to Appendix E.]

**Table C-3 TECO/E Commands**

Command	Description
:EGRTS`	Switch to private default run-time-system.
:EGRTS foo`	Switch to RTS "foo".
:EGFSS string`	File string scan "string".
:EGCCL cmd`	Try "cmd" as a CCL command.
:EGRUN file`	Try to run "file".
:EGRUN file=xx`	Try to run "file" with "xx" placed in core common.
:EGEMT`	Issue a monitor directive. The FIRQB is loaded from Q-registers A through P and the XRB is loaded from Q-registers Q through W. The low byte of the value in Q-register A is the monitor EMT code to issue. If the high byte of the value in Q-register A is >0 then the text part of Q-register A is put into the XRB for a 'write' (XRLEN= size of A, XRBC=size of A, XRLOC->A); if Q-register A high byte is <0 then the text part of Q-register A is put into the XRB for a 'read' (XRLEN=size of A, XRBC=0, XRLOC->A). Returned value is -1 for success, 0 for unrecognized command, or >0 for the RSTS/E error code. The FIRQB is placed in the numeric part of Q-reg A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P. The XRB is placed in the numeric part of Q-regs Q, R, S, T, U, V, W.

See also Section 5.1.2, File Specification Switches.

## C.4 Specific Features of RSX-11/M, RSX-11/D, and IAS

[For "RSX-11 Operating Characteristics", refer to Appendix F.]

## Incompatible, Obsolete, and System-Specific Commands

### C.4 Specific Features of RSX-11/M, RSX-11/D, and IAS

**Table C–4 TECO/RSX Commands**

Command	Description
<CTRL/Z><CTRL/Z> <CTRL/Z>	Returns control to the operating system immediately. Equivalent to typing <CTRL/C><CTRL/C>.

See also Section 5.1.2, File Specification Switches.

### C.5 Specific Features of VAX/VMS

[For "VAX/VMS Operating Characteristics", refer to Appendix G.]

**Table C–5 TECO/VMS Commands**

Command	Description
<CTRL/Z><CTRL/Z> <CTRL/Z>	Returns control to VAX/VMS immediately. Equivalent to typing <CTRL/C><CTRL/C>.
EG`	Exit with special status TECO's normal exit status code (as reflected by the DCL symbols \$STATUS and \$SEVERITY) is 1 (success). If the exit was due to an EG` command, the exit status code is 9 (also success).
:EGSYM symbol_name symbol_value`	Defines the DCL local symbol "symbol_name" with a value string of "symbol_value". Useful for creating symbols that will be used by a command procedure upon TECO's exit.

See also section 5.1.2, File Specification Switches.

### C.6 Specific Features of OS/8

[For "OS/8 Operating Characteristics", refer to Appendix H.]

**Table C–6 TECO-8 Commands**

Command	Description
<CTRL/S>	If used as the first keystroke after TECO's prompt, this command is the same as *Z, that is, it saves the last command string in Q-register Z. This command echoes as *Z and does not have to be followed by a <DELIM>.

See also Section 5.1.2, File Specification Switches.

### C.7 Specific Features of TOPS-10

[For "TOPS-10 Operating Characteristics", refer to Appendix I.]

## Incompatible, Obsolete, and System-Specific Commands

### C.7 Specific Features of TOPS-10

**Table C-7 TECO-10 Commands**

Command	Description																				
nEM	Positions a magtape. You must open the magtape for input with an appropriate ER command before it can be positioned. Exact positioning depends on the value of n, as described below. You must reopen the magtape for output before you can output to it. [TECO-10 only]																				
	<table> <tr> <th>Value</th><th>Action</th></tr> <tr> <td>1EM</td><td>Rewind the currently-selected input magtape to load point. EM is identical to 1EM.</td></tr> <tr> <td>3EM</td><td>Write an end-of-file record.</td></tr> <tr> <td>6EM</td><td>Skip ahead one record.</td></tr> <tr> <td>7EM</td><td>Back up one record.</td></tr> <tr> <td>8EM</td><td>Skip ahead to logical end of tape (defined by two successive end-of-file marks). Leave the magtape positioned between the two end-of-file marks so that successive output correctly overwrites the second EOF.</td></tr> <tr> <td>9EM</td><td>Rewind and unload.</td></tr> <tr> <td>11EM</td><td>Write 3 inches of blank tape.</td></tr> <tr> <td>14EM</td><td>Advance tape one file. This leaves the tape positioned so that the next item read will be the first record of the next file (or the second end-of-file mark at the logical end-of-tape).</td></tr> <tr> <td>15EM</td><td>Backspace tape one file. This leaves the tape positioned so that the next item read will be the end-of-file mark preceding the file backspaced over (unless the file is the first file on the tape).</td></tr> </table>	Value	Action	1EM	Rewind the currently-selected input magtape to load point. EM is identical to 1EM.	3EM	Write an end-of-file record.	6EM	Skip ahead one record.	7EM	Back up one record.	8EM	Skip ahead to logical end of tape (defined by two successive end-of-file marks). Leave the magtape positioned between the two end-of-file marks so that successive output correctly overwrites the second EOF.	9EM	Rewind and unload.	11EM	Write 3 inches of blank tape.	14EM	Advance tape one file. This leaves the tape positioned so that the next item read will be the first record of the next file (or the second end-of-file mark at the logical end-of-tape).	15EM	Backspace tape one file. This leaves the tape positioned so that the next item read will be the end-of-file mark preceding the file backspaced over (unless the file is the first file on the tape).
Value	Action																				
1EM	Rewind the currently-selected input magtape to load point. EM is identical to 1EM.																				
3EM	Write an end-of-file record.																				
6EM	Skip ahead one record.																				
7EM	Back up one record.																				
8EM	Skip ahead to logical end of tape (defined by two successive end-of-file marks). Leave the magtape positioned between the two end-of-file marks so that successive output correctly overwrites the second EOF.																				
9EM	Rewind and unload.																				
11EM	Write 3 inches of blank tape.																				
14EM	Advance tape one file. This leaves the tape positioned so that the next item read will be the first record of the next file (or the second end-of-file mark at the logical end-of-tape).																				
15EM	Backspace tape one file. This leaves the tape positioned so that the next item read will be the end-of-file mark preceding the file backspaced over (unless the file is the first file on the tape).																				
n^Q	n^QC is identical to nL. This command returns the number of characters between the buffer pointer and the nth line separator (both positive and negative). This command converts line oriented command argument values into character oriented argument values. Used after an expression.																				
F0	Pointer position of start of window. Same as 6:W.																				
FZ	Pointer position of end of window.																				
FH	Same as F0,FZ.																				
FX	Software maintained horizontal coordinate of location of cursor on screen (0-origin). Updated by TECO on terminal output. If TECO cannot determine the updated location of the cursor, this value is set to -1.																				
nFX	Set value of FX register to n.																				
FY	Software maintained vertical coordinate of location of cursor on screen (0-origin). Updated by TECO on terminal output. If TECO cannot determine the updated location of the cursor, this value is set to -1.																				
nFY	Set value of FY register to n.																				
FP	Equivalent to FY,FX.																				

(continued on next page)

## Incompatible, Obsolete, and System-Specific Commands

### C.7 Specific Features of TOPS-10

**Table C–7 (Cont.) TECO-10 Commands**

Command	Description
m,nFP	Equivalent to mFYnFX.
m,nFF	Updates FY and FX registers as if the m,nT command were executed.
nFF	Assumes that the scope cursor is pointing at the character that is just to the right of the text buffer pointer (the current character). Then this command advances n screen lines and goes to the end of that line and returns that pointer position.
nFQq	Compares characters beginning at dot with characters beginning at the nth character in Q-register q (0-origin). When a match fails or the match ends, this command moves the pointer to after the last character that matched and returns the index into the Q-register.
E=filespec`	Renames the input file.
E&filespec`	Run the specified program when TECO exits. Can take a numeric argument which is the run-offset. 0 is the default.
m,nE*	Does an arbitrary TRMOP to your terminal. [Not available on TOPS-20.]
::Gq	Same as :Gq but types literally.
::ER	Same as ER but no defaults are used.
^Z	Closes output file and exits from TECO.
^P	Returns current page number.
n^P	Executes P commands until page n has been reached.
n^Y	Executes Y commands until page n has been reached.
n,m=	Same as m=n^T.
m,n:^T	Does an arbitrary TTCALL.

See also Section 5.1.2, File Specification Switches.

---

## RT-11 Operating Characteristics

### D.1 Startup

TECO is started with the

```
.R TECO
```

command. TECO is now immediately ready to accept commands. The text buffer and Q-register areas are empty.

The EDIT command:

```
.EDIT/TECO filespec
```

is used to edit an already existing file. It is equivalent to

```
.R TECO
*EBfilespec`Y`
```

For those RT-11 users that will use TECO as the primary editor, a monitor SET command is provided:

```
.SET EDITOR TECO
```

Once this command is issued, the /TECO option on the EDIT command is no longer necessary since the default editor is now TECO. Since this SET command only has affect between system bootstraps, it is recommended that the command be placed in the appropriate startup file (e.g., STARTS.COM).

Now, assuming the SET command has been issued, the command

```
.EDIT filespec
```

can be used to edit an already existing file.

The standard RT-11 EDIT command options are all available with TECO.

```
.EDIT/CREATE filespec
.EDIT/INSPECT filespec
.EDIT/OUTPUT:filespec filespec
```

Another option, /EXECUTE, is also available:

```
.EDIT/EXECUTE[:string] filespec
```

The /EXECUTE option causes TECO to process the filespec (assumed .TEC filetype) as a set of TECO commands. If "string" is used, the string is placed into TECO's text buffer. If "string" contains only alphanumeric characters, it does not have to be enclosed in quotes. If it is to contain blanks, it must be quoted with single quotes. The equivalent TECO commands would be

```
.R TECO
```

## RT-11 Operating Characteristics

### D.1 Startup

\*ERfilespec`YHXZHKIstring`MZ``

Note the input file remains open and can provide more input to the macro.

### D.2 File Specification

The file access commands ER, EB, EI, and EW accept a file specification in the standard RT-11 format:

dev:filename.type

in which dev: is a physical device name or a user assigned logical name; if dev: is not specified, the default DK: is assumed. The filename field must be specified in the commands ER, EB, EI, and EW and be a legal RT-11 filename. The type field is a file extension and must be explicitly given if used (there is no default). The EB and EW commands also accept the extended notation for an output file size

dev:filename.type[n]

The optional [n] specifies the output file size where n is the number of blocks to be allocated.

### D.3 Backup Files

The EB command maintains one level of file backup on RT-11. The pre-edited input file name is changed to:

filename.BAK

before the new output file is closed with the original name. Only normal file closing commands (EC, EF, EG, and EX) cause this renaming to happen. If TECO is aborted or the output file is purged by the EK command, the input filename remains unchanged. Note only one .BAK file for a given name is kept; earlier .BAK backup files are deleted each time a new backup file is created.

A good policy to follow when editing is to close the edited file frequently enough so that an unexpected incident would not cause a substantial loss of work. Files should be backed up regularly. TECO has the power to let an unsuspecting user alter a good file into a completely useless state. The SRCCOM program can be used to verify an editing session.

### D.4 EXIT and GO

If TECO is exited via the EGstring` command, the string is passed to the system as the next command to execute. This string may be any valid command or an indirect command file specification.

### D.5 REENTER and CLOSE

The RT-11 REENTER command may always be used to continue TECO. Its primary differences from running TECO is that when REENTER is used, the text buffer and Q-register areas are unmodified, as opposed to when TECO is run the text buffer and Q-register areas are cleared. The input and output file are always lost upon reentering TECO. If an output file was open before reentering TECO, the file will have to be recreated with the appropriate E-command. (Note that the monitor commands GT ON, GT OFF, LOAD, and UNLOAD disallow a REENTER.)



The output file is not closed if TECO is aborted. The RT-11 CLOSE command can be used to make the output file permanent, but be aware that the output file will not be complete because of internal buffers that TECO keeps. TECO may be reentered after a CLOSE command.

## D.6 File Recovery

TECO can be a useful tool in recovering ASCII files lost on a block replaceable device. TECO allows block replaceable devices to be opened in a non-file structured mode. This gives the user the capability to open a disk and access ASCII data anywhere on it, independent of file boundaries. The command

ERdev:`

is used to open the device at which point \_ (underscore or backarrow) searches may be used to locate specific ASCII data and transfer it to new output files. Note that files tend to get reproduced, in whole or part, many places on a block replaceable device; be sure to verify that any given text is indeed complete and the correct version.

## D.7 System Crash Recovery

TECO and RT-11 are highly reliable, but if during an important edit session a random system failure should occur, the following procedure may help save some or all of the editing.

1. Bootstrap the system
2. Immediately perform a SAVE command to save as much of memory as possible into a file on SY:. The address range form of the SAVE command must be used. The SAVE command will not allow any part of the monitor to be saved, e.g., if you have a 28K system and are running SJ you cannot save 28K but only 26.3K.
3. Perform standard startup procedures, e.g., DATE.
4. Use TECO on the SAVEd file to try and recover useful parts of the edit.

## D.8 VT11 Graphics Support

If the monitor supports the VT11 graphics processor (GT ON and GT OFF work) TECO will automatically start up in display mode, adjusting to both the size of the display screen and to the presence or absence of the scroller.

If the display fails to start with a working VT11, TECO has decided that there is not enough free memory and will not allocate the display file buffer or start the display.

See Section 5.17 for a description of the available commands to interact with the display.

Various aspects of the display screen become immediately obvious upon seeing them; the text pointer, its position and shape and its position between lines; wrap around of more than 72 characters per line; the scroller interaction and so on. Experiment with a scratch file for more familiarity.



---

## RSTS/E Operating Characteristics

### E.1 Startup

RSTS supports all of the standard TECO invocation commands, namely:

```
TECO
TECO filespec
TECO filespec=filespec
MAKE filespec
MUNG filespec
MUNG filespec,text
```

The CCL command switches /DETACH and /SIZE:n (or /SIZE:+n) can be used with TECO. If /DETACH is used and the user's detached-job quota has not yet been reached, TECO will detach the job before any further processing. If /SIZE:n is used, TECO will pre-expand the text and Q-register storage area to nK. If /SIZE:+n is used, TECO will set the text storage and Q-register storage area to n+4K initially (TECO's default startup size is 4K).

### E.2 File Specification

The file access commands ER, EB, EW, and EI accept a file specification in the standard RSTS/E format:

```
dev:[p,pn]filename.ext
```

in which dev: is a physical device name or a logical device name; if dev: is not specified, the public structure is assumed. If [p,pn] is not specified, the user's current logged in account is assumed. The filename field must be specified whenever the device name references a file structured device. The .ext field is a file extension and must be explicitly given if used. There is no default extension except for EI commands which default the .ext field to .TEC.

The file specification switches /n, /B+, /, and /B2 may be applied to the ER, EW, and EB commands. These switches invoke special handling of program lines containing <LF> or & continuation characters. See Appendix J for more information.

The file specification switch /B causes TECO to handle the file as a stream of 8-bit binary characters; normal ASCII format processing (such as removal of null characters) is not done.

The file specification switches /RONLY, /MODE:n, and /CLUSTERSIZE:n can be included in a file specification. TECO automatically opens any disk input file in /RONLY mode. The file size switches /FILESIZE:n and /SIZE:n might leave an output file larger than the amount of data output by TECO. These file size switches are therefore illegal and produce an error if included in a file specification.

## RSTS/E Operating Characteristics

### E.2 File Specification

The EB and EW commands also accept the extended notation for an output file protection code

dev:[p,pn]filename.ext<prot>

The optional <prot> specifies the output file protection code.

### E.3 Backup Files

The EB command maintains one level of file backup on RSTS/E. The pre-edited input file name is changed to

filename.BAK

before the new output file is closed with the original name. Only normal file closing commands (EC, EF, EG, and EX) cause this renaming to happen. If TECO is aborted or the output file is purged by the EK command, the input filename remains unchanged. Note only one .BAK file for a given name is kept; earlier .BAK backup files are deleted each time a new backup file is created.

A good policy to follow when editing is to close the edited file frequently enough so that an unexpected incident would not cause a substantial loss of work. Files should be backed up regularly. TECO has the power to let an unsuspecting user alter a good file into a completely useless state. The FILCOM program can be used to verify an editing session.

### E.4 EXIT and GO

If TECO is exited via the EGstring` command, the "string" is executed as a RSTS/E CCL command after the input and output file(s) are closed.

### E.5 ET Flag Handling

Bit 6 (detach) is handled specially by TECO. Every time the ET flag is read (used as a numeric value), TECO ensures that Bit 6 is on if the job is attached or off if the job is detached. This allows a TECO macro to check for "detachedness". If an attempt is made to set Bit 6 and the user's detached-job quota has been reached, the request is ignored and Bit 6 will read back as a 0 (assuming the job is attached). If Bit 6 is set and the user's detached-job quota has not yet been reached, the job will become detached. Further reading of Bit 6 will return a 1, indicating the detached condition.

---

## RSX-11 Operating Characteristics

### F.1 Startup

RSX-11 systems support all of the standard TECO invocation commands, namely:

```
TECO
TECO filespec
TECO filespec=filespec
MAKE filespec
MUNG filespec
MUNG filespec,text
```

If any of these commands are not recognized by the system, check with your system manager to see that TECO is properly installed.

TECO macros may also be invoked with the command:

```
TECO @filespec
```

It is exactly equivalent to

```
MUNG filespec
```

In systems supporting dynamic task expansion, TECO will expand its buffer space as necessary. Also, TECO'S buffer space may be explicitly allocated in the startup command

```
RUN $TEC/INC=n
```

### F.2 Initialization

TECO searches for the TECO.INI startup file in the current default device and directory. TECO's memory, in which a plain TECO command edits the file last edited with a TECO filespec or a MAKE filespec command, is implemented with a file named TECF00.TMP, also stored in the current default device and directory.

The initial value of the ED flag is always 1.

When TECO is initially invoked it will automatically set the ET and EU flags according to the user's terminal characteristics. If the terminal supports CRT style rubouts, then bit 1 of the ET flag is set to do the same in TECO. If the terminal supports lower case type in, then bit 2 of the ET flag is set and the EU flag is set to -1 to turn off case flagging. If the terminal is a CRT type terminal and the version of TECO includes the screen support package, then bit 9 of the ET flag is set. While the command line is being processed, bit 7 of the ET flag is also set to cause TECO to exit should any errors occur. ET bit 7 is cleared every time TECO reaches prompt (\*) level.

### **F.3 File Specification**

The file access commands ER, EB, EW, and EI accept a file specification in the standard RSX-11 format:

`dev:[p,pn]filename.typ;version`

in which dev: is a physical device name or a logical device name; if dev: is not specified, SY: is assumed. If [p,pn] is not specified, the user's current default directory is assumed. The filename field must be specified whenever the device name references a file structured device. The typ field is a file type and must be explicitly given if used. There is no default type except for EI commands which default the .typ field to .TEC.

The /B2 file specification switch may be applied to the ER, EW, and EB commands and invokes special handling of program lines containing & continuation characters. See Appendix J for more information.

The switch /RW may be applied to any file specification in an ER, EW, and EI command. If the file specification references a magtape, the tape is rewound before the file is opened. Note that for output files, this has the effect of zeroing the tape. The /RW switch is ignored for all other device types.

The presence of version numbers in Files-11 causes file processing to behave slightly differently under RSX-11 than under other operating systems. For example, no .BAK files are used; each execution of an EB command simply produces a new version of the file. Thus a user may retain any level of backup he feels to be comfortable. It also means that one must occasionally delete obsolete files to avoid cluttering the disk. Thus the command

`EBname.typ;version``

is equivalent to the commands

`ERname.typ;version`EWname.typ;0``

The EW command also creates a new version (one higher than the current highest) if no version number is given. If an explicit version number is given, then that number is used, and if another file of the same name, type, and version previously existed, it is superseded without warning. (See use of the EP and EK commands below.)

In reading files, version numbers behave the same as in other RSX-11 utilities: the default is the highest version. This leads to a problem in re-opening the input file while a file is being edited with EB. Since the output file is already created and in the directory, the input file is no longer the highest version. One may deduce the version number of the input file by doing a :G\* (typing the file string of the output file) and subtracting one from that version number.

In symmetry with the EB command, the EK command functions by simply deleting the current output file. Note, however, that a supersede (EW of same name, type, and version) is not undone - the file is already deleted!

The EP and EA commands, while simulating two channels each with an open file for each of input and output, in fact only keep one file open for each to conserve buffer space. This means that they are only useful for disk files. Also, it means that if you open a file and then supersede it, you should not switch the input channel away from it with an EP or ER` command, since it will not be possible to open the file again.

## F.4 Wild Card Lookup

The EN command will process wild card lookups on RSX-11. To preset the wild card lookup file specification, use the standard RSX-11 format

dev:[p,pn]filename.typ;version

The device name must reference a file structured disk device or magtape. All other fields of the file specification may be fully wild (\*), including either or both halves of the directory. The version number may be explicit, wild, or default. As with the other file specification commands, there is no default file type.

## F.5 Exiting From TECO

The normal method of exiting from TECO is with the EX command. This copies the remaining input file to the output file, closes all files and exits.

The <CTRL/C> (or Caret-C) command is the "give up and get out" command. Executed from main command level, it will cause TECO to exit regardless of the state of the buffer. If there is an open output file, it is deleted. The <CTRL/C> command is roughly equivalent to EKHKEX.

## F.6 <CTRL/C>

The action taken when the user types <CTRL/C> depends on what TECO is doing.

If TECO is executing commands, or is awaiting type-in for the ^T command, the ?XAB error occurs.

If TECO is at command level, typing <CTRL/C> cancels the command string currently being typed and returns TECO to its prompt. Two consecutive <CTRL/C> characters will cause an instant HKEKEX exit.

Sometimes it is desirable for a TECO macro to detect when a <CTRL/C> was typed. By detecting the <CTRL/C>, the macro can exit cleanly back to command level (pop saved Q-registers, restore any flag values, etc.). To do this, the macro sets Bit 15 (Octal 100000, Decimal -32768) of the ET flag. When a <CTRL/C> is typed, TECO will automatically turn off Bit 15, but will continue execution of the macro. The macro periodically checks Bit 15 and exits cleanly if it ever goes off. For example:

```
[0 [1 -32768#ETET < ... ET; > 32767&ETET ]1 ]0
```

Setting the <CTRL/C> intercept bit in the ET flag must be done with some care; if the bit is set inside a command loop which does not check it, it will be impossible for the user to abort the loop. The only remedy for this situation is to abort TECO from another terminal.

## F.7 EXIT and GO

If TECO is exited via the EGstring` command, TECO closes its files and exits. It then causes "string" to be executed as an MCR command using the spawn system directive. This feature works only on RSX-11M V3.2 and RSX-11M+ V1 or later.

## **F.8 ET Flag Handling**

TECO will automatically turn off the following bits in the ET flag on every error: Bit 0 (image output), Bit 3 (no echo on ^T), Bit 4 (cancel ^O), Bit 5 (no stall on ^T), and Bit 15 (^C trap).

In addition, TECO always turns off Bit 7 (exit on error, etc.) every time it reaches prompt (\*) level.

Bit 6 (the detach flag) controls TECO'S treatment of the terminal. Normally, TECO keeps the terminal attached to gain control of <CTRL/C> interrupts. Setting bit 6 of the ET flag causes TECO to run with the terminal detached. All commands function normally, except that typing <CTRL/C> causes the MCR to be activated, allowing other tasks to be run from the same terminal concurrently with TECO. It is, of course, the user's problem to sort out the confusion that will arise if both TECO and another task request input from the terminal at the same time.

## **F.9 File Record Format**

Files-11 files are record structured, while TECO's text buffer is ASCII stream. Thus TECO must make format conversions when reading and writing files. The conversion depends on the record attributes of the file. While reading a file, the records are packed into the buffer. If the file is implied carriage control (the standard RSX-11 source format) or Fortran carriage control, TECO inserts a carriage return and line feed after each record to make each record appear as a line of text in the buffer, unless the record ends with ESCAPE, carriage return, line feed, vertical tab, or form feed. A record ending in form feed is interpreted as an end of page mark; it stops the read operation and the form feed is not entered in the buffer. If the file has print file carriage control, TECO interprets the carriage control bytes and inserts the resulting carriage return and line feed characters about the record. If the input file has no carriage control (also called internal carriage control), TECO simply packs the records together in the text buffer.

On output, TECO scans the text buffer for carriage return, line feed, vertical tab, and form feed characters. Each such character delimits the end of an output record. If the output file is implied or Fortran carriage control, and the record ends with exactly carriage return / line feed, the carriage return and line feed are not output with the record; Otherwise, the record is output in its entirety. The carriage return and line feed are also output with the record if the record ends with ESCAPE / carriage return / line feed.

Switches may be applied to the input and output files to control their carriage control attributes. The switch /CR forces implied carriage control; /-CR forces no (internal) carriage control; /FT forces Fortran carriage control. When a carriage control switch is applied to an input file, the file is read as if it had that attribute; when the switch is applied to an output file, the file is written with that attribute. Applying a switch to an EB file specification causes the switch to apply to both input and output files. When an output file is created, its carriage control attributes are defaulted to those of the currently open input file as follows:



Input	Output
implied	implied
none	implied
FORTTRAN	FORTTRAN
print file	implied

Files read with the EI command have their record attributes interpreted in the same manner. This leads to an unexpected side effect with EI files containing an entire command. The last record of the file presumably contains as its last characters the two alt modes which initiate execution of the macro. If the file is implied carriage control, however, there are also the final carriage return / line feed belonging to the last record, which remain in the type in buffer while the macro executes. If the macro attempts to receive input with the <CTRL/T> command, the carriage return / line feed will be the first two characters read. Alternatively, if the macro does no type in, the carriage return / line feed will be read by TECO as the first two characters of the next command. Then no asterisk (\*) will appear as the prompt for the next command. The remedy for both cases is for the macro to execute an EI` command early on. This causes the remainder of the indirect file to be discarded and further input to be read from the terminal.

## F.10 Command Line Processing

The mechanism used to process the command line in RSX-11 TECO is designed to allow sophisticated TECO users the greatest flexibility in customizing TECO for their own use. It functions as follows:

The initialization routine places the original MCR command line (if any) into the filename buffer. It copies into the text buffer the text of a TECO macro that will be used to interpret the command line. Then it starts up TECO with the command:

HXY HKG\* HXZ HK :EITECO``

in the type in buffer. This loads the command line into Q-register Z and the macro into Q-register Y. It then executes the file named TECO.TEC located in the user's default directory, if it exists. After the user's TECO.TEC, and any files it might link to with EI, have been executed, TECO executes the command MY``, thus executing the macro to interpret the command line and open the files requested.

The TECO.TEC mechanism should not be used for simple initialization; the standard TECO.INI facility should suffice for that. The alternate TECO.TEC facility is provided for the sophisticated user who wants his own command processing and thus wishes to usurp control from the normal initialization.

If an EI` command (to close the indirect command file) is executed during the processing of a user's TECO.TEC startup file, the final MY`` which causes processing of the command line is not executed. This results from the fact that the MY`` normally appears in TECO's type-in after all command files have been processed. Executing the EI` command causes all "type ahead" to be discarded to allow a TECO command file to prompt and read input from the terminal (and not read extraneous type ahead). It is assumed that a TECO startup file that executes EI` and reads input from the terminal will want to manage the rest of TECO's startup. If it still wants to process the command line, it must issue the MY itself.



---

## VAX/VMS Operating Characteristics

### G.1 Startup

VAX/VMS supports all of the standard TECO invocation commands, namely:

```
TECO
TECO filespec
TECO filespec=filespec
MAKE filespec
MUNG filespec
MUNG filespec,text
```

If any of these commands are not recognized by the system, see the installation instructions (Section G.16) in this appendix.

VMS Versions 5.0 and later recognize the EDIT/TECO command. See the VMS Help for supported qualifiers.

### G.2 Initialization

TECO performs initialization by attempting to translate the logical name TEC\$INIT. If this name does not translate, no special initialization is done. If it translates to a string of the form "\$filespec" (where "\$" is a dollar sign), TECO executes the TECO commands in the specified filespec during initialization. If TEC\$INIT translates to any other string, TECO executes that string as TECO commands during initialization. The original command line is available in the text buffer when the user initialization commands execute. Generally, you would set user private modes and/or flags at this time. This is also the normal place to detect, strip off, and do something appropriate with user private qualifiers.

TECO's memory is controlled by the logical name TEC\$MEMORY. If this name translates to a string of the form "\$filespec", TECO uses the specified file for its memory. Otherwise, TECO uses the logical name itself as the memory.

If TECO is requested to load VTEDIT at startup (e.g., via a TECO /VTEDIT command), it attempts to translate the logical name TEC\$VTEDIT. If the name is defined, the resulting filespec is used as the file from which to load the scope editor. If TEC\$VTEDIT is not defined, TECO attempts to translate the logical name TEC\$LIBRARY. If that logical is defined, it is used as the (device and) directory name of where to find VTEDIT.TEC. Elsewise, TECO defaults to SYSS\$LIBRARY:VTEDIT.TEC.

These four logical names (TEC\$INIT, TEC\$MEMORY, TEC\$VTEDIT, and TEC\$LIBRARY) are the information holders corresponding to the four standard :EG commands INI, MEM, VTE, and LIB respectively (see Section 5.1.3).

## G.3 File Specification

All file specifiers are subject to the VAX/VMS file specifier rules; logical names and multi-level directories are properly handled, including transparent network file access. The filename field must be specified whenever the device name references a file structured device. The type field must be explicitly given if used. There is no default type except for EI commands which default the type field to .TEC.

Qualifiers may be applied to file specifications to control format and attributes; see Section G.13.

The qualifier /RW may be applied to any file specification in an ER, EW, and EI command. If the file specification references a magtape, the tape is rewound before the file is opened. Note that for output files, this has the effect of zeroing the tape. The /RW qualifier is ignored for all other device types.

The presence of version numbers in Files-11 causes file processing to behave slightly differently under VAX/VMS than under other operating systems. For example, no .BAK files are used; each execution of an EB command simply produces a new version of the file. Thus a user may retain any level of backup he feels to be comfortable. It also means that one must occasionally delete obsolete files to avoid cluttering the disk. Thus the command

`EBname.typ;version``

is equivalent to the commands

`ERname.typ;version`EWname.typ;0``

The EW command also creates a new version (one higher than the current highest) if no version number is given. If an explicit version number is given, then that number is used, and if another file of the same name, type, and version previously existed, it is superseded without warning. (See use of the EK command below.)

In reading files, version numbers behave the same as in other VAX/VMS utilities: the default is the highest version. This leads to a problem in re-opening the input file while a file is being edited with EB. Since the output file is already created and in the directory, the input file is no longer the highest version. One may deduce the version number of the input file by doing a `ER`G*`` which types the file string of the current (primary) input file.

---

### Note

---

On VMS systems you can set the version limit of files and directories to a maximum value, so that VMS will automatically delete the oldest versions when more files than the version limit are present.

When you use the EB command to open a file (either explicitly or implicitly) TECO will open the input file and the output file. If the version limit for that file is 1, VMS will automatically mark the input file for deletion upon closure of that file. So make sure that you do not terminate the edit in any way without writing the contents of the buffers and the remainder of the input file to the output file, or your input file will be lost!

---

In symmetry with the EB command, the EK command functions by simply deleting the current output file. Note, however, that a supersede (EW of same name, type, and version) is not undone - the file is already deleted!

When files are processed concurrently on the primary and secondary channels, all files are kept open. Thus the problems that occur under RSX-11 do not exist under VAX/VMS.

## G.4 Wildcard Lookup

Wild card file name processing supports all of the wild carding facilities of RMS. For VAX/VMS V2 and later systems this includes embedded \* and % and multi-level directory wildcarding.

## G.5 Symbol Constituents

The match control character ^EC and the conditional n"C accept the VAX/VMS symbol constituent character set, which consists of upper and lower case alphabets, numerics, ., \$, and \_.

## G.6 Exiting From TECO

The normal method of exiting from TECO is with the EX command. This copies the remaining input file to the output file, closes all files and exits.

The <CTRL/C> (or Caret-C) command is the "give up and get out" command. Executed from main command level, it will cause TECO to exit regardless of the state of the buffer. If there is an open output file, it is deleted. The <CTRL/C> command is roughly equivalent to EKHKEX.

TECO's normal exit status code (as reflected by the DCL symbols \$STATUS and \$SEVERITY) is 1 (success). An exit caused by an EG` command sets the exit status code to 9 (also success). An error exit caused by an error or <CTRL/C> with the 128 bit set in ET sets the exit status code to hexadecimal 1000002C which is SSS\_ABORT with the inhibit message bit set. All other exits are from fatal internal errors and the exit status code is the fatal error status code.

## G.7 <CTRL/C>

The action taken when the user types <CTRL/C> depends on what TECO is doing.

If TECO is executing commands, or is awaiting type-in for the ^T command, the ?XAB error occurs.

If TECO is at command level, typing <CTRL/C> cancels the command string currently being typed and returns TECO to its prompt. Two consecutive <CTRL/C> characters will cause an instant HKEKEX exit.

Sometimes it is desirable for a TECO macro to detect when a <CTRL/C> was typed. By detecting the <CTRL/C>, the macro can exit cleanly back to command level (pop saved Q-registers, restore any flag values, etc.). To do this, the macro sets Bit 15 (Hex 8000, Octal 100000, Decimal -32768) of the ET flag. When a <CTRL/C> is typed, TECO will automatically turn off Bit 15, but will continue execution of the macro. The macro periodically checks Bit 15 and exits cleanly if it ever goes off. For example:

## VAX/VMS Operating Characteristics

### G.7 <CTRL/C>

```
[0 [1 -32768#ETET < ... ET; > 32767&ETET ]1 ]0
```

Setting the <CTRL/C> intercept bit in the ET flag must be done with some care; if the bit is set inside a command loop which does not check it, it will be impossible for the user to abort the loop. The only remedy for this situation is to abort TECO with <CTRL/Y>, resulting in the loss of the edit.

### G.8 <CTRL/Y>

<CTRL/Y> is not handled at all by TECO and will result in trapping to the command interpreter. Should you accidentally type <CTRL/Y>, immediately type CONTINUE in response to the DCL prompt to resume editing.

### G.9 EXIT and GO

If TECO is exited with the EGstring` command, the string is passed to the command interpreter as the next command to execute after TECO has closed its files and exited.

EG without an argument executes the DCL command TECO\_EG, if a TECO\_EG symbol exists.

## G.10 Operating System Functions

The :EG command is used to perform operating system functions without exiting TECO (see Section 5.1.3).

The :EGLOG command can be used to manipulate logical names.  
For example:

Command	Function
:EGLOG FOO`	Loads the translation of logical name FOO into the filespec buffer. Returns -1 if the name exists, 0 if not
:EGLOG foo `	Deassigns logical name FOO
:EGLOG foo bar`	Defines logical name FOO

All logical name assignments are done in the process logical name table.

The :EGSYM command can be used to manipulate DCL symbols.  
For example:

Command	Function
:EGSYM FOO`	Loads the translation of symbol FOO into the filespec buffer. Returns -1 if the symbol exists, 0 if not
:EGSYM foo `	Deletes symbol FOO
:EGSYM foo bar`	Creates symbol FOO

To spawn a subprocess, use the :EGSPAWN command (SPAWN can be abbreviated to SPA). The format is:

```
:EGSPA[WN][/QUALIFIERS][COMMAND]
```

All of the qualifiers allowed on the DCL SPAWN command are allowed.  
For example:

```
:EGSPAWN/PROMPT="Foo> "/PROCESS="TECO_SUB"  
.. spawns an interactive subprocess  
:EGSPAWN DIR/OUT=TEMP.TMP *.BAR`  
.. spawns a command
```

The return value for :EGSPA is the subprocess exit status.

To attach to a detached process, use the :EGATTACH command.  
It is compatible with the DCL ATTACH command:

```
:EGATTACH process-name`  
.. attaches to the named process  
:EGATTACH/IDENTIFICATION=pid`  
.. attaches to the process with the given PID  
:EGATTACH`  
.. attaches to the parent process
```

## G.11 Flag Handling

TECO-32 does 32-bit math. Flags (such as the ET flag) are sign-extended from 16 to 32 bits when interrogated.

### G.11.1 The EE flag

The EE flag allows the setting of an alternate 'escape' character for terminals using the LK201 or similar keyboards.  
For example, 96EE sets accent grave ( ` ) as the alternate escape character. The EE flag is set to the ASCII value of the alternate delimiter. This character will echo as ` regardless of the actual value of the alternate delimiter.

Setting the 8192 bit in the ET flag sets EE to 96.

When read by ^T or in screen mode, the alternate escape character will return an escape (27).

### G.11.2 The EJ flag

The -1EJ flag (operating system) returns 1024.

TECO-11 returns 5. Code written for TECO-11 such as:

```
-EJ-5"E . . . '
```

can be changed to:

```
-1EJ-5*(-1EJ-1024)"E . . . '
```

and work on either TECO-32 or TECO-11.

## VAX/VMS Operating Characteristics

### G.11 Flag Handling

#### G.11.3 The ET flag

TECO will automatically turn off the following bits in the ET flag on every error: Bit 0 (image output), Bit 3 (no echo on ^T), Bit 4 (cancel ^O), Bit 5 (no stall on ^T), and Bit 15 (^C trap).

In addition, TECO always turns off Bit 7 (exit on error, etc.) every time it reaches prompt (\*) level.

Bit 6 (the detach flag) detaches your physical terminal from your process (similar to ^Y, DISCONNECT/CONTINUE).

The DETACH bit can be interrogated to check if the user has re-connected to the process. For example:

```
0,64ET !Disconnect! < . . . some work . . . ET&64"E 0; >
```

The iteration exits when the user re-connects.

#### G.11.4 The :W flag

TECO-32 uses scrolling regions and editing functions to screen updating on terminals with the proper functions.

This has been found to 'break' with some editing macros that modify the terminal characteristics. A new :W flag, 8:W, has been added. Setting 8:W nonzero disables TECO-32's use of scrolling regions.

Another new terminal flag is 9:W.

It returns a bit-encoded mask that describes the terminal characteristics.

Attempts to set 9:W are ignored.

The value returned has the following meanings:

Value	Meaning
1	Is ANSI CRT
2	Has EDIT mode features
4	Can do reverse scrolling
8	Has special graphics
16	Can do reverse video
32	Can change width
64	Has scrolling regions
128	Can erase to end-of-screen

### G.12 Q-registers

In TECO-32 all Q-registers and the text buffer can grow up to 65535 characters in size each, subject to virtual memory constraints.

### G.13 File Record Format

Files-11 files are record structured, while TECO'S text buffer is ASCII stream. Thus TECO must make format conversions when reading and writing files. The conversion depends on the record attributes of the file. While reading a file, the records are packed into the buffer. If the file is implied carriage control (the standard VAX/VMS source format) or FORTRAN carriage control, TECO inserts a carriage return and line feed after each record to make each record appear as a line of text in the buffer, unless the record ends with ESCAPE, carriage



return, line feed, vertical tab, or form feed. A record containing a form feed is interpreted as an end of page mark; it stops the read operation and the form feed is not entered in the buffer. The portion of the record after the form feed, if any, is saved for the next input command. If the file has print file carriage control, TECO interprets the carriage control bytes and inserts the resulting carriage return and line feed characters about the record. If the input file has no carriage control (also called internal carriage control), TECO simply packs the records together in the text buffer.

On output, TECO scans the text buffer for carriage return, line feed, vertical tab, and form feed characters. Each such character delimits the end of an output record. If the output file is implied or FORTRAN carriage control, and the record ends with exactly carriage return / line feed, the carriage return and line feed are not output with the record; Otherwise, the record is output in its entirety. The carriage return and line feed are also output with the record if the record ends with ESCAPE / carriage return / line feed.

Qualifiers may be applied to the input and output file specifications. When a carriage control qualifier is applied to an input file, the file is read as if it had that attribute; when the qualifier is applied to an output file, the file is written with that attribute. Applying a qualifier to an EB file specification causes the qualifier to apply to both input and output files.

The following qualifiers may be used to control input processing:

Qualifier	Meaning
/-CR	Force "none" file record attributes
/CR	Force "Implied" file record attributes
/FT	Force "FORTRAN" file record attributes
/FTN	Same as /FT
/B2	BASIC-PLUS-x handling (see Appendix J)

The following qualifiers may be used to control output processing:

Qualifier	Meaning
/VAR	Create file record format "Var"
/STM	Create file record format "Stm"
/-CR	Create file record format "None"
/CR	Create file record format "Implied"
/FT	Create file record format "FORTRAN"
/FTN	Same as /FT
/B2	BASIC-PLUS-x handling (see Appendix J)

The following qualifiers may be used to control open/create/access options:

Qualifier	Meaning
/RW	Rewind before open/create
/SH	Enable file sharing

## VAX/VMS Operating Characteristics

### G.13 File Record Format

Qualifier	Meaning
/SHR	Same as /SH

Below is a complete list of valid TECO input file record format/attribute combinations and the resulting file format/attribute combinations for EB commands:

Input		Output	
Format	Attribute	Format	Attribute
Fix	none	Var	Implied
Fix	Implied	Var	Implied
Fix	FORTTRAN	Var	FORTTRAN
Var	none	Var	Implied
Var	Implied	Var	Implied
Var	FORTTRAN	Var	FORTTRAN
VFC	Print	Var	Implied
Stm	none	Stm	Implied
Stm	Implied	Stm	Implied

Files read with the EI command have their record attributes interpreted in the same manner. This leads to an unexpected side effect with EI files containing an entire command. The last record of the file presumably contains as its last characters the two alt modes which initiate execution of the macro. If the file is implied carriage control, however, there are also the final carriage return / line feed belonging to the last record, which remain in the type in buffer while the macro executes. If the macro attempts to receive input with the <CTRL/T> command, the carriage return / line feed will be the first two characters read. Alternatively, if the macro does no type in, the carriage return / line feed will be read by TECO as the first two characters of the next command. Then no asterisk (\*) will appear as the prompt for the next command. The remedy for both cases is for the macro to execute an EI` command early on. This causes the remainder of the indirect file to be discarded and further input to be read from the terminal as soon as the double alt mode is encountered.

## G.14 Command Line Processing

The mechanism used to process the command line in VAX/VMS TECO is designed to allow sophisticated TECO users the greatest flexibility in customizing TECO for their own use. It functions as follows:

The initialization routine places a built-in command decoding TECO macro into Q-register Y and the original command line (including the keyword TECO, MAKE, or MUNG) into Q-register Z.

It then looks for a user private command decoding TECO macro by attempting

1. an EITECO`` if the logical name "TECO" exists, otherwise
2. an EISYS\$LOGIN:TECO`` if the logical name "SYS\$LOGIN" exists, otherwise
3. an EITECO``

If the EI succeeds, the found file is executed. The file may, of course, do anything it pleases. Generally, you would use a private command decoder to automate some sort of system specific editing package. For example, to implement an editing package called FOOBAR, you would define the logical TECO with

```
$ DEFINE/SYSTEM TECO dev:[dir]FOOBAR
```

define a DCL foreign command symbol with

```
$ FOOBAR := = $SYSS$SYSTEM:TECO FOOBAR
```

and place the TECO macro that implements the FOOBAR editor in dev:[dir]FOOBAR.TEC. Now, whenever a TECO, MAKE, MUNG, or FOOBAR command is issued, the your macro will gain control. It should fetch Q-register Z (the original command line), check the command type, if it's FOOBAR then go do its thing, else it should terminate and execute TECO's built-in command decoder.

```
GZ                ! Put command into text buffer !
J ::@S/FOOBAR/"U   ! Is the command for us? !
@EI//             ! Not for us, turn us off !
HK               ! Clear out text buffer !
MY               ! Let the real TECO takeover !
'                ! It's for us, just flow on... !
$$               ! Double ESCAPE; 1st command's end !
...              ! Do your own thing here... !
```

This mechanism should not be used for simple initialization; the standard TEC\$INIT facility should suffice for that. This facility is provided for the sophisticated user who wants his own command processing and thus wishes to usurp control.

## G.15 Help

On VAX/VMS, TECO can provide HELP in any of three ways.

You can issue a HELP command at TECO's asterisk prompt. This HELP command obeys the standard VAX/VMS HELP command syntax rules. The HELP command may be terminated by either RETURN or ``.

You can type the / command immediately after an error. (See Section 4.2.) TECO will respond with a one-paragraph description of the error.

You can set EH to 3. (See Section 5.16.) This will cause TECO to automatically respond with a one-paragraph description of any error which occurs.

## G.16 Installing TECO

TECO is distributed with VAX/VMS; the files are already in place. If TECO will receive heavy use, it should be installed /OPEN and /HEADER\_RESIDENT.

The following commands, if used in a LOGIN.COM file, would define the three normal TECO invocation commands:

```
$ TE*CO := = $SYSS$SYSTEM:TECO TECO
$ MA*KE := = $SYSS$SYSTEM:TECO MAKE
$ MU*NG := = $SYSS$SYSTEM:TECO MUNG
```

## VAX/VMS Operating Characteristics

### G.16 Installing TECO

The asterisk allows for abbreviation of these commands. For example, the TE\*CO definition permits TE, TEC, or TECO.

One may include command qualifiers in the command definitions. For example, one can define a command to invoke TECO with VTEDIT as follows:

```
$ VTECO := $SYSS$SYSTEM:TECO TECO /VTEDIT
```

### G.17 Callable TECO interface routines

Callable routines return reasonable error status. If, for example, TEC\$DO\_COMMAND exits with a search failure, (?SRH), the value TECOS\_SRH is returned.

All TECOS symbols are resolved by linking to SYSS\$SHARE:TECOSHR.EXE.

**Table G-1 TECO symbols**

Symbol	Hexadecimal	Decimal
TECOS_BNI	00FF8008	16744456
TECOS_CPQ	00FF8010	16744464
TECOS_DTB	00FF8018	16744472
TECOS_ERR	00FF8020	16744480
TECOS_FNF	00FF8028	16744488
TECOS_IAA	00FF8030	16744496
TECOS_IEC	00FF8038	16744504
TECOS_IFC	00FF8040	16744512
TECOS_IIA	00FF8048	16744520
TECOS_ILL	00FF8050	16744528
TECOS_ILN	00FF8058	16744536
TECOS_IPA	00FF8060	16744544
TECOS_IQC	00FF8068	16744552
TECOS_IQN	00FF8070	16744560
TECOS_IRA	00FF8078	16744568
TECOS_ISA	00FF8080	16744576
TECOS_ISS	00FF8088	16744584
TECOS_IUC	00FF8090	16744592
TECOS_MEM	00FF8098	16744600
TECOS_MRP	00FF80A0	16744608
TECOS_NAB	00FF80A8	16744616
TECOS_NAC	00FF80B0	16744624
TECOS_NAE	00FF80B8	16744632
TECOS_NAP	00FF80C0	16744640
TECOS_NAQ	00FF80C8	16744648

(continued on next page)

## VAX/VMS Operating Characteristics

### G.17 Callable TECO interface routines

**Table G–1 (Cont.) TECO symbols**

Symbol	Hexadecimal	Decimal
TECOS_NAS	00FF80D0	16744656
TECOS_NAU	00FF80D8	16744664
TECOS_NFI	00FF80E0	16744672
TECOS_NFO	00FF80E8	16744680
TECOS_NYA	00FF80F0	16744688
TECOS_OFO	00FF80F8	16744696
TECOS_PDO	00FF8100	16744704
TECOS_POP	00FF8108	16744712
TECOS_SNI	00FF8110	16744720
TECOS_SRH	00FF8118	16744728
TECOS_STL	00FF8120	16744736
TECOS.UTC	00FF8128	16744744
TECOS_UTM	00FF8130	16744752
TECOS_XAB	00FF8138	16744760
TECOS_YCA	00FF8140	16744768
TECOSM_NOCOMMAND	00000002	2
TECOSM_READONLY	00000001	1
TECOSV_NOCOMMAND	00000001	1
TECOSV_READONLY	00000000	0

## TECO\$INIT

---

## TECO\$INIT

This routine should be called before calling any other interface routines.

### Format

TECO\$INIT

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$INIT returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

None.

---

## TECO\$EDIT

Callable TECO entry point. This routine is called to start the callable editor.

### Format

TECO\$EDIT(in,[out],[com],[opt],[xlt])

### Returns

VMS Usage: cond\_value  
 type: longword integer  
 access: write only  
 mechanism: by value in R0

### Returns

TECO\$EDIT returns a status value. Any TECO error message is mapped into a status code of TECO\$\_xxx, where xxx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### in

VMS Usage: cond\_value  
 type: longword integer  
 access: read only  
 mechanism: by descriptor

Descriptor of the input file name

#### out

VMS Usage: cond\_value  
 type: longword integer  
 access: read only  
 mechanism: by descriptor

Descriptor of the output file name.

Defaults to same name as the input file. [i.e. EBinput-file``]

#### com

VMS Usage: cond\_value  
 type: longword integer  
 access: read only  
 mechanism: by descriptor

Descriptor for the TECO startup command.

If this argument is not specified, and the TECO\$\_M\_NOCOMMAND option is not specified, the standard TECO initialization is done. The logical name TEC\$INIT is used to find a startup command file.

#### opt

VMS Usage: cond\_value  
 type: longword integer  
 access: read only

## TECO\$EDIT

mechanism: by reference

Address of an options longword. Valid options are:

TECO\$M_NOCOMMAND	Don't execute the TECO initialization file
TECO\$M_READONLY	Open the file in INSPECT mode

### xlt

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Address of a bound procedure mask [BPM] for the xlate routine. See the information below.

It is not necessary to initialize TECO before calling TECO\$EDIT.

### Callable TECO Xlate routine

When TECO\$EDIT is called, one of the arguments is the XLATE routine. This routine is called by the :EGXLT command.

The format of this command is:

m,n:EGXLT string`

It calls the XLATE routine with 3 arguments:

- The string given. Passed by descriptor. Leading spaces and tabs are removed from this string.
- Two arguments, passed by reference. The first is the M value, the second N. Modifying these locations modifies the XLATE routine's return values. The return value to the :EGXLT command is, by default, -1 if there is an XLATE routine, 0 if there is none. The XLATE routine itself can modify the return values with a call to TECO\$VALUES.

The XLATE hook is analogous to the callable EDT facility. It allows a user of Callable TECO to easily return to the calling program, perform some operation, and then return to TECO.



---

## TECO\$POINTERS

Returns position of dot and z.

### Format

TECO\$POINTERS([dot],[z])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$POINTERS returns a status value. Any TECO error message is mapped into a status code of TECO\$\_xxx, where xxx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### dot

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by reference

Gets text pointer relative to the start of the buffer (.) in TECO.

#### z

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by reference

Gets a pointer to the end of the text buffer. Z in TECO.

## TECO\$SET\_DOT

---

## TECO\$SET\_DOT

Set pointer.

### Format

TECO\$SET\_DOT(new-dot)

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$SET\_DOT returns a status value. Any TECO error message is mapped into a status code of TECO\$\_xxx, where xxx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**new-dot**  
VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

New value to set into the text pointer.

Must be between 0 and the end of the buffer [Z].

---

## TECO\$ADVANCE\_LINE

Advance lines [nL command].

### Format

TECO\$ADVANCE\_LINE(lines)

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$ADVANCE\_LINE returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### lines

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Number of line terminators to advance.

Acts like the nL command in TECO; 0 moves to the start of the current line, 1 moves forward one line, -1 backward one line.

---

## TECO\$DELETE\_TEXT

Deletes characters.

### Format

TECO\$DELETE\_TEXT(chars)

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$DELETE\_TEXT returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**chars**  
VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by

Number of characters to delete.

Acts exactly like the nD command in TECO. 3 deletes 3 characters after DOT, -3 deletes 3 characters before DOT. Exceeding the buffer boundaries will generate a "DTB Delete too big" error.

---

## TECO\$GET\_Q\_REG

Returns text and value of Q-register.

### Format

TECO\$GET\_Q\_REG(q-name,[value],[text])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$GET\_Q\_REG returns a status value. Any TECO error message is mapped into a status code of TECOS\_xxx, where xxx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### q-name

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for name of the q-register to obtain.

Upper or lower case character, optionally preceded by a dot (.) for local q-registers. Any descriptor type allowed by STR\$COPY\_DX is allowed. The text is copied from the q-register to the destination string.

#### value

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by reference

Gets 'value' portion of q-register.

#### text

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for string to receive the text stored in the q-register.

---

## TECO\$LOAD\_Q\_REG

Loads q-register text and/or value.

### Format

TECO\$LOAD\_Q\_REG(q-name,[value],[text])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$LOAD\_Q\_REG returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### q-name

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for name of the q-register to load.

Upper or lower case character, optionally preceded by a dot (.) for local q-registers. Any descriptor type allowed by STR\$COPY\_DX is allowed. The text is copied to the q-register from the source string.

#### value

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Sets 'value' portion of q-register.

#### text

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for the text string to load into the q-register.

---

## TECO\$MUNG

Executes q-register.

### Format

TECO\$MUNG(q-name)

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$MUNG returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**q-name**  
VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for q-register name to execute.

MUNG executes the TECO program loaded into the given q-register. The current TECO command level's local q-registers are saved before executing the MUNG, unless the q-register name specifies a local q-register.

---

## TECO\$LOAD\_TEXT

Load text buffer.

### Format

TECO\$LOAD\_TEXT(new-text[,delete])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$LOAD\_TEXT returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### new-text

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor for text string to be loaded into the text buffer. The text is copied from the source string to the text buffer.

#### delete

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Number of characters to delete before insert. Negative values delete text before DOT, positive values delete after DOT.



---

## TECO\$GET\_TEXT

Gets descriptor of text buffer.

### Format

TECO\$GET\_TEXT(text[,start][,end])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$GET\_TEXT returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

#### text

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

Descriptor of string to receive a copy of the extracted text. The text extracted is copied to this string.

#### start

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Text buffer offset to start the extract at. Defaults to zero [beginning of buffer, B].

#### end

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Text buffer offset to end at. Defaults to end of text buffer [Z].

The START and END values are checked and limited to the beginning and end of the text buffer, respectively.

---

### TECO\$DO\_COMMAND

Execute a TECO command.

#### Format

TECO\$DO\_COMMAND(command)

#### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

#### Returns

TECO\$DO\_COMMAND returns a status value. Any TECO error message is mapped into a status code of TECOS\_\$xx, where xx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

#### Arguments

##### **command**

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by descriptor

TECO command to execute.

TECO\$DO\_COMMAND executes a single TECO string and returns. Returned values [m,n] can be obtained with the TECO\$VALUES routine.

---

## TECO\$START

Allow user TECO commands.

### Format

TECO\$START([flag])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$START returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**flag**  
VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

Default is zero.

TECO\$START turns control over to the TECO interpreter until an exit command is executed [EX, EG, or :EGXLT]. If the flag is true [low bit set], EXIT commands return control to caller immediately, without paging out the text buffer.

---

## TECO\$GET\_VALUES

Obtain numeric values.

### Format

TECO\$GET\_VALUES([m],[n])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$GET\_VALUES returns a status value. Any TECO error message is mapped into a status code of TECO\$\_xxx, where xxx is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**m**

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by reference

This longword receives TECO's M [first numeric] value.

**n**

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by reference

This longword receives TECO's N [second numeric] value.

This routine is used to obtain numeric values from TECO code.

---

## TECO\$SET\_VALUES

Set numeric return values.

### Format

TECO\$SET\_VALUES([m],[n])

### Returns

VMS Usage: cond\_value  
type: longword integer  
access: write only  
mechanism: by value in R0

### Returns

TECO\$SET\_VALUES returns a status value. Any TECO error message is mapped into a status code of TECOS\_XXX, where XXX is the TLA [three-letter-abbreviation] of the error message. (See Appendix B)

### Arguments

**m**

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

This longword is use to set TECO's M [first numeric] value.

**n**

VMS Usage: cond\_value  
type: longword integer  
access: read only  
mechanism: by reference

This longword is used to set TECO's N [second numeric] value.

This routine is used to set numeric return values for TECO code. If the M argument is present, it sets the COMMA flag.



---

## OS/8 Operating Characteristics

### H.1 Startup

TECO is started with the:

```
.R TECO
```

command. TECO is now immediately ready to accept commands. The text buffer and Q-register areas are empty.

The TECO command

```
.TECO filespec
```

is used to edit an already existing file. It is equivalent to:

```
.R TECO
*EBfilespec`Y``
```

OS/8 "remembers" the filespec as the name of the last file that has been edited.

The MAKE command:

```
.MAKE filespec
```

is used to create a new file. It is equivalent to:

```
.R TECO
*EWfilespec``
```

OS/8 "remembers" the filespec as the name of the last file that was edited.

The command:

```
.TECO filespec1=filespec2
```

is used to edit filespec2 into filespec1. That is, filespec2 is opened as the input file, and filespec1 is created as the output file. It is equivalent to:

```
.R TECO
*ERfilespec2`EWfilespec1`Y``
```

OS/8 "remembers" the filespec1 as the name of the last file that was edited.

The command:

```
.TECO
```

with no arguments, causes CCL to execute the command:

```
.TECO filespec
```

where filespec was the file that was previously remembered as the last file to be edited. The system purposely does not remember filenames from one day to the next, but it will remember names across bootstraps.

## OS/8 Operating Characteristics

### H.1 Startup

The command:

```
.MUNG filespec
```

executes the specified TECO program. The default extension is .TEC . This is equivalent to the sequence:

```
.R TECO
*ERfilespec`YHXYHKMY``
```

Another format of this command is:

```
.MUNG filespec,argument
```

which is used to pass an argument to the TECO program to control its action. This is equivalent to the sequence:

```
.R TECO
*ERfilespec`YHXYHKIargument`MY``
```

The argument may be the name of a file that the TECO program is to mung, or it may be a command to the program to specify what action to take, or whatever. It is up to the TECO program to decode this argument (which is left in the text buffer) and take appropriate action. A TECO program executed via the MUNG command must never destroy the text storage area of Q-register Y and expect to ever see the light of day again.

Note the input file remains open and can provide more input to the macro.

## H.2 Startup Conditions

The initial value of the EU flag is 0 if the CCL command SET TTY NO SCOPE had been previously issued, and is -1 if the CCL command SET TTY SCOPE had previously been issued.

The initial value of the ET flag is as follows:

Bit	Initial value
1	0
2	0 (1 if terminal is a scope)
4	0
8	0
16	0
32	0
64	0
128	1 (TECO's prompt sets this to 0)
256	0
512	0 (1 if VT support is present)
1024	0 (1 if VR12 support is present)
2048	0

The initial value of the ED flag is 1.



## H.3 File Specification

The file access commands ER, EB, and EW accept a file specification in the standard OS/8 format:

dev:filename.type

in which dev: is a physical device name or a user assigned logical name; if dev: is not specified, the default DSK: is assumed. The filename field must be specified in the commands ER, EB, and EW and be a legal OS/8 filename. The type field is a file extension and must be explicitly given if used (there is no default). Any characters after the second will be ignored, thus the filespecs FOO.TEC and FOO.TE are equivalent. The EB and EW commands do not accept the extended notation for an output file size:

dev:filename.type[n]

specifying an output size allocation.

## H.4 Backup Files

The EB command maintains one level of file backup on OS/8. The pre-edited input file name is changed to:

filename.BK

before the new output file is closed with the original name. Only normal file closing commands (EC, EF, EG, and EX) cause this renaming to happen. If TECO is aborted or the output file is purged by the EK command, the input filename remains unchanged. Note only one .BK file for a given name is kept; earlier .BK backup files are deleted each time a new backup file is created.

A good policy to follow when editing is to close the edited file frequently enough so that an unexpected incident would not cause a substantial loss of work. Files should be backed up regularly. TECO has the power to let an unsuspecting user alter a good file into a completely useless state. The SRCCOM program can be used to verify an editing session.

## H.5 EXIT and GO

If TECO is exited via the EGstring` command, the string is passed to the system as the next command to execute. This string may be any valid command or an indirect command file specification. The command may be either a KBM or a CCL command. This command is especially useful while running under BATCH.

If TECO is exited via the EG` command, then OS/8 will re-execute the last explicit compile-class command that was executed that day. The commands that are considered to be compile-class commands are:

COMPILE file  
LOAD file  
EXECUTE file  
LINK file  
MACRO file

This feature, combined with OS/8's other remembering features, minimizes the number of keystrokes necessary to do normal program development. The programmer does not have to constantly type in the name of the file he is working with. A typical debugging session would look like this:

## OS/8 Operating Characteristics

### H.5 EXIT and GO

```
.MAKE FOO.MAC
*!type in assembly language file to be executed!
*EX`
.EXECUTE FOO
(get error messages)
.TECO
*!fix bugs!
*EG` !re-compile and execute program!
(watch program work or repeat process)
```

### H.6 <CTRL/C>

The action taken when the user types <CTRL/C> depends on what TECO is doing. At command level <CTRL/C> is an immediate action command. If typed as the very first character in a command string (not necessarily the first keystroke) it aborts TECO and returns to the keyboard monitor. If this was done accidentally, TECO may be restarted (at your own risk) by using ODT to branch to location 207 in your program's image. If <CTRL/C> is typed in the middle of entering a command string, then the ?XAB error message is given and TECO reprompts with its asterisk. Note that if TECO executes <CTRL/C> as a command from command level, TECO is aborted. If TECO executes a <CTRL/C> command from within a macro, TECO is also aborted. If a <CTRL/C> is typed while TECO is running, or while TECO is typing on the terminal, or while an error message is printing, then the ?XAB error message is given and TECO reprompts with its asterisk. TECO will abort similarly, if <CTRL/C> is typed while TECO is waiting for input because of a ^T command. Note that if TECO is performing I/O using non-system handlers, the non-system handler may intercept the <CTRL/C> and abort back to the keyboard monitor. In such a case, you may attempt to re-enter TECO. However, part of your file has been lost; good luck in attempting to issue an EF command. Manually resetting the value of Z might recover your data.

If TECO is executing commands or doing I/O, a <CTRL/C> will stop the operation and generate the ?XAB error message.

Sometimes it is desirable for a TECO macro to detect when a <CTRL/C> was typed. By detecting the <CTRL/C>, the macro can exit cleanly back to command level (restore any flag values, etc.). To do this, the macro sets Bit 0 (Octal 4000, Decimal 2048) of the ET flag. When a <CTRL/C> is typed, TECO will automatically turn off Bit 0, but will continue execution of the macro. The macro periodically checks Bit 0 and exits cleanly if it ever goes off. If the <CTRL/C> trap bit is on, then the ^T can read a <CTRL/C> typed at the terminal. It has an ASCII value of 3.

### H.7 File Recovery

TECO can be a useful tool in recovering ASCII files lost on a block replaceable device. TECO allows non-file-structured devices to be opened in a non-file structured mode. This gives the user the capability to open a disk and access ASCII data anywhere on it, independent of file boundaries. To do this, you must issue a command of the form:

.SET dev: NOFILES

to the monitor to make it think that your disk is non-file-structured. The command:

ERdev:`

is used to open the device at which point \_ (underscore or backarrow) searches may be used to locate specific ASCII data and transfer it to new output files. Note that files tend to get reproduced, in whole or part, many places on a block replaceable device; be sure to verify that any given text is indeed complete and the correct version.

If the disk's directory has not been clobbered (or if you are willing to create a new one), then it is not necessary to turn the disk into a non-file-structured device. Merely open up a file early on the disk for input and read through end-of-files until you locate the lost file. To read through end-of-files, you must use the /S switch on an ER, EB, or EW command. For example, the command:

ERFOO.MAC/S`

will open the file FOO.MA for input and put TECO into "SUPERTECO" mode. In this mode, TECO will not treat a <CTRL/Z> found in a file as an end-of-file character. Instead, <CTRL/Z> will be treated like any other character. It is not a line terminator or a page terminator. This mode continues until an ER, EW, or EB command is issued without a /S switch.

## H.8 VR12 Graphics Support

If TECO is run on a PDP-12, TECO will automatically start up in display mode, adjusting to both the size of the display screen and to the presence or absence of the scroller.

On a PDP-12, TECO only permits one-page input and output handlers.

See Section 5.17 for a description of the available commands to interact with the display.

Various aspects of the display screen become immediately obvious upon seeing them; the text pointer, its position and shape and its position between lines; wrap around of more than 72 characters per line, and so on. Experiment with a scratch file for more familiarity.

## H.9 Exceptions

TECO-8 does not support the following commands which are described in this manual:

1. Secondary streams (EP, EA, ER`, EW`)
2. Auxiliary command streams (EI)
3. Wildcards (EN)
4. Zeroing of directories (EZ)
5. Magtape commands (EM)
6. View command (nV)
7. Bounded searches
8. Anchored searches

## OS/8 Operating Characteristics

### H.9 Exceptions

9. Search verification (ES)
10. Command verification (EV)
11. Backward searches
12. Extended string build or match constructs (^Ex)

The following incompatibilities exist between TECO-8 and Standard TECO:

1. In octal mode, the digits 8 and 9 are not treated as errors when occurring in a numeric string.
2. The \*q immediate action command is not implemented. Instead, the immediate action command \* has the same effect as \*Z of the standard. (The immediate mode command ^S is still accepted for compatibility with OS/8 TECO V5.)

### H.10 Chaining to TECO

A user program may chain to TECO passing it a command to be executed. There are two formats that such a command may take.

Format 1 (the TECO command format) passes TECO a valid TECO command to be executed. This TECO command is placed in a buffer starting at location 17600, one 7-bit ASCII character per word. A negative word represents a pointer to a continuation buffer in field 1. There may be any number of continuation buffers, but they must all begin above location 4000 in field 1. Since TECO clobbers most of field 1, these buffers must in fact start above location 7400. TECO will never load into page 7400 of field 1. The buffer ends with a fullword 0.

Format 2 (the CCL command format) passes TECO a CCL command to be parsed and executed. Such a command usually begins with the words TECO, MAKE, or MUNG, but is not limited to these words. Such a CCL command is placed in a buffer starting at location 17601, one 7-bit ASCII character per word. Location 17600 must be a fullword 0 to specify that this format is being used. A negative word in the buffer represents a pointer to a continuation buffer in field 1 as described above. The buffer ends with a fullword 0. If this format is used, the passed CCL command will be parsed and executed by TECO.TEC as described below. A user may write his own TECO.TEC, thus implementing his own CCL commands. There is no limit to the possibilities, other than the user's imagination.

### H.11 User Initialization

If a user has a file called TECO.INI on SYS:, then when TECO starts up (via a CCL command, it will execute the contents of this file (as a TECO macro). This file must contain a valid TECO program (which will execute out of Q-register W). God help you if you have any errors in this program. This start-up file must not modify itself (Q-register W) and must not modify the contents of Q-register V. It should not indiscriminately modify the contents of Q-register Z or the text buffer. TECO.INI will be executed before TECO opens any files. That is, if TECO was invoked via a MAKE command, TECO.INI will be executed before the EW command (for the MAKE) is executed. At this point, the text buffer will contain a copy of the CCL command that invoked TECO (assuming your monitor has TECO.TEC support). However, TECO has not as yet parsed this line. The user may examine this line for himself, and modify it, but you had better know what you are doing (and do it right!). TECO.TEC will parse the contents of the text buffer at the conclusion of execution of TECO.INI.

If your monitor does not have TECO.TEC support, or if a user program chained to TECO passing it a TECO command (rather than a CCL command), then the initial TECO command will be in Q-register Z when TECO.INI gets control. That command has not as yet been executed. The initialization file may examine the contents of Q-register Z to determine what TECO command will be executed and proceed accordingly. It may also modify the contents of Q-register Z (but you better know what you are doing).

In this case, TECO.INI is started up via the sequence

```
@:ER/SYS:TECO.INI/"SYHXWHK
@^UZ^@teco command^@
MW+0ES,,XWMZES"N0ESMX``
```

which loads TECO.INI into Q-register W, loads the chain argument consisting of an appropriate teco command into Q-register Z, and temporarily stores the value returned by TECO.INI in the search verification flag (this feature may change in a subsequent release). Q-register W and ES are cleared before the post-processing command in Q-register X is executed. Note that the chain argument may not contain any embedded nulls.

## H.12 Returned Values from TECO.INI

TECO.INI may also return a value. If your monitor does not support TECO.TEC, then only two values are permitted. Returning a 0 (or not returning anything) is the normal sequence of events. Returning a 1 means that TECO should execute the contents of Q-register X (via an MX command) after it executes the initial TECO command (in Q-register Z). TECO.INI may set up Q-register X with the appropriate post-processing commands. A typical use of this feature would be to have TECO.INI load up Q-register I with an editing macro and then put an "MI" command in Q-register X for subsequent execution. If your monitor does have TECO.TEC support, then TECO.TEC can support additional returned values. It is recommended that TECO.TEC support the returned values of 0 and 1 as above, but in addition, it may support additional values determined by the user.

Note that TECO.INI is not invoked if TECO is started with a RUN or R command.

## H.13 TECO.TEC Support

If the version of CCL you are using to invoke TECO supports TECO.TEC, then it will chain to TECO with a 0 at location 17600 and will pass TECO the invokig CCL command (beginning at location 17601). If TECO is invoked in this manner, it will parse this CCL command by executing the TECO command line parser macro stored in SYS:TECO.TEC. This macro can be modified by the user to parse switches or do any special processing that is desired. TECO.TEC is started up via the command:

```
@I^@ccl command^@:ER/SYS:TECO.TEC/"F^ACan't find SYS:TECO.TEC
^A^C^CA,,ZXV,,ZKMV,,XV``
```

which puts your CCL command in the text buffer and then loads (the first page of) TECO.TEC into Q-register V. TECO.TEC is then executed with the MV command and then Q-register V is cleared. It is the responsibility of TECO.TEC to parse the command line in the text buffer and do the appropriate processing and clean-up. It is also the responsibility of TECO.TEC to execute a user's start-up file (TECO.INI) if one is present. Note that TECO.TEC is not invoked if

## OS/8 Operating Characteristics

### H.13 TECO.TEC Support

TECO is started via a RUN or R command. Also note, that the CCL command may not contain any embedded nulls.

## H.14 Overlays

The key to writing fast TECO programs lies in understanding TECO-8's overlay structure. If TECO-8 is run in 16K or more (20K or more if VT support is present), then the overlays will be memory-resident rather than disk-resident. Although this is much faster than swapping from the disk, swapping from memory still involves some overhead, so it would be wise to structure your TECO program to minimize the number of swaps necessary.

The overlay structure is designed so that the minimal number of swaps will be required unless obscure TECO features are used. There are five overlays to TECO:

1. The I/O-overlay. This overlay handles file opening and is initially resident. Thus no swapping is necessary to do an initial ER, EW, or EB.
2. The Q-overlay. This overlay contains most of the frequently used conditional commands and branching commands. It is intended that this overlay swap in once and remain in memory until TECO is exited.
3. The X-overlay. This is the exit overlay and handles commands needed only when TECO is exiting, such as EX, EF, EC, and EG. It is intended that this overlay will swap in only once when you are ready to leave TECO.
4. The F-overlay. This overlay contains the flag commands and other little-used commands. It is intended that this overlay be not used at all, or if it is used, it will be used so infrequently that it will not slow down system performance.
5. The E-overlay. This is the error overlay. It is swapped in only when an error occurs. It is intended that this overlay never be swapped in.

To write efficient TECO code, the user must know exactly which commands are handled by which overlay. This information is summarized below.

Overlay	Commands
I-overlay	ERfile`, EWfile`, EBfile`, :ERfile`, :EBfile`
Q-overlay	Otag`, n"Xthen   else`, n:, search:, n<...>, <...>
X-overlay	EC, EG`, EGcmd`, EF, EK, EX, *q, ?, nEJ, n^_, V, ^B, ^E, ^F, ^L, ^N, ^Uqtext`
F-overlay	ED, EH, EO, ES, ET, EU, ^D, ^O, \, n\, n=, n=, n:=, n:=,

Several things are immediately obvious. The command OTT should always be preferred to the V command. ELSE clauses should be avoided. (In future releases, we will try to move the processing of the | command into overlay Q.) The commands \ and = should be used as infrequently as possible from within long-running macros. Xq is preferred to ^Uq to load up a Q-register. -n-1 is preferred to n^\_ to take a one's complement. Radix changes should be avoided. Flags, such as ET and ED, should be set once at the beginning of a macro, and then not fiddled with if at all possible.



## H.15 Installation Instructions

The source of TECO consists of the following modules:

TECO.MAC	Main module
TECINI.MAC	Initialization module
TECTBL.MAC	Tables
TECDEF.MAC	Global definitions
TECO12.MAC	VR12 support
TECOVT.MAC	VT support
TECOVI.MAC	I/O-overlay
TECOVQ.MAC	Q-overlay
TECOVX.MAC	X-overlay
TECOVF.MAC	F-overlay
TECERR.MAC	E-overlay and error processor
TECSRH.MAC	Search processor
TECNUM.MAC	Arithmetic processor

Each of these modules should be assembled (using MACREL V2 or later). This can be accomplished via the command

```
.MAC TEC???MAC
```

if your monitor supports wildcards in compile-class commands.

The resulting relocatable modules are then linked together (using LINK V2 or later) to produce the executable TECO.SV image which should be put on SYS: (but it may reside on any device). If your monitor supports TECO.TEC, then TECO.TEC must be placed on SYS:.

## H.16 Arithmetic Precision

TECO-8 performs 13-bit arithmetic except that multiplication and division by negative numbers gives unpredictable results. All numbers stored in Q-registers are 13 bits long. Numbers stored in flags (such as ET, EU, etc.) are only 12-bits long. When storing a number into a flag, the high order (sign bit) is lost. When using the value of a flag in an arithmetic expression, the 12-bit value is sign extended first.

## H.17 Alternate Starting Address

The normal starting address of TECO is location 00200. In this (normal) mode, TECO will simulate tabs by spaces on type out and will simulate vertical tabs and form feeds by line feeds. If your terminal has hardware tabs and vertical tabs (such as a KSR-35), then TECO can take advantage of these features. To enable this ability, you should change TECO's starting address to be 05200. This can be done by the monitor commands:

```
.GET SYS:TECO  
.SAVE SYS:TECO;5200
```

## **OS/8 Operating Characteristics**

### **H.18 VT05 Support**

#### **H.18 VT05 Support**

TECO will automatically handle command string scope editing correctly on a VT05. The VT support (obtained via use of the -1W command) will handle VT05's correctly. The VTEDIT macro does not currently support the VT05 keypad.



---

# TOPS-10 Operating Characteristics

## I.1 STARTUP

TECO is started with the:

```
.R TECO
```

command. TECO is now immediately ready to accept commands. The text buffer and Q-register areas are empty. Initial commands may also be specified by following the monitor command with a dollar sign (\$) and then some TECO commands. For example:

```
.R TECO $3EH
```

starts TECO with the help level flag set to 3.

The TECO command

```
.TECO filespec
```

is used to edit an already existing file. It is equivalent to:

```
.R TECO  
*EBfilespec`Y``
```

TOPS-10 "remembers" the filespec as the name of the last file that has been edited.

The MAKE command:

```
.MAKE filespec
```

is used to create a new file. It is equivalent to:

```
.R TECO  
*EWfilespec``
```

TOPS-10 "remembers" the filespec as the name of the last file that was edited.

The command:

```
.MAKE filespec1=filespec2
```

is used to edit filespec2 into filespec1. That is, filespec2 is opened as the input file, and filespec1 is created as the output file. It is equivalent to:

```
.R TECO  
*ERfilespec2`EWfilespec1`Y``
```

TOPS-10 "remembers" the filespec1 as the name of the last file that was edited.

# TOPS-10 Operating Characteristics

## I.1 STARTUP

The command:

.TECO

with no arguments, causes CCL to execute the command:

.TECO filespec

where filespec was the file that was previously remembered as the last file to be edited. The system purposely does not remember filenames from one editing session to the next, that is, when you log out, the system "forgets" the name of the file you were editing.

TECO-10 does not require the use of the MUNG command to execute TECO macros because runnable TECO programs can be created via use of the EE command and these can then be run with the standard R or RUN command. This TECO command has the format:

EEfilespec`

which saves away the current image of TECO in the filename specified. The default extension is .EXE. When the file is subsequently run (using the R or RUN monitor command), TECO resumes execution with the TECO command immediately following the EE command.

## I.2 Startup Conditions

The initial value of the EU flag is 0 if you are running on a terminal that does not support lower case, and is -1 if you are running on a terminal that does support lower case.

The initial value of the ET flag is as follows:

Bit	Initial value
1	0
2	0 (1 if terminal is a scope)
4	1
8	0
16	0
32	0
64	0
128	1 (TECO's prompt sets this to 0)
256	0
512	0 (1 if VT support is present)
1024	0
2048	0

The initial value of the ED flag is 1.

## I.3 File Specification

The file access commands ER, EB, and EW accept a file specification in the standard TOPS-10 format:

dev:filename.type[p,pn]

in which dev: is a physical device name or a user assigned logical name; if dev: is not specified, the default DSK: is assumed. The filename field must be specified in the commands ER, EB, and EW and be a legal TOPS-10 filename. The type field is a file extension and must be explicitly given the first time. Thereafter, if a corresponding command is given with no extension specified, the system uses the previously specified extension as the default. The same defaulting rules hold for the dev: field. The <prot> construct is permitted on any output filespecification to allow setting the protection of the file being created.

## I.4 Backup Files

The EB command maintains one level of file backup on TOPS-10. The pre-edited input file name is changed to:

filename.BAK

before the new output file is closed with the original name. Only normal file closing commands (EC, EF, EG, and EX) cause this renaming to happen. If TECO is aborted or the output file is purged by the EK command, the input filename remains unchanged. Note only one .BAK file for a given name is kept; earlier .BAK backup files are deleted each time a new backup file is created.

A good policy to follow when editing is to close the edited file frequently enough so that an unexpected incident would not cause a substantial loss of work. Files should be backed up regularly. TECO has the power to let an unsuspecting user alter a good file into a completely useless state. The FILCOM program can be used to verify an editing session.

## I.5 EXIT and GO

If TECO is exited via the EG` command, then TOPS-10 will re-execute the last explicit compile-class command that was executed during that session.

## I.6 <CTRL/C>

The action taken when the user types <CTRL/C> depends on what TECO is doing. At command level <CTRL/C> is an immediate action command. If typed as the very first character in a command string (not necessarily the first keystroke) it aborts TECO and returns to the monitor. No Control-C trapping is available under TOPS-10. The ?XAB error message is not supported. If <CTRL/C> is typed in the middle of entering a command string, then TECO returns control to the monitor. Note that if TECO executes <CTRL/C> as a command from command level, TECO is aborted. If TECO executes a <CTRL/C> command from within a macro, TECO is also aborted. If two consecutive <CTRL/C>s are typed while TECO is running, or while TECO is typing on the terminal, or while an error message is printing, then control returns to the operating system. If one <CTRL/C> is typed to TECO while it is waiting for input, then control returns to the operating system.

### I.7 Exceptions

TECO-10 does not support the following commands which are described in this manual:

1. Secondary streams (EP, EA, ER`, EW`)
2. Wildcards (EN)
3. Immediate aids LF and BS.

The following incompatibility exists between TECO-10 and DEC's TOPS-10 TECO V24:

The nA command under TOPS-10 TECO V24 always returned the value of the current character, regardless of the value of n. In TECO-10, 0A gives the value of the current character.

### I.8 User Initialization

If a user has a file called TECO.INI in his area, then when TECO starts up (via a CCL command), it will execute the contents of this file (as a TECO macro). This file must contain a valid TECO program. TECO.INI will be executed before TECO opens any files. That is, if TECO was invoked via a MAKE command, TECO.INI will be executed before the EW command (for the MAKE) is executed.

### I.9 Installation Instructions

To create TECO for TOPS-10 from the sources, issue the following commands:

```
.LOAD/MAC/COMPILE TECO10.T10+TECO10.MAC
.SAVE TECO10
.LOAD/MAC/COMPILE TECERR.T10+TECO10.MAC
.SAVE TECERR
```

To create TECO for TOPS-20 from the sources, issue the following commands:

```
@LOAD/MAC/COMPILE TECO10
@SAVE TECO20
```

This builds a raw TECO. This version of TECO does not contain any window support since the W and :W commands are implemented as macros. To load window support, issue the following commands:

```
.RUN TECO10 (or TECO20)
*EITECO10.TEC``
*EETECO``
```

You now have a runnable TECO image with window support.

### I.10 TMPCOR Support

The EQ and E% commands support the pseudo-device TMP: for TMPCOR. Only the first three letters of the file name will be used, to try and access a TMPCOR file. If that fails, it will try nnnNAM.TMP where nnn is your job number and NAM is the three-character name. For example: for job 23, EQqTMP:FOOBAR` will read TMPCOR file FOO or 023FOO.TMP.

## I.11 Q-Register Names

Any printable character (except open parenthesis) is valid as a Q-register name. A Q-register whose name is a lower case alphabetic character is the same as the Q-register whose name consists of the corresponding upper case letter. Thus Qa and QA are equivalent commands. Q-register names may also be up to 6 characters long, by enclosing the name in parentheses, for example, Q(FOOBAR). Q-register names may contain any printable characters, however all characters other than letters, digits, dollar-sign, space, and underline are reserved for special use by TECO. A Q-register name consisting entirely of zero or more spaces is the same as Q-register (), which is special and discussed below. Trailing spaces in Q-register names are discarded, and lower case is converted to upper case.

## I.12 Referencing the Text Buffer as a Q-Register

The Q-register with the null name: () is the text buffer. The numeric part of this Q-register is the value of dot. The sequence [A]() causes Q-register A to share with the text buffer. The old main text buffer is lost (unless it is also sharing with some Q-register or if it has been saved on the Q-register push-down list). The text in Q-register A becomes the text buffer and the numeric part of Q-register A is used for "." if it is in range, otherwise dot is set to 0.

## I.13 Sharing of Q-Register Pointers

Q-registers may share their text with each other and with the text buffer as a result of [ and ] commands. When a Q-register is pushed onto the Q-register pushdown list, all that is pushed is the numeric part of the Q-register and a pointer to the text part of the Q-register. Thus a command such as [A]B would cause Q-registers A and B to share the same text. The commands X, ^U, and EQ could be applied to either Q-register without modifying the other, since the Q-register is unbound from its previous text first. However, the colon-modified forms of X and ^U append to the existing text, so a :X or :^U command for either of them would affect the other.

## I.14 Editing Line Sequence Numbered Files

Some ASCII files have a special type of line number at the beginning of each line. These "line-sequence numbers" conform to certain rules so that they may be ignored or treated specially by compilers and other programs. The standards for line-sequence numbers are given in the LINED Program Reference Manual.

TECO does not need line-sequence numbers for operation, but TECO can be used to edit files containing them. If such a file is edited with TECO-10, the line-sequence numbers are, in the normal case, simply preserved as additional text at the beginning of each line. The line-sequence numbers may be deleted, edited, and inserted exactly like any other text. On output, the line-sequence numbers are output according to the standard, except that the tab after the number is output only if it is already there. Leading zeros are added as necessary. If a line without a line-sequence number is encountered, a line-sequence number word of five spaces is placed at the beginning of the line.

The following switches are available for use with line-sequence numebred files. These switches are merely added to the appropriate file selection command.

ERfilespec/SUPLSN`

## TOPS-10 Operating Characteristics

### I.14 Editing Line Sequence Numbered Files

EBfilespec/SUPLSN`

causes line sequence numbers to be suppressed at input time. The numbers will not be read into the editing buffer. Also, the tabs following the line-sequence numbers, if they exist, will be suppressed.

EWfilespec/SUPLSN`

causes the line-sequence numbers to be suppressed at output time. Tabs following the line-sequence numbers will also be suppressed if they exist.

EWfilespec/GENLSN`

EBfilespec/GENLSN`

causes line sequence numbers to be generated for the output file if they did not already exist in the input file. Generated line-sequence numbers begin at 00010 and continue with increments of 10 for each line.

Note that these switches are needed only if a change is to be made in the format of the file being edited. If no switches are specified, a file is output in the same form as it was input.

### I.15 Compiler Restrictions

TECO-10 is a compiler rather than an interpreter. This means, that before your command string is executed, TECO-10 compiles it into assembly language code. This makes it much faster than most other TECOs. Before executing a macro (with the Mq command) TECO compiles the program in the macro. The next time the macro is executed, TECO notes that the macro has already been compiled and merely branches to the compiled code. If the contents of the Q-register are changed (via an X or U command), then TECO notes that it must re-compile the commands should the Q-register be invoked as a macro.

One consequence of this is that if a syntax error is detected in a command, no portion of that command will have been executed. For example, typing the command HK= will yield the ?NAE error message and the text buffer will NOT be cleared. Another consequence of this is that you must not invoke a macro two different times using two different numbers of arguments. If a macro gets initially invoked with two arguments, then all subsequent invocations must supply two arguments. Also, TECO cannot tell while compiling an Mq command whether or not the macro returns a value. Therefore it assumes that a value is always returned. This value can be explicitly removed by following the Mq command with an <ESCAPE>. The MqA command will compile the A command as if it were an nA command rather than an APPEND.

---

## Basic-Plus/Basic-Plus-2 Handling

### J.1 Purpose

In BASIC-PLUS and BASIC-PLUS-2 a program statement may be continued over more than one text line. A text line which is not the last text line of a given statement may need to be flagged as a "continued" line. (See the appropriate language reference manual for requirements of each version of each language processor.)

The standard continuation flag is the & character. A line which must be marked as "continued" is written with an & as the last non-whitespace character before the <CR><LF>.

(An older form of continuation was, in BASIC-PLUS only, to end the text line by typing a <LF> key instead of <CR>.)

The switches described in this appendix cause TECO to manipulate the text files in such a way that each text line appears in the editing buffer to be completely "conventional": that is, each text line ends with a standard <CR><LF>, and no text line has a trailing & character. You can correctly edit well-formed BASIC-PLUS or BASIC-PLUS-2 source files without having to be concerned about continuation conventions.

### J.2 Method

TECO allows you to work with "unencumbered" lines in the editing buffer by removing continuation conventions as each text line is read in. You indicate that you want this processing by specifying a switch on the file specification supplied to the ER (or the EB) command.

TECO (again) adds appropriate continuation conventions to each text line as it is written out from the editing buffer. You indicate that you want this processing by specifying a switch on the file specification supplied to the EW (or the EB) command.

### J.3 Input Processing

As TECO reads each new text line into the editing buffer, it is examined for a trailing & character. If one is found, TECO removes it, and then additionally removes any trailing spaces and/or tabs.

(In the case of <LF> continuation, TECO converts that kind of line terminator character sequence to <CR><LF>.)

The available switches follow. (Consult the appropriate appendix to see which switches are supported on your operating system.)

/B2	TECO strips trailing & sequences
-----	----------------------------------

## Basic-Plus/Basic-Plus-2 Handling

### J.3 Input Processing

/n	Same as /B2 for input processing
/B+	TECO changes <LF><CR><NUL> sequences to <CR><LF>.

### J.4 Output Processing

As TECO writes each text line from the editing buffer, it takes a look at the text line which will follow. If the following line begins with a digit, the current line is written without change. If the following line does not begin with a digit, TECO assumes that the current line must be continued.

The convention applied to an output text line which TECO marks as "continued" depends on the form of the switch you specify. (Consult the appropriate appendix to see which switches are supported on your operating system.)

/B2	TECO appends a space and an &
/n	TECO appends an &, after padding out the line with tabs and/or spaces to make the & appear in column n
/B+	TECO ends the line with a <LF><CR><NUL> sequence (instead of <CR><LF>.)

### J.5 Form Feed Interaction

When TECO reads a page of text into the editing buffer, it stops when a <form feed> character is encountered, or when the buffer has been filled to capacity. In the latter case, the last line will be complete, and will include its line delimiter. However, no look-ahead is done to examine the next line in the file.

If TECO is being used with one of the switches described in this appendix, a very long BASIC-PLUS-x source program which is not segmented with <FF> characters can cause the last line in the buffer to not be the last text line of a multi-line source statement.

When the buffer is written to the output file, TECO has no way of determining whether the next text line (the first line of the next buffer load) will begin with a line number. TECO makes the assumption that the buffer's last text line is not to be "continued", and terminates it with <CR><LF>. If this assumption is incorrect (frequently the case), remaining text lines of that multi-line statement will be lost when the program is subsequently OLDed.

To avoid this problem, lengthy BASIC-PLUS or BASIC-PLUS-2 programs should be segmented with <form feed> characters before editing them using these TECO switches. Segments of approximately 150 to 200 lines are convenient. Starting a new page even more frequently to make listings readable isn't a bad idea, either.



---

# VTEDIT—A keypad editing macro for TECO-11 and TECO-32

## K.1 Introduction

The Video Terminal Editor (called VTEDIT) is a feature of the TECO text editor which is available under RSTS/E, RT-11, RSX-11M, IAS, and VAX/VMS. VTEDIT allows you to see all changes to the text as you make them.

The Video Terminal Editor has two parts:

- The scope driver, which keeps the terminal screen updated, is a part of the TECO run-time system (RSTS/E), the TECO.SAV program (RT-11), the TEC.TSK task (RSX-11M and IAS), or the TEC.EXE image (VAX/VMS).
- The keyboard interface, which accepts your typed input and either inserts it into the text buffer or interprets it as an editing command, is a TECO macro called VTEDIT.TEC.

This appendix describes the scope driver and the VTEDIT macro. The scope driver is a standard part of TECO. However, VTEDIT is just one of many possible keyboard interfaces. Your installation may have modified it, and therefore may have its own version of the scope editing keyboard interface.

### K.1.1 How to read this appendix

The following conventions are used throughout this appendix:

- An underlined dollar sign (\$) indicates typing the ESCape key.
- The notation 'CTRL/x' indicates a control character, which you enter by holding down the CTRL key and typing the indicated character.
- Items in square brackets are optional.
- The letter 'n' indicates a numeric argument. You enter a numeric argument by typing \$[–] followed by a series of digits. The short sequence \$– is equivalent to \$–1. The first non-numeric keystroke you type after entering a numeric argument should be an editing function which takes an argument. Any other keystroke will cause the editor to discard the number. (It is also possible for a numeric argument to be an expression using standard operators. This is discussed later in the appendix.) Unless otherwise stated, the editor uses a default value of 1 if you omit an optional argument.
- The letter 'q' indicates any alphabetic Q-register name. If the syntax of the editor requires a Q-register name and you type any non-alphabetic character, VTEDIT sounds the terminal bell and ignores the command.

## VTEDIT—A keypad editing macro for TECO-11 and TECO-32

### K.1 Introduction

#### K.1.2 Invoking the Video Terminal Editor Macro

The Video Terminal Editor macro is stored in the file VTEDIT.TEC. For RSTS/E and VAX/VMS, the file is usually stored in the system library (SY:\$ for RSTS/E, SYS\$LIBRARY: for VAX/VMS). Other operating systems may store VTEDIT.TEC elsewhere; this appendix uses VAX/VMS for its examples. When you use the macro, it resides in Q-register I and uses all of the numeric Q-registers. (The macro saves these Q-registers when it starts and restores them when it exits, but they are unavailable from command mode while the macro is executing.) To load Q-register I with VTEDIT, execute the following TECO command:

```
EI$VTEDIT$$
```

The message 'VTEDIT.TEC loaded into Q-Reg I' is printed to confirm the load. Start the macro by typing the TECO command:

```
MI$$
```

These two commands cannot be in the same command sequence. You must execute the EI command before you can execute Q-register I.

If you use the DCL commands TECO or MAKE to invoke TECO, you can automatically load and start the VTEDIT macro by including the following switch after the DCL command name:

```
/VT
```

If you always want VTEDIT included, create a TECO start-up command file called TECO.INI in your account. This file can contain any TECO commands, and should end with the following commands:

```
ET&512"N 1+ ' 0
```

Then, whenever you use the TECO or MAKE DCL commands on a DEC\_CRT terminal, TECO automatically loads and starts VTEDIT. To suppress this loading, you must start editing with the command:

```
TECO/NOINIT
```

The model TECO.INI file does a more advanced set of checks as well as implementing some other random things (such as humorous messages based on the time of day). Of course, experimentation is encouraged.

#### K.1.3 Interpreting the Terminal Screen

While you are running the Video Terminal Editor macro, your terminal screen displays up to 24 lines of the text in TECO's text buffer. This 'window' into your buffer always includes the current line (the line containing the text pointer). The location of the text pointer within the buffer is indicated by the terminal's cursor. All characters that you type on the keyboard that are not commands to VTEDIT are immediately inserted into the text buffer at the current pointer position and are displayed on the screen.

Certain other special symbols can appear on the screen:

- A 'E<sub>F</sub>' symbol indicates the end of your text buffer and signifies that there is an implicit form feed at the end of the buffer.
- A diamond (♦) indicates the end of your text buffer if there is no implicit form feed there.
- A diamond (♦) immediately following one of the buffer end symbols above signifies that your input file is at end-of-file.

# VTEDIT—A keypad editing macro for TECO-11 and TECO-32

## K.1 Introduction

- A 'L<sub>F</sub>' symbol indicates that the text pointer is positioned between a carriage return and a line feed. This symbol appears only as the first symbol on a line, and it overlays the first character of the line. It allows you to tell the difference between the text pointer being at the start of a line of text and being just before the end of it.
- A 'N<sub>L</sub>' symbol indicates that the current line of text is too long to fit on the current line of the screen. If TECO's truncate flag bit (bit 8, value 256) is not set in the ET flag word (the normal case), the remainder of the text appears on the next line of the screen, preceded by the 'N<sub>L</sub>' and a space. If the truncate flag is set (via the TECO command ET#256ET), the remainder of the text is not displayed, but the special symbol appears at the end of the line to remind you that there are characters which cannot be displayed. You can control the number of characters that appear on each line by using the CTRL/V command.

If you are using a DEC\_CRT terminal in ANSI mode, VTEDIT will display any text between the current text pointer and the marked buffer location (see the Set Mark command) in reverse video to highlight the marked region.

A feature of the scope driver which allows you to view the exact contents of your text buffer is called 'see-all' mode. When see-all mode is turned on, all special characters and hidden spacing are visible on the screen as special graphic symbols:

TAB	appears as a or 'H <sub>T</sub> ' symbol. The spacing after the tab, which is normally blank space on the screen that does not correspond to any characters in the text, is shown as a series of centered dots (·).
LINE FEED	not immediately preceded by a RETURN appears as a 'L <sub>F</sub> ' symbol. If the line feed is in the middle of the line, the blank space at the beginning of the next line is filled (as for TAB). A LINE FEED immediately after a RETURN is not displayed with any special symbol (see RETURN below).
RETURN	followed by a line feed appears as a 'C <sub>R</sub> ' symbol. Any blank space before the end of the line is thus visible. A RETURN causing an overprint line (i.e., a RETURN not immediately followed by a LINE FEED) appears as a not-equal (≠) sign. The overprint characters are displayed on the next line instead of over (and thus clobbering) the original characters. This allows full viewing of all characters in overprint lines.
FORM FEED	appears as a 'F <sub>F</sub> ' symbol. No extra blank lines are shown. If the form feed is in the middle of the line, the blank space at the beginning of the next line is filled.
VERTICAL TAB	appears as a 'V <sub>T</sub> ' symbol. If the vertical tab is in the middle of the line, the blank space at the beginning of the next line is filled.
ESCAPE	appears as a Greek pi (π) to distinguish it from a dollar sign.
CTRL/x	appears as a plus-or-minus (±) sign followed by the character 'x' to distinguish it from a real uparrow.

## VTEDIT—A keypad editing macro for TECO-11 and TECO-32

### K.1 Introduction

#### K.1.4 Numeric Arguments

Simple numeric arguments consist of ESCape, an optional minus sign, and a digit string which is always interpreted in decimal. None of the keystrokes of a simple numeric argument echo on the terminal.

More advanced numeric arguments can be expressions using standard arithmetic operators. These numeric arguments also start with ESCape, an optional minus sign, and, then, a digit. If the first non-digit typed is one of the valid operators listed below, expression mode is entered. The top line of the display is erased and a line of the form:

Radix x, Expression arg1 op arg2

is displayed. The initial radix is 10, arg1 will be the initial digit string, and op will be the operator.

Further digit string typing will fill in arg2 until a non-digit is typed. If it is another valid operator, the current arg1 op arg2 expression will be evaluated, the result will replace arg1, arg2 will be set to zero, and op will be the typed operator.

If it is a radix change command, the radix is immediately changed. No evaluation of the expression occurs. The radix change is local to the current numeric expression; it is not subsequently remembered.

If it is an arg2 stand-in, the current arg2 is replaced by the specified value.

If it is RUBOUT, the current arg2 is reset to zero.

If it is CTRL/U, the entire expression is abandoned and the display is fixed up.

Else it is assumed to be an editing command. The current arg1 op arg2 will be evaluated and the result becomes the numeric argument to that command.

The valid numeric expression operators are:

Operator	Operation
+	Addition
–	Subtraction
*	Multiplication
/	Division
#	Logical .OR.
&	Logical .AND.
=	Evaluate current arg1 op arg2

The valid radix change commands are:

Command	Operation
^D	Change radix to decimal (base 10)
^O	Change radix to octal (base 8)
^X	Change radix to hexadecimal (base 16)

The valid arg2 stand-ins are:

Argument	Value
^.	Current value of the text buffer pointer
^Z	Current total number of characters in the text buffer
^^x	The value of character 'x'

## K.2 Editing Text

In addition to allowing you to enter text, the Video Terminal Editor macro has a large repertoire of powerful editing commands. You request an editing function by using the auxiliary keypad on the terminal or by typing control characters or control sequences on the standard keyboard. Some functions require you to type several keys.

When you start the Video Terminal Editor macro with the TECO command:

`-1MIS$`

it draws a keypad layout on the screen. This is a summary of the available editing functions. You can redisplay this keypad layout by using the ESCape H command (see the section on Miscellaneous Commands).

The following sections describe the editing functions available in VTEDIT. When an editing function is equivalent to a simple TECO command, the equivalent command appears after the description. The macro sounds the terminal bell when it detects an error (such as `$!`, which is an illegal sequence).

### K.2.1 Moving the Cursor

This section describes the commands which move the cursor around the screen.

## Set Mark

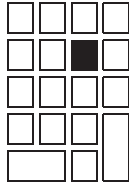
---

## Set Mark

Mark the current pointer position.

### Syntax

9 (keypad key)



### Description

Mark the current text buffer pointer position for use with later commands (see individual commands). To move the mark, simply type the 9 key at the new position. If you type the 9 key when the pointer is at the marked position, the mark is removed and the bell sounds. Thus, to remove a mark, type the 9 key until the bell rings. The mark is always removed when it is used.

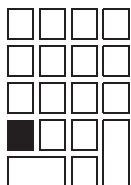
---

## Jump

Move the pointer to a specific position.

### Syntax

[n]1 (keypad key)



### Description

Move the text pointer to position n. If n is not given but a mark is set, jump to the mark. Otherwise, jump to the beginning of the text buffer.

In TECO: nJ

# Jump to End of Buffer

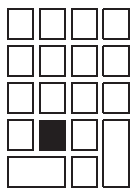
---

## Jump to End of Buffer

Move the pointer to End of Buffer.

### Syntax

2 (keypad key)



### Description

Move the text pointer to the end of the text buffer.

In TECO: ZJ



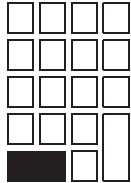
---

## Advance Line

Move the pointer forward over lines.

### Syntax

[n]0 (keypad key)



### Description

Advance the text pointer by n lines, leaving it at the beginning of a line.

In TECO: nL

## Back Line

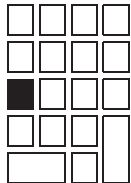
---

## Back Line

Move the pointer backwards over lines.

### Syntax

[n]4 (keypad key)



### Description

Back up the text pointer by n lines, leaving it at the beginning of a line.

In TECO: -nL

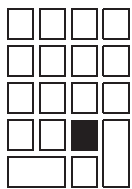
---

**Start of Line**

Move the pointer to the start of the line.

**Syntax**

3 (keypad key)



**Description**

Move the text pointer to the start of the current line.

In TECO: 0L

## End of Line

---

## End of Line

Move the pointer to the end of the line.

### Syntax

[n]BACKSPACE (special key or CTRL/H)

F11	BS	F13	F14
-----	----	-----	-----

### Description

Move the text pointer to the end of the nth line from the current line. If n is not given or is equal to 1, move to the end of the current line. The end of the line is just before a carriage return or form feed.

In TECO: nL 2R

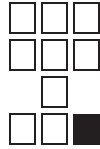
---

## Advance Character

Move the pointer forward.

### Syntax

[n]→ (right arrow keypad key)



### Description

Advance the text pointer by n characters.

In TECO: nC

## Back Character

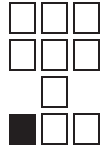
---

### Back Character

Move the pointer backward.

#### Syntax

[n]← (left arrow keypad key)



#### Description

Meaning: Back up the text pointer by n characters.

In TECO: nR

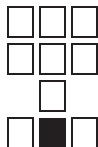
---

## Down in Column

Move the pointer down in the same column.

### Syntax

[n]↓ (down arrow keypad key)



### Description

Advance the text pointer by n lines, keeping it in the same column of the screen if possible. Column position is determined by counting characters and expanding TABs. If the target line is shorter than the position of the pointer at the start, the pointer is left at the end of the line. If the desired column in the target line is in the middle of a tab stop, the pointer is left at the TAB character. Repeatedly typing this key and the up arrow key preserves the target column, and repositions to it when possible.

## Up in Column

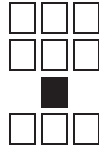
---

## Up in Column

Move the pointer up in the same column.

### Syntax

[n]↑ (up arrow keypad key)



```
□ □ □
□ □ □
■
□ □ □
```

### Description

Back up the text pointer by n lines, keeping it in the same column of the screen if possible. Column position is determined by counting characters and expanding TABs. If the target line is shorter than the position of the pointer at the start, the pointer is left at the end of the line. If the desired column in the target line is in the middle of a tab stop, the pointer is left at the TAB character. Repeatedly typing this key and the down arrow key preserves the target column, and repositions to it when possible.





---

## Advance Word

Move pointer across words.

### Syntax

[n]CTRL/F

 + 

### Description

Advance the text pointer by n words, leaving it at the first character of the nth word. Words are normally delimited by sequences of spaces, tabs, commas, carriage returns, and line feeds. See the section on selecting word delimiters.

## Back Word

---

### Back Word

Move pointer backwards across words.

#### Syntax

[n]CTRL/R

Ctrl + R

#### Description

Back up the text pointer by n words, leaving it at the first character of the nth previous word.

---

## Advance or Back Continuously

Advance or Back Continuously.

### Syntax

\$ followed by one of the six motion commands

### Description

\$ followed by one of the six motion commands for advance or back by line (0 or 4 keypad keys), character (left or right arrow keypad keys), or word (CTRL/F or CTRL/R)

Move the text pointer continuously by one unit (line, character, or word) in the selected direction, updating the screen each time the pointer moves. You can stop the motion by typing any non-keypad key. The character typed to stop the motion is otherwise ignored. Motion also stops (and the bell sounds) when the text pointer reaches the beginning or end of the buffer.

## Advance Screen Image

---

### Advance Screen Image

Move the pointer forward one full screen image.

#### Syntax

[n]\$↓ (ESCape, down arrow keypad key)

Esc + ↓

#### Description

Advance the text pointer the correct number of lines to move the current bottom line of the screen to just off the top of the screen. This allows you to 'page' through the text buffer. Overflow lines (lines too long for the screen) and overprint lines will cause slightly more or less than one screen's worth of data to be passed over.

In TECO: 24L

---

### Back Screen Image

Move the pointer backwards one full screen image.

#### Syntax

[n]\$↑ (ESCape, up arrow keypad key)

Esc + ↑

#### Description

Back up the text pointer the correct number of lines to move the current top line of the screen to just off the bottom of the screen. This allows you to 'page' through the text buffer. Overflow lines (lines too long for the screen) and overprint lines will cause slightly more or less than one screen's worth of data to be passed over.

In TECO: -24L

## Back Over Last Operation

---

### Back Over Last Operation

Back up the pointer to where it was before the last operation.

#### Syntax

\$6 (ESCape, 6 keypad key)

Esc + 6

#### Description

Back up the text pointer the correct number of characters position it to where it was prior to the last operation. For example, if you have just saved some text with the PF1 keypad key, then this would re-position you to the start of the saved text.

In TECO: ^SC

### **K.2.2 Searching Text**

This section describes the commands which search for the occurrence of specific text strings.

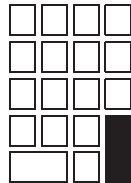
---

### Set Search Argument and Search Text Buffer

Search the current text buffer for string.

#### Syntax

[n]ENTER (keypad key)



#### Description

Get a search argument from the keyboard and search for the nth occurrence of it in the text buffer. When you type the ENTER key, VTEDIT erases the top two lines of the screen and prompts you for a search argument with 'Buffer search:'. Your previous search argument (from any kind of search) is initially displayed following the prompt. If you wish to reuse your previous argument, simply type either the keypad period (.) or a double ESCape. To edit your previous argument, type an initial RUBOUT. This deletes the last character of your argument, but retains the rest as if you had just entered it. Further editing and/or entry is now done in the normal way. Any other initial keystroke erases the entire previous search argument; you may now proceed to enter your new argument. The search argument may be any valid TECO search construct. Your search string appears on the screen as you type. You can use RUBOUT to edit your typing, and you can type CTRL/U to abandon the entry of the search string, abort the search command, and clean up the screen. When you are ready to perform the search, type either the keypad period (.) or a double ESCape. VTEDIT cleans up the screen, saves the search argument, and positions the pointer after the nth occurrence of the string within the text buffer. If n is negative, the search is done backwards in the buffer. If the string is found and there is no active mark set, the found string is shown in reverse video (on those terminals that support it) until the next keystroke. If the string is not found, VTEDIT moves the pointer to the start of the text buffer and sounds the bell.

In TECO: n:Stext\$





---

### Set Search Argument and Search through File

Search the rest of the file for string.

#### Syntax

[n]CTRL/N

 + 

#### Description

This command is similar to the previous one, but performs a through the whole file search. If you do not have an open output file, the terminal bell rings and the command is ignored. The prompt is 'File search:'. During the file searching, the cursor is positioned in the upper left corner of the screen. If the nth occurrence is not found, the bell sounds (the text buffer will be empty).

In TECO: n:Ntext\$

---

### Set Search Argument and Destructively Search File

Search the rest of the file for a string without output.

#### Syntax

[n]\$CTRL/N

Esc + Ctrl + N

#### Description

This command is similar to the previous one, but doesn't output passed over text. If you have an open output file and destructive mode (bit 1, value 2) in the TECO edit mode flag (ED) is not set, the terminal bell rings and the command is ignored. The prompt is 'Destructive file search:'. During the file searching, the cursor is positioned in the upper left corner of the screen. If the nth occurrence is not found, the bell sounds (the text buffer will be empty).

In TECO: n:\_text\$

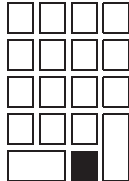
---

## Search Again

Perform the same search as before.

### Syntax

[n]. (keypad key)



### Description

Search in the text buffer for the nth occurrence of the search argument which was most recently saved via any search command. If n is negative, the search is done backwards in the buffer. If you have not yet used the ENTER command, VTEDIT searches for the string you most recently searched for from TECO before you started VTEDIT. If the string is not found, VTEDIT moves the pointer to the start of the text buffer and sounds the bell.

### **K.2.3 Deleting Text**

This section describes the commands for deleting text.

---

## Rub Out Character

Delete the character before the pointer.

### Syntax

[n]DEL



### Description

Delete n characters before the text pointer, starting with the character before the cursor.

In TECO: -nD

**Delete Character**

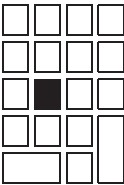
---

**Delete Character**

Delete the character after the pointer.

**Syntax**

[n]5 (keypad key)



**Description**

Delete n characters, starting with the character under the cursor.  
In TECO: nD

---

## Rub Out Word

Delete the word before the pointer.

### Syntax

[n]CTRL/B

Ctrl + B

### Description

Delete the text between the start of the nth previous word and the current text pointer.

## Delete Word

---

### Delete Word

Delete the word after the pointer.

#### Syntax

[n]\$CTRL/B

Esc + Ctrl + B

#### Description

Delete the text between the current text pointer and the start of the n+1st word beyond the pointer.



---

## Rub Out Line

Delete from start-of-line to pointer.

### Syntax

CTRL/U

Ctrl + U

### Description

Delete the text between the start of the current line and the text pointer.

In TECO: OK

## Delete Line

---

### Delete Line

Delete the rest of the line.

#### Syntax

[n]CTRL/K

Ctrl + K

#### Description

Delete n lines of text, starting at the current text pointer (the character under the cursor). If n is not given but a mark is set, delete the text between the mark and the current text pointer. Otherwise, delete one line.

In TECO: nK



---

## Delete Rest of Line

Delete from the pointer to end-of-line.

### Syntax

[n]CTRL/D

 + 

### Description

Delete the text between the current text pointer and the end of the nth line from the pointer. The carriage return and line feed at the end of the nth line (if any) are not deleted.

In TECO: nK 13I\$ 10I\$ 2R

## Delete or Restore Text

In TECO: .+^S,.K

#### **K.2.4 Inserting and Moving Text**

This section describes the commands that insert and move text.

## Open Line

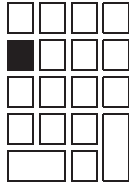
---

## Open Line

Insert an empty line into the text buffer.

### Syntax

[n]7 (keypad key)



### Description

Insert *n* return/line feed sequences after the current text pointer. The effect of this command is to move the rest of the text on the current line down to a new line.

In TECO: *n*<13I\$ 10I\$> 2\**n*R




---

## Get Q-register Contents

Load q-register into text buffer.

### Syntax

CTRL/Gq

 +  + 

### Description

Insert the contents of Q-register q in the text buffer. In this command, q can be any alphabetic or numeric Q-register name, or it can be either of the special names '\*' (file name buffer) and '\_' (search string buffer).

In TECO: Gq

## Load Q-register




---

### Load Q-register

Load q-register from text buffer.

#### Syntax

[n]CTRL/X[:]q

 +  + 

#### Description

Copy n lines from the text buffer into Q-register q. If n is not given and there is a mark set, copy the text between the mark and the current text pointer. Otherwise, copy one line. VTEDIT moves the text pointer to the end of the copied text. If the colon is present, append the text to the Q-register instead of copying it. You can not specify Q-register I.

---

#### Note

If you are using VAX/VMS, CTRL/X is intercepted by the terminal service as cancel all type ahead. You can use a combination of CTRL/T and CTRL/G to cut and restore text to Q-registers.

---

In TECO: nXq






---

## Load Q-register and Delete Text

Load q-register from text buffer and delete.

### Syntax

[n]CTRL/T[:]q

 +  + 

### Description

Similar to CTRL/X, but delete the copied text.

## Save Text

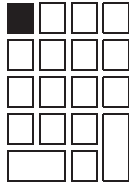
---

### Save Text

Save text into the cut buffer.

#### Syntax

[n]PF1 (keypad key)



#### Description

Copy *n* lines from the text buffer into the VTEDIT cut buffer. If *n* is not given and there is a mark set, copy the text between the mark and the current text pointer. Otherwise, copy one line. VTEDIT moves the text pointer to the end of the copied text. If you repeatedly type the PF1 key with no intervening keystrokes, VTEDIT appends successive lines of text to the cut buffer. You can delete the text which was cut by typing the 6 keypad key.

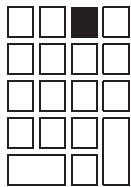
---

Paste Text

Insert cut buffer into text.

Syntax

PF3 (keypad key)



Description

Insert the contents of the cut buffer into the text buffer.

---

### Insert ASCII Value

Insert literal character in text buffer.

#### Syntax

[n]CTRL^

Ctrl + N

#### Description

Accept a character from the terminal and insert into the text buffer the string representing the ASCII value of that character. If n is present, insert the string representing n and do not accept an extra character. VTEDIT inserts the number using the radix that was set when you started the macro. You can change this radix by executing a TECO command with the PF2 key or CTRL/E function.

In TECO: ^^x\ (where x is the next character you type)

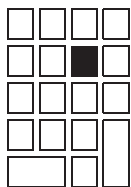
---

## Insert Special Character

Insert character into the text buffer.

### Syntax

n9 (keypad key)



### Description

Insert into the text buffer the character whose ASCII value is n.

In TECO: nI\$

## **VTEDIT—A keypad editing macro for TECO-11 and TECO-32**

### **Insert Special Character**

#### **K.2.5 Advancing through the Input File**

This section describes the commands that are used to read the input file.

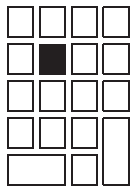
---

Advance Page

Move pointer across pages of text.

Syntax

[n]8 (keypad key)



Description

Go forward n pages in the file(s) being edited. If you do not have an open output file, the terminal bell sounds and the command is ignored.

In TECO: nP

## Append Page or Line



---

### Append Page or Line

Append a page or a line of text to the buffer.

#### Syntax

[n]CTRL/A

 + 

#### Description

Append the next *n* pages from the input file to the current text buffer. If *n* is 0, append only one line from the file. If *n* is negative (e.g., S-CTRL/A), keep appending pages from the input file until end-of-file. If there are form feeds separating the text in the file, VTEDIT inserts form feeds in the text buffer.

In TECO: *n* < ^E" T 12I\$ ' A >



---

## Append Page or Line with no Form Feeds

Append a page or a line of text and discard form feeds.

### Syntax

[n]SCTRL/A

Esc + Ctrl + A

### Description

Append the next n pages from the input file to the current text buffer, discarding any form feeds that appear in the input file. If n is negative (e.g., S-SCTRL/A), keep appending pages from the input file until end-of-file. If n is 0, append only one line from the file.

In TECO: n< A >

## Yank Page

---

## Yank Page

Get a new page from the input file.

### Syntax

[n]\$Y

Esc + Y

### Description

Yank n pages, leaving the nth page in the text buffer. If you have an open output file and destructive mode (bit 1, value 2) in TECO's edit mode flag (ED) is not set, the terminal bell rings and the command is ignored.

In TECO: n< Y >

---

## Inhibit Yank Protection

### Syntax

\$\$Y

Esc + ␣ + Esc + Y

### Description

This command inhibits yank protection; it sets destructive mode (bit 1, value 2) in TECO's edit mode flag (ED).

In TECO: 0,2ED

## Enable Yank Protection

---

## Enable Yank Protection

### Syntax

\$0\$Y

[Esc] + 0 + [Esc] + Y

### Description

This command enables yank protection; it clears destructive mode (bit 1, value 2) in TECO's edit mode flag (ED).

In TECO: 2,0ED

### **K.2.6 Executing TECO commands**

This section describes the commands that enable you to execute TECO commands under control of the VTEDIT macro.

## Execute Q-register Macro

---

### Execute Q-register Macro

Execute a macro contained in a q-register.

#### Syntax

[n]CTRL/Eq

 +  + 

#### Description

Execute the macro in Q-register q. If n is present, use its value as an argument to the macro; if not, execute the macro with no argument. VTEDIT moves the cursor to the top left corner of the screen while the macro is being executed.

In TECO: nMq or Mq

---

## Repeat Q-register Macro

Repeatedly execute a macro in a q-register.

### Syntax

[n]CTRL/Eq

Esc + Ctrl + E + q

### Description

Execute the macro in Q-register q in an iteration loop. If the value of n is zero, the iteration is infinite. Otherwise, n is the iteration count for the loop. If n is not given, this command is the same as CTRL/Eq.

In TECO: n< Mq > or < Mq >

## Set TECO Command and Execute

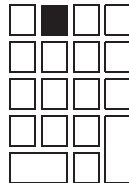
---

### Set TECO Command and Execute

Load a TECO command and execute it.

#### Syntax

[n]PF2 (keypad key)



#### Description

Get a TECO command from the keyboard and execute it. When you type the PF2 keypad key, VTEDIT erases the top two lines of the screen and prompts you for a command. Your previous command string is initially displayed following the 'Command:' prompt. If you wish to reuse your previous command, simply type either the keypad period (.) or a double ESCape. To edit your previous command, type an initial RUBOUT. This deletes the last character of your command, but retains the rest as if you had just entered it. Further editing and/or entry is now done in the normal way. Any other initial keystroke erases the entire previous command string; you may now proceed to enter your new command. You may enter any valid TECO command string. Your command string appears on the screen as you type. You can use RUBOUT to edit your typing, and you can type CTRL/U to abandon the entry of the command, bypass command execution, and clean up the screen. When you are ready to execute the command, type the keypad period (.) or a double ESCape. VTEDIT cleans up the screen, saves the command string, and executes the command. VTEDIT moves the cursor to the top left corner of the screen while the command is being executed.

If n is present, its value is used as an argument to the TECO command string.

In TECO:

^U9command\$ nM9

^U9command\$ M9

---

#### Note

VTEDIT truly stores the TECO command in Q-register 9. A subsequent CTRL/G 9 command will insert the command into the text buffer. To save the command in an alphabetic Q-register for later usage, You can use the following keystroke sequence:

9	Keypad key 9 to set a mark
CTRL/G 9	Get TECO command from Q-register 9
CTRL/T q	Take command away to Q-register q

---



---

## Re-execute TECO Command

Execute stored TECO command.

### Syntax

[n]\$\$

Esc + Esc

### Description

Execute the TECO command most recently set with the PF2 keypad key. VTEDIT moves the cursor to the top left corner of the screen while the command is being executed.

If n is present, its value is used as an argument to the TECO command.

## Set TECO Command and Repeat Execution

---

### Set TECO Command and Repeat Execution

Get a TECO command and repeatedly execute it.

#### Syntax

[n]\$PF2 (keypad key)

Esc + PF2

#### Description

A TECO command is gotten from the keyboard just like the PF2 command. If n is present, the TECO command is executed n times. Else the TECO command is executed in an infinite iteration.

In TECO:

`^U9command$ n< M9 >`

`^U9command$ < M9 >`

---

## Start/End Learning Mode

Enter a command sequence and remember it.

### Syntax

\$L

Esc + ␣

### Description

VTEDIT can 'learn' a series of keystrokes and (re-)execute them upon request. An initial \$L puts VTEDIT into learning mode. Every subsequent keystroke is remembered by VTEDIT until another \$L is typed. To cancel a learned sequence or to ensure that you are out of learning mode, type \$L until the terminal bell sounds. The bell signals that the null learning sequence (i.e., \$L\$L) has been entered.

## Execute Learned Sequence

---

### Execute Learned Sequence

Execute a previously remembered command sequence.

#### Syntax

[n]\$X

Esc + X

#### Description

This command causes VTEDIT to (re-)execute the previously learned keystroke sequence. If you do not have any learning sequence remembered, the terminal bell rings. The optional numeric argument is the number of times to execute the learned keystroke sequence.

### **K.2.7 Controlling the Video Terminal**

This section describes the commands that are used to control the display.

## Repaint Screen

---

### Repaint Screen

Repaint.

#### Syntax

CTRL/W

Ctrl + W

#### Description

Repaint the screen. This command is useful if the display on the screen is destroyed by a broadcast from the system manager. This command also restores special keypad mode on the editing keypad, and can be used if the terminal was powered down.

VTEDIT requests your terminal to send its terminal id as part of this command. You should, therefore, wait until the screen stops updating/flickering before further typing. As part of the screen repaint, a series of ESCape sequences are sent to the terminal.

---

## Set Cursor Line

Set the position of the line where the cursor remains.

### Syntax

nCTRL/W

n + Ctrl + W

### Description

The cursor line is set to n. The scope driver will try to keep the cursor on the nth line of the screen. The initial value of this cursor line is 2/3rd's of the way down the screen (16 for 24 line screens).

An effective CTRL/W command is executed after setting the cursor line; please read the notes above.

## Toggle See-all Display Mode

---

### Toggle See-all Display Mode

Set screen mode so that all characters are displayed.

#### Syntax

CTRL/V

Ctrl + V

#### Description

This command turns on the scope driver see-all mode, or turns it off if it is currently on.



---

## Set Terminal Characteristics

Set the characteristics of the terminal.

### Syntax

nCTRL/V

 +  + 

### Description

Change the display characteristics of the terminal. VTEDIT requests your terminal to send its terminal id as part of this command. You should, therefore, wait until the screen stops updating/flickering before further typing. If you have selected some illegal terminal characteristic, VTEDIT should catch it and automatically establish something that will work.

If n has the value 0, VTEDIT tells the scope driver that your terminal is a VT52. Use this command only if TECO made a wrong guess when it started (which shouldn't happen) or if you detach and change terminal types during an editing session. The value 2 means that the terminal is a DEC\_CRT, but should be controlled in VT52 compatibility mode. The value 4 means that the terminal is a DEC\_CRT and should be controlled with ANSI standard escape sequences. This is the optimal setting for DEC\_CRT terminals, and must be used if you want to display 132 columns or to display the marked region in reverse video. When you change modes on a DEC\_CRT, the VTEDIT sends the proper escape sequence to the terminal to change its operation.

If n has an even value from 10 to 132, that value is used as the new width for the display. You must be in DEC\_CRT ANSI mode to set a width greater than 80. You can set a width narrower than terminal can display. Setting a width wider than your terminal can handle may cause unpredictable results. Note that the scope driver reserves the last two characters of each line, and therefore can display only n-2 characters if the width is set to n. The default width is 80 characters when you start VTEDIT.

If n has a negative value from 10 to 24, it sets the height of the display. You should not use this command, unless you have a DEC\_CRT in ANSI mode which has the capability to change the hardware scrolling region.

## Enter/Exit Screen Holding Mode

---

## Enter/Exit Screen Holding Mode

Toggle hold screen mode.

### Syntax

[n]\$CTRL/V

Esc + Ctrl + V

### Description

This command toggles "holding screen" mode. In normal mode (the default), VTEDIT attempts to keep the cursor positioned on the line specified by the nCTRL/W command. Any advance or back line command will scroll the screen one line thus keeping the cursor in the same physical location on the screen. In holding screen mode, VTEDIT will only scroll the screen when the cursor position would have been off the screen. For example, an advance line from the last line on the screen or a back line from the top line on the screen will cause the screen to scroll and the cursor to be positioned at its default location, the line specified by nCTRL/W. This mode is useful for terminals without reverse scrolling capabilities and/or for operation over a low speed communication lines.

If n is present and has the value 0, holding screen mode is unconditionally turned off; if n is -1, holding screen mode is unconditionally turned on.

If n is positive, holding screen mode is turned on with the cursor limit set to n. VTEDIT will only scroll the screen if the cursor position would have been within the top or bottom n lines on the screen.

---

#### Note

---

This feature only works on VT52 terminals and should not be used on DEC\_CRT terminals.

---

### **K.2.8 Selecting Word Delimiters**

Several of the VTEDIT editing commands use the idea of a 'word' in the text buffer. Normally, words are delimited by sequences of spaces, tabs, commas, carriage returns, and line feeds. If you have a special editing application, you may require a different definition of a word. The following commands allow you to change that definition.

## Use Standard Words

---

## Use Standard Words

Set the standard word delimiting characters.

### Syntax

\$T

Esc + T

### Description

Restore the set of word delimiters to the standard set: space, tab, comma, carriage return, line feed.

---

## Use BASIC-PLUS Words

Use the BASIC-PLUS word delimiting characters.

### Syntax

\$O\$T

[Esc] + [O] + [Esc] + [T]

### Description

Use a set of word delimiters appropriate for editing BASIC-PLUS code. This command appends the following special characters to the set of word delimiters:

( ) + - \* ^ \ ! = : < > /

## Clear Delimiter Set

---

## Clear Delimiter Set

Clear word delimiting characters.

### Syntax

\$ \$T

[Esc] + [ ] + [ESC] + [ ]

### Description

Clear out the word delimiter set. You must follow this command by adding one or more word delimiters (see below).

---

## Add Word Delimiter

Add characters to the word delimiters.

### Syntax

n\$T

+  Esc +

### Description

Add the character whose ASCII value is n to the set of word delimiters. The value of n must be between 1 and 127.

## Get Next Word

---

### Get Next Word

Move word(s) from surrounding lines to current line.

#### Syntax

[n]\$N

Esc + N

#### Description

Move the next n words to the current line. If n is positive, the cursor should be at the end of a line; the first n words of the next line(s) are moved to the end of the current line. If n is negative, the cursor should be at the beginning of a line; the last n words of the previous line(s) are moved to the beginning of the current line.



### **K.2.9 Exiting**

This section describes the commands used to exit from VTEDIT and TECO.

## Exit from VTEDIT

---

## Exit from VTEDIT

### Syntax

CTRL/Z or CTRL/C

Ctrl + Z or Ctrl + C

### Description

Exit from the Video Terminal Editor macro to TECO's asterisk prompt.

---

## Exit from TECO

Exit from TECO.

### Syntax

\$CTRL/Z

Esc + Ctrl + Z

### Description

Exit from the Video Terminal Editor macro and from TECO, copying the input file to the output file and closing both files. This command turns off special keypad mode and moves the cursor to the bottom of the screen.

In TECO: EX

## Exit from TECO Without Preserving Output

---

### Exit from TECO Without Preserving Output

Exit from TECO and undo all changes.

#### Syntax

\$-CTRL/Z

Esc + ␣ + Ctrl + Z

#### Description

Delete any output file which may be open, undo any edit backup in effect, and exit from TECO. This command turns off special keypad mode and moves the cursor to the bottom of the screen.

In TECO: HK EK EX

### **K.2.10 Miscellaneous Commands**

This section describes the rest of the commands.

## Find Front of File

---

## Find Front of File

Go back to the beginning of the file.

### Syntax

\$F

Esc + F

### Description

The current file open for editing is closed out and, then, re-opened. You are left positioned at the first page of the file. If you have no current output file the terminal bell sounds and the command is ignored.

In TECO: EC EBfile\$ Y

---

### Worry About Operating System Crashing

Close the edit session and re-open at the current point.

#### Syntax

\$1\$F

[Esc] + [1] + [Esc] + [F]

#### Description

Close out the file currently open for editing, re-open it, and re-find your current editing position. Similiar to the command above, but automatically re-positions you to where you left off editing. This command is useful if your operating system tends to crash...

## Exit from TECO Marking Position

---

## Exit from TECO Marking Position

Exit TECO but remember where you are.

### Syntax

\$O\$F

[Esc] + [O] + [Esc] + [F]

### Description

Close out the file currently open for editing marking the current editing position. TECO's memory file is updated to reflect the file currently being edited and the /FIND switch is appended to the file specification. This command is used in conjunction with the command below.



---

## Find Marked Position

Find the point where you stopped editing.

### Syntax

\$\$F

[Esc] + [ ] + [Esc] + [F]

### Description

(Re-)find marked editing position. The editing position marked by some previous \$\$F command is re-found.

## Redisplay Keypad Layout

---

### Redisplay Keypad Layout

Help for keypad.

#### Syntax

\$H

Esc + H

#### Description

Clear the screen and display the help text for the Video Terminal Editing macro. Type any non-keypad character to redisplay the text window and resume editing. VTEDIT does not insert that character into the text.

---

## Set Left Margin

Set the position of the left margin.

### Syntax

[n]\$M

Esc + M

### Description

Set the left margin at column n. If you have set a left margin, the VTEDIT automatically inserts enough tabs and spaces to get the cursor to the margin whenever you type a carriage return or an open line command (7 key on keypad). If n is negative, VTEDIT sets the margin to -n and uses only spaces for filling. If n is not present, the margin is set at the current column. If there is any text on the line when this you give this command, VTEDIT shifts it to the new margin.



## Index to TECO commands and special characters

**Table L–1 Index to TECO commands**

Char	Meaning	Section
NULL	Discarded on input; Ignored in command	5.18.5
^A	Output message to terminal	5.4
^B	Current date	5.11
^C	Stop execution	4.1, 5.1.3, 5.13
^D	Set radix to decimal	5.10
^E	Form Feed flag	5.11
^E<n>	(Match char) Match ASCII code n	5.8
^EA	(Match char) Match alphabetics	5.8
^EB	(Match char) Match separator character	5.8
^EC	(Match char) Match Symbol Constituent	5.8
^ED	(Match char) Match numerics	5.8
^EGq	(Match char) Match contents of Q-reg	5.8
^EL	(Match char) Match line terminators	5.8
^EMx	(Match char) Match any number of x	5.8
^EQq	(String char) Use contents of Q-reg q	5.8
^ER	(Match char) Match alphanumerics	5.8
^ES	(Match char) Match non-null space/tab	5.8
^EUq	(String char) Use ASCII code in Q-reg	5.8
^EV	(Match char) Match lower case alphabetic	5.8
^EW	(Match char) Match upper case alphabetic	5.8
^EX	(Match char) Match any character	5.8
^E[]	(Match char) Match one of list	5.8
^F	Contents of console switch register	5.11
n^F	Return terminal number of job n	5.11
^G^G	Kill command string	4.1
^G<sp>	Retype current command line	4.1
^G*	Retype current command input	4.1
^H	Current time of day	5.11

(continued on next page)

## Index to TECO commands and special characters

**Table L-1 (Cont.) Index to TECO commands**

Char	Meaning	Section
BS	Back up and type one line	1.6.1, 4.1
TAB	Insert tab and text	5.6
LF	Line terminator; Ignored in commands	1.2, 3.3, 5.18
LF	Advance and type one line	1.6.1, 4.2
VT	Line terminator; Not a TECO command	3.3
FF	Page terminator; Output Form Feed	1.2, 3.3
CR	End input line; Ignored in commands	1.2, 4.1, 5.18
^N	End of file flag	5.11
^Nx	(Match char) Match all but x	5.8
^O	Set radix to octal	5.10
^O	Kill terminal output	5.4
^P	Not a TECO command	
^Q	Resume terminal output	5.4
^Q	Convert line arg into character arg	C.1, C.7
^Qx	(String char) Use x literally	5.8
^R	Value of current radix	5.11
n^R	Set radix to n	5.11
^Rx	(String char) Use x literally	5.8
^S	-(length) of last referenced string	5.11
^S	Suspend terminal output	5.4
^S	Save last command string	C.6
^S	(Match char) Match separator character	5.8
^T	ASCII value of next character typed	5.11
^T	Read and decode next keystroke typed	5.11
n^T	Type ASCII character of value n	5.4
n:^T	Output binary byte of value n	5.4
^U	Kill command line	4.1
^Uq	Put string into Q-register q	5.9
:^Uq	Append string to Q-register q	5.9
n^Uq	Put ASCII char "n" into Q-register q	5.9
n:^Uq	Append ASCII char "n" to Q-register q	5.9
^V	Enable lower case conversion	5.18.7
^Vx	(String char) Force x to lower case	5.8
^W	Enable upper case conversion	5.18.7
^W	Refresh scrolled display	4.2
^Wx	(String char) Force x to upper case	5.8
^X	Search mode flag	5.16
^X	(Match char) Match any character	5.8

(continued on next page)

## Index to TECO commands and special characters

**Table L-1 (Cont.) Index to TECO commands**

Char	Meaning	Section
^Y	Equivalent to ".+^S,."	5.11
^Z	Size of text in all Q-registers	5.11
^Z^Z^Z	Immediate exit from TECO	4.1
ESC	String and command terminator	1.1, 4.1, 5.13
ESC	String and command terminator	5.18.5
^[	String and command terminator	5.18.5
^\ ^]	Not a TECO command	
^^x	ASCII value of x	5.11
^_	Ones complement (logical NOT)	5.10
SP	Ignored in commands	5.18
!	Define label	5.13
"	Start conditional	5.14
n"<	Test for less than zero	5.14
n"=	Test for equal to zero	5.14
n">	Test for greater than zero	5.14
n"A	Test for alphabetic	5.14
n"C	Test for symbol constituent	5.14
n"D	Test for numeric	5.14
n"E	Test for equal to zero	5.14
n"F	Test for false	5.14
n"G	Test for greater than zero	5.14
n"L	Test for less than zero	5.14
n"N	Test for not equal to zero	5.14
n"R	Test for alphanumeric	5.14
n"S	Test for successful	5.14
n"T	Test for true	5.14
n"U	Test for unsuccessful	5.14
n"V	Test for lower case	5.14
n"W	Test for upper case	5.14
#	Logical OR	5.10
\$	Separate TECO commands	5.18.5
n%q	Add n to Q-register q and return result	5.9
&	Logical AND	5.10
'	End conditional	5.14
(	Expression grouping	5.10
)	Expression grouping	5.10
*	Multiplication	5.10

(continued on next page)

## Index to TECO commands and special characters

**Table L-1 (Cont.) Index to TECO commands**

Char	Meaning	Section
*q	Save last command in Q-register q	4.2
+	Addition	5.10
,	Argument separator	5.10
-	Subtraction or negation	5.10
.	Current pointer position	5.11
/	Division	5.10
/	Type detailed explanation of error	4.2
0-9	Digit	
:	Modify next command	3.2.3
;	Exit iteration on search failure	5.13
n;	Exit iteration if n is positive	5.13
::	Exit iteration on search success	5.13
n::	Exit iteration if n is negative	5.13
n<	Iterate n times	5.12
=	Type in decimal	5.10
==	Type in octal	5.10
===	Type in hexadecimal	5.10
:=	Type in decimal, no carriage return	5.10
:= =	Type in octal, no carriage return	5.10
:= ==	Type in hexadecimal, no carriage return	5.10
>	End iteration	5.12
?	Toggle trace mode	5.18.4
?	Type out command string in error	4.2
@	Modify next text argument	3.2.2
A	Append to buffer	5.2
nA	ASCII value of char in buffer	5.11
n:A	Append n lines to buffer	5.2
B	0 - beginning of buffer	5.11
nC	Advance n characters	5.3
nD	Delete n characters	5.5
m,nD	Delete between m and n	5.5
DELIM	Character typed as surrogate for ESCape	1.1, 3.2.2, 4.1, 5.16
DELIM	Advance and type one line	1.6.1, 4.2
E%q	Write Q-register q into a file	5.1.6
EA	Select secondary output stream	5.1.4
EB	Open input and output	5.1.1
EC	Close out (copy in to out and close)	5.1.3

(continued on next page)



## Index to TECO commands and special characters

**Table L–1 (Cont.) Index to TECO commands**

Char	Meaning	Section
ED	Edit mode flag	5.16
EE	ESCape surrogate character	5.16
EF	Close output file	5.1.3
EG	Close out and exit with command	5.1.3
:EG	Execute operating system function	5.1.3, C.3, C.4, C.5
EH	Help level flag	5.16
EI	Open indirect command file	5.1.4
m,nEJ	Set environment characteristics	5.15
nEJ	Return environment characteristics	5.15
EK	Kill output file	5.1.3
EL	Open log file	5.1.3
nEM	Position magtape	C.7
EN	Wildcard lookup	5.1.5
EO	Version of TECO	5.11
nEO	Set TECO to function as version n	5.16
EP	Select secondary input stream	5.1.4
EQq	Read file into Q-register q	5.1.6
ER	Open input file	5.1.1, 5.1.4
ES	Search verification flag	5.16
ET	Type out control flag	5.16
EU	Case flagging flag	5.16
EV	Edit verify flag	5.16
EW	Open output file	5.1.1, 5.1.4
EX	Close out and exit	5.1.3
EY	Read without yank protection	5.2
EZ	Zero output tape	5.1.3
nE_	Search without yank protection	5.7
F'	Flow to end of conditional	5.13
F<	Flow to start of iteration	5.13
F>	Flow to end of iteration	5.13
F	Flow to ELSE part of conditional	5.13
m,nFB	Search between locations m and n	5.7
nFB	Search, bounded by n lines	5.7
m,nFC	Search and replace between m and n	5.7
nFC	Search and replace over n lines	5.7
nFD	Search and delete string	5.5, 5.7
nFK	Search and delete intervening text	5.5, 5.7

(continued on next page)

## Index to TECO commands and special characters

**Table L-1 (Cont.) Index to TECO commands**

Char	Meaning	Section
nFN	Global string replace	5.7
FR	Replace last string	5.6
nFS	Local string replace	5.7
nF_	Destructive search and replace	5.7
Gq	Get string from Q-register q into buf	5.9
G*	Get last filespec string into buffer	C.1
:Gq	Type Q-register q on terminal	5.9
G_	Get last search string into buffer	C.1
H	Equivalent to "B,Z"	5.11
I	Insert text	5.6
nI	Insert ASCII character "n"	5.6
nJ	Move pointer to "n"	5.3
nK	Kill n lines	5.5
m,nK	Delete between m and n	5.5
nL	Advance n lines	5.3
Mq	Execute string in Q-register q	5.9
nN	Global search	5.7
O	Go to label	5.13
nO	Computed goto	5.13
nP	Advance or back up n pages	5.2
m,nP	Write out chars m to n	5.2
nPW	Write buffer n times	5.2
m,nPW	Write out chars m to n	5.2
Qq	Number in Q-register q	5.9, 5.11
:Qq	Size of text in Q-register q	5.9, 5.11
nR	Back up n characters	5.3
nS	Local search	5.7
m,nS	Search for nth occurrence within m characters	C.1
::S	Compare string	5.7
SEEALL	"View All" characters display mode	3.1
nT	Type n lines	5.4
m,nT	Type from m to n	5.4
nUq	Put number n into Q-register q	5.9
nV	Type n current lines	5.4
m,nV	Type m before & n after current line	5.4
W	Scope "WATCH"	5.17.1
n:W	Return scope characteristics	5.17.1, C.1
m,n:W	Set scope characteristics	5.17.1

(continued on next page)

## Index to TECO commands and special characters

**Table L-1 (Cont.) Index to TECO commands**

Char	Meaning	Section
nXq	Put n lines into Q-register q	5.9
m,nXq	Put characters m to n into Q-register	5.9
n:Xq	Append n lines to Q-register q	5.9
m,n:Xq	Append characters m to n into Q-register	5.9
Y	Read into buffer	5.2
Z	End of buffer value	5.11
[q	Q-register push	5.9
\	Value of digit string in buffer	5.10, 5.11
n\	Convert n to digits in buffer	5.10
]q	Q-register pop	5.9
n_	Global search without output	5.7
`	ESCape surrogate timeout	1.1, 3.2.2, 4.1, 5.16
a-z	Treated the same as upper case A-Z	
{	Not a TECO command	
	Start ELSE part of conditional	5.14
}	Not a TECO command	
~	Not a TECO command	
DEL	Delete last character typed in	4.1



---

# Glossary of Obscure TECO Terms

## **Accent grave**

The ` character. Echoed for TECO's text and command string delimiter when an ESCape surrogate is in effect. Actually set as the ESCape surrogate by the 8192 ET bit.

## **Abort-on-error bit**

The 128's bit of the ET flag. If this bit is set, then TECO will abort execution on encountering an error (after printing the error message) and control returns to the operating system. This bit is initially set when TECO starts up, but is reset whenever TECO issues its prompt.

## **<ALTMODE>**

One of several characters that TECO treats specially for use as a delimiter. Known as ESCAPE in more recent times, but traditional TECO users will still go on ending their command strings with "ALT" "ALT".

## **Anchored search**

A search (S) or search and replace (FS) command that is preceded by a ::. This indicates that the search must match the characters immediately following the current pointer position. If these characters do not match, no further searching is performed and the text buffer pointer does not move.

## **Argument pair**

A command of the form m,n where m and n are numbers (or TECO commands that return numbers). When used before a command that normally acts on lines, the argument pair causes the command to work on characters. The characters specified by this argument pair is the set of characters between pointer positions m and n. The number m should normally be less than or equal to n.

## **ASCII code**

The American Standard Code for Information Interchange. The code used by TECO to represent characters internally. Consult Appendix A for details.

## **@-sign modified command**

A command that is preceded by an @-sign modifier.

## **@-sign modifier**

An at-sign (@) that proceeds a command. It indicates to TECO that the string argument to the command is to delimited on both sides by a user-supplied delimiter rather than to be delimited only at the end by a TECO-designated delimiter (normally <ESCAPE>).

**Automatic refresh**

A refresh of the buffer display done when scrolling is active (when 7:W is non-zero) just before TECO issues its asterisk prompt. Automatic refresh can be disabled by the 128 ED bit.

**Automatic type-out**

The feature of TECO that causes lines of text to be automatically typed out. The ES flag may be used to control the automatic type out of lines after search commands, and the EV flag may be used to cause automatic type out of lines after command execution.

**Auto-trace mode**

A mode that TECO goes into when the 4's bit of the EH (Help-level) flag is set. In this mode, TECO will automatically print out the erroneous command string after an error occurs.

**Backup protection**

The process of preserving the user's original file (as a backup) when editing that file with the EB (Edit with Backup) command.

**Backwards searches**

A search that proceeds in the backwards direction. If the string being looked for does not occur at the current pointer position, the pointer position is moved back one character and the search is tried again. This continues until the string is found or until the boundary of the search has been reached or until the beginning of the current text buffer has been reached. Backward searches are initiated by using a negative argument to a search command or by using an argument pair m,n with m greater than n to an FB or FC command.

**Bounded searches**

A search command that requires searching only a portion of the text buffer. Of particular importance is the case where you only want to search the current line for a given string. Bounded searches are accomplished using the FB command.

**Case flagging**

A mode of TECO wherein, on type out, it will flag alphabetic characters (in either upper or lower case) by preceding them with a single quote. Lower case flagging is particularly useful on terminals that do not display lower case. Case flagging is controlled by EU, the case flag. Setting EU to 0 sets lower case flagging mode; setting EU to 1 sets upper case flagging mode, and setting EU to -1 removes all case flagging.

**Character-oriented editor**

An editor that allows modification of single characters, so that if just one character of a line is wrong, the entire line does not have to be retyped. TECO is a character-oriented editor (although it has a number of facilities for dealing with text lines, too).

**Colon-modified command**

A command that is preceded by a colon (:) modifier.

**Colon-modifier**

A colon preceding a TECO command, used to indicate that the action of the command is to change. Frequently indicates that the command is to return a value (-1 if the command succeeded, 0 if the command failed).

**Command line**

The current line of the command string that is being typed into TECO.

**Command string scope editing**

The feature of TECO that is enabled when editing is performed on a CRT terminal. In this mode, typing the immediate action <DELETE> command causes the character deleted from the command string to physically disappear from the screen. Other commands, such as <CTRL/U> also behave differently, taking best advantage of the properties of a video terminal.

**Command string**

The string of TECO commands that is currently being typed into TECO, or is currently being executed by TECO.

**Comment**

An informative message used within a TECO program, to make the code more readable. The comment explains the meaning and purpose of the associated TECO commands. The comment is enclosed in exclamation marks.

**Compile-class command**

A set of operating systems commands (such as COMPILE) that causes compilation (translation) of a source file written in a computer language into machine instructions. The EG` command is useful to finish an editing session and re-execute the last compile-class command (normally a command that compiles the file that was just edited).

**Conditional**

A TECO language construct used to specify code to be conditionally executed depending upon the results of some test. The most general form of the TECO conditional is `n"X <then-clause> | <else-clause> '` which tests the number `n` using condition `X` (See Section 5.13 for details). The commands specified by the `<then-clause>` are executed if the condition succeeds, otherwise the commands specified by the `<else-clause>` are executed.

**Control-character**

An ASCII character whose octal code is in the range 0-37. Usually denoted in this manual by `<CTRL/X>` where `X` is the character whose ASCII code is 100 (octal) greater than the ASCII code for the control character being represented. TECO displays such a character as `^X` (Caret-X) except for `<TAB>`, `<LF>`, `<VT>`, `<FF>`, and `<CR>` which have their normal display, and `<ESC>` which is displayed as `$` (dollar sign) or, if a user-designated ESCAPE surrogate is in effect, as ``` (accent grave). Anytime a single control-character, `<CTRL/X>` is a valid TECO command, the two-character sequence, `^X`, may be used instead.

**<CTRL/C> trapping**

A mode of operation wherein a TECO macro will regain control (rather than TECO) when <CTRL/C> or <CTRL/C><CTRL/C> is typed on the user's terminal to abort execution of the current command string. <CTRL/C> trapping is enabled by setting the high-order bit of the ET flag.

**Current character**

The character immediately following the current text buffer pointer position.

**Cursor**

A visible pattern on a CRT terminal (usually blinking) that would specify to a true scope editor (like VTEDIT) the current location of the text buffer pointer, or which would specify the location where subsequently typed characters would be displayed.

**<DELIM>**

Terminology in this manual for an ESCape typed at the terminal and passed to TECO. On newer terminals, there may be no ESCape key. Some versions of TECO provide for an "ESCape surrogate", a user-designated key which is to cause transmission of an ESCape to TECO. The term <DELIM> is used to indicate an ESCape transmitted to TECO whether it was generated by the designated "surrogate" key or by a "real" ESCape key. An ESCape types out as ` when a surrogate is active, but as \$ when no surrogate is active.

**Destructive search**

A form of global search in which pages of text that are passed over (because they did not contain the string being searched for) are discarded rather than written to the output file. Destructive searches are initiated in TECO via use of the \_ command.

**Display editor**

A true display editor is one which makes efficient use of a CRT terminal or display scope. Such an editor maintains a "window" into the text being edited. As characters are typed on the terminal, these characters immediately are entered into the text buffer and the window is immediately updated to reflect this change. With the help of some macro support, TECO can be made to be a true display editor. A more modest use of a CRT terminal (and which requires no macro support) is scrolling, q.v.

**Dot**

A mnemonic for "the current text buffer pointer position". Stems from the fact that the . (dot, or period) TECO command returns this number as its value.

**E command**

One of several two-character TECO commands that start with the letter 'E'.

**Echo mode**

A normal mode of operation in which TECO will automatically echo (display) each character that is typed in response to a ^T command. Opposite of no-echo mode. This mode is controlled by the 8's bit of the ET flag. (0 means no-echo mode.)

**ED flag**

The edit level flag.



**Edit-class command**

A type of operating system command (such as MAKE and TECO) that specifies that file editing is to occur. Many operating systems remember the argument specified with the last Edit-class command, so that the next time an edit-class command is used without an argument, the previous argument can be recalled.

**Edit level flag**

A bit-encoded flag, referenced by the ED command, that describes how TECO should behave with reference to certain features. See Section 5.16 for more details.

**Edit verify flag**

A flag that describes how TECO should act after processing of a command string. This flag can be set so that TECO will display the line just edited on the terminal after each command. See Section 5.16 for more details.

**EH flag The help level flag.**

Either-case search mode A standard mode of operation in which alphabetic characters specified within a search string are permitted to match a corresponding character of either upper or lower case.

**Else-clause**

The part of a conditional command that is executed if the condition is not satisfied. In TECO, these are the commands that occur between the | and the ' characters within the conditional construct.

**End-of-file flag**

A read-only flag, referenced by the ^N command that specifies whether or not end-of-file has been seen on the currently selected input stream. A value of 0 means that end-of-file has not been seen; a value of -1 means that end-of-file has been reached. This flag is initially 0 and is reset to 0 each time a new file is selected for input.

**EO level**

The current version level of TECO-10.

**ES flag**

The search verification flag.

**<ESCAPE>**

The character whose ASCII code is 33 (octal). It is a general-purpose delimiter used by TECO. Traditionally known as an <ALTMODE>. When no ESCape surrogate is active, an ESCape types out as \$ (dollar sign). See also <DELIM>.

**ESCape surrogate**

A character (designated via the 8192 ET bit, or via the EE flag) which causes an ESCape to be transmitted to TECO. When an ESCape surrogate is active, an ESCape types out as ` (accent grave). See also <DELIM>.

**ET flag**

The terminal characteristics flag

**EU flag**

The upper/lower case flag.

**EV flag**

The edit verify flag.

**Exact-case search mode**

A mode of operation in which alphabetic characters within a search string must match the exact case (upper case or lower case) of characters specified.

**Exit Protection**

A protective feature of TECO that prevents a user from exiting TECO if a potential loss of data is imminent. The EX and EG commands are aborted (with the ?NFO error message) if there is text in the text buffer, but no output file is open.

**F command**

One of several two-character TECO commands that start with the letter 'F'.

**Flag**

A data register used by TECO to control various modes of operation. The contents of the flag are set by specifying the new value before the flag name; and the contents are returned by using the flag name without a numeric argument. The TECO flags are: ^X, ^E, ^N, ED, EE, EH, EO, ES, ET, EU, and EV.

**Flow command**

A TECO-11 command that is used to flow (branch) to a particular flow control character. The flow commands are F<, F>, F', and F | .

**Form feed flag**

A read-only flag, referenced by the ^E command that specifies whether the previous append from the input file terminated because of encountering a form feed character in the input file, or because the text buffer became nearly full. The value of this flag is automatically set to -1 by TECO if input is terminated by the presence of a form feed character in the input file.

**Garbage collection**

A process used by TECO-10 to collect unused memory when more memory is required.

**Global Q-register**

A Q-register available throughout all macro levels (including "outside", or at prompt level). Global Q-registers are named A-Z and 0-9. See also Local Q-Registers, and Section 3.3.2.

**Global search**

A type of search that continues through the entire input file until the specified characters are found. Successive pages of the file are read into the text buffer and then written out into the output file (if the string is not located). Global searches in TECO are initiated via the N command.

**Hard-copy editing mode**

A mode of operation that TECO uses when the user's terminal is not a CRT. In this mode, when a character is rubbed out using the <DELETE> key, the rubbed-out character is re-typed on the terminal as a visible indication that this character was rubbed out. Opposite of scope editing mode. This mode can be entered, even on a scope terminal, by turning off the 2's bit of the ET flag.

**Help level flag**

A bit-encoded flag, referenced by the EH command, that controls properties of TECO having to do with error messages and user assistance.

**Immediate command**

A special command to TECO that takes immediate effect, requiring no <DELIM>s to begin its execution.

**"Immediate action editing characters"**

are commands such as <DELETE> and <CTRL/U>, which perform editing of the TECO command string currently being typed in. See Section 4.1.

**"Immediate inspection commands"**

are single-character commands which can be typed as the very first character after TECO's prompting asterisk, and which cause the current location counter to be moved by one line and the new current line to be typed out. Examples are <LF> and <BS>. See Section 1.6.1.

**"Immediate action commands"**

are other commands which may be typed right after TECO's prompting asterisk, such as the \*q to save the previously-typed command in a Q-register or the ^W to re-display a scrolling buffer window. See Section 4.2.

**"Immediate ESCape sequence commands"**

are invoked by keys which generate escape sequences, such as "arrow" or "auxiliary keypad" keys. See Section 4.4. Such commands are usually used to perform editing of the TECO command string currently being entered into TECO. For example, the commands <DELETE> and <CTRL/U> are immediate action commands.

**Iteration**

A language construct that permits a sequence of operations to be re-executed indefinitely or a given number of times. In TECO, this is accomplished by enclosing the commands in angle brackets.

**Kernel**

The TECO-11 kernel refers to the TECO-11 module that implements all those features of TECO-11 that are common to all PDP-11 operating systems. Operating system specific features and the interface to the operating system is accomplished by linking an I/O module tailored for the desired operating system with the kernel.

**Keypad editor**

A true scope editor that uses special keys on the terminal (such as a VT52 or VT100) to control editing functions. VTEDIT is an example of a keypad editor.

**Line**

A portion of text delimited by <LF>, <VT>, <FF>, or the beginning or end of the text buffer. The final delimiter is considered to be part of the line.

**Line-numbered file**

In TOPS-10, an ASCII file that contains line numbers embedded in the start of each line. TECO does not require these line numbers, but can handle them if they are present. They can also be generated or suppressed via the /GENLSN and /SUPLSN switches respectively.

**Line-oriented editor**

An editor that primarily uses line numbers to direct editing, and most of whose commands are line-oriented. TECO is a character-oriented editor, but also has many facilities that work with lines.

**Line-wrap mode** A standard mode of TECO's window support wherein lines that are too long to fit on a single physical line of the user's terminal are automatically continued on the next line. Opposite of truncate mode.

**Literal type-out mode**

A mode that TECO can be put into by setting the 1's bit in the ET flag, or (for a single character on a one-shot basis) by using the :^T command. In this mode, any characters typed by a TECO program via use of one of the commands T, V, ^A, or :G, will be displayed on the user's terminal without any modification. When not in this mode, TECO will convert characters that normally do not print to a form that can be displayed on the user's terminal (e.g. <CTRL/X> displays as ^X and <ESCAPE> displays as \$ or, if a user-designated ESCAPE surrogate is in effect, as `). Literal type out mode is useful when trying to do real-time displays on a CRT terminal. Normal (up-arrow) mode is particularly useful to let you see what characters are really in your file.

**Local Q-register**

A Q-register available to only a single macro level (including "outside", or at prompt level). The local Q-registers for a particular macro level are automatically saved and restored by TECO around execution of a lower-level macro. Local Q-registers are named .A-.Z and .0-.9. See also Global Q-Registers, and Section 3.3.2.

**Log file**

An audit trail that TECO-10 can keep showing all the commands that were typed to TECO and/or all the type out made by TECO. This is useful for reviewing what went wrong with a 'bad' edit. The log file is initiated with the EL command (see Appendix C).

**Macro**

A sequence of TECO commands intended to be loaded into a Q-register and executed as a TECO program.

**Macro level**

Two commands within the same TECO macro are said to be at the same macro level. When one TECO macro calls another, the calling macro is said to be at the higher macro level, and the called macro at the lower macro level.

**Match control construct**

A command, consisting of certain special characters, used within a search string to indicate to TECO that special groups of characters are permitted to match at this point.

**Memory expansion**

TECO's act of acquiring additional storage from the operating system when the currently allocated storage is insufficient to handle the current TECO command. Typically, TECO will attempt to acquire this additional memory before it completely runs out of memory, so as to allow a 'buffer zone' for the user. This allows him to complete a few more commands even in the case where TECO is unable to get more memory. The informative message "[nK Bytes]" or its equivalent is printed on the terminal informing the user that memory usage has expanded.

**Mung**

A recursive acronym for "Mung Until No Good"; an act applied by novice TECO users to their source files.

**MUNG command**

An operating system command used to invoke a pre-written TECO program. The most general form of this command is "MUNG file,data" where "file" is the name of a TECO source program, and "data" is data to be passed to that program.

**No-echo mode**

A mode of operation in which TECO will not automatically echo (display) the character typed by the user in response to the ^T command. This mode is entered by setting the 8's bit of the ET flag. Opposite of echo mode.

**Page**

A portion of text delimited by form feeds. The form feeds are not considered to be part of the page. Sometimes the term 'page' is used to refer to all the text currently in the text buffer.

**Panic Mode**

A condition that occurs (on small, single-user operating systems), when, in the middle of outputting during an edit, the output device fills up so that the I/O transfer cannot continue. TECO recovers gracefully from this condition by printing the ?FUL error message and returning control to TECO without any loss of data. At this point, the user closes the current output file and opens another one on another device (with more room) and resumes editing. At a subsequent time, the two parts of his file can be concatenated back together.

**Pipeline editor**

An editor which only makes sequential edits to a file. The file to be edited is read into the text buffer one piece at a time. Each piece is edited and then written out. Once a piece has been written out, further editing to that piece is not possible unless the output file is re-opened in a later edit as a new file to be edited. TECOs are pipeline editors, with the exception of TECO-11 on VAX/VMS (which offers the capability to page backwards as well as forward).

**Pointer preservation mode**

A mode of operation in which the text buffer pointer will not change after a failing search. This mode is controlled by the 16's bit of the ED flag.

**Primary input stream**

A term used by TECO-11 to refer to the main input file that TECO is using.

**Primary output stream**

A term used by TECO-11 to refer to the main output file that TECO is using.

**Prompt level**

A TECO command is said to be executed from prompt level if it was typed in directly in response to TECO's prompt, as opposed to being executed from a macro.

**Q-register**

One of 36 global or 36 local user-accessible registers provided by TECO. Each Q-register can hold both a number and a string of text. Of particular importance is the ability to store TECO command strings in Q-registers to be used as "macros".

**Q-register push down list**

A last-in first-out stack available to users for saving and restoring the contents of Q-registers.

**Read-with-no-wait mode**

A mode of operation in which the ^T command will not hang until a key is typed on the user's terminal. In this mode, if no key has been struck, TECO returns a -1 as the value of the ^T command. This mode is entered by setting the 32's bit of the ET flag.

**Scrolling**

A form of TECO operation available on CRT terminals capable of supporting split-screen operation. The bottom n lines are used for TECO's asterisk prompt and the usual typed command lines; the remaining lines above are used for an automatically-updated window into the text buffer. A valuable training aid: one can experiment with TECO commands and immediately observe their effects. Not a bad idea for advanced users, either. Controlled by the n,7:W command.

**SEALL mode**

A mode of window operation in which all characters have a distinctive visible display, including characters such as <TAB>, <CR>, and <LF>, which normally do not print. Controlled by the 3:W flag (see Section 5.17). Also known as "View all" mode.

**Search verification flag**

A flag, referenced by the ES command, that controls the action of TECO subsequent to the execution of a command string containing a search command. Proper setting of this flag will enable the user to verify that the search located the correct string, by having the line containing the string found displayed on the terminal. See Section 5.16 for more details.

**Search mode flag**

A flag, referenced by the ^X command, that controls how TECO treats alphabetical characters within search strings. This flag is used to put TECO into either exact-case mode, or either-case mode. If the ^X flag is set to 0, then either case matches in searches. If the ^X flag is set to 1, then exact case matches are required.

**Secondary input stream**

A term used by TECO-11 to refer to an auxiliary input "channel" that was formed by use of the EP command.

**Secondary output stream**

A term used by TECO-11 to refer to an auxiliary output "channel" that was formed by use of the EA command. See Section 5.1.4 for details.

**Scope editing mode**

A mode of TECO in which command line scope editing (q.v.) algorithms are used. This mode is enabled by setting the 2's bit of the ET flag. It is usually automatically enabled by TECO if the operating system can detect that the user has invoked TECO from a scope terminal. Opposite of hard-copy editing mode.

**Split Q-registers**

The feature of TECO that permits storing of both a number and a string within a Q-register. Each Q-register can be considered to consist of two compartments.

**String build construct**

A command, consisting of special characters, used within a text argument to build up the contents of the text argument from other components.

**SUPER TECO mode**

A mode of TECO-8 wherein TECO will read past the end-of-file mark (CTRL/Z) of a sequential ASCII file. This mode is enabled by using the /S switch on an ER or EB command and is terminated when an ER or EB command is issued with no /S switch. This mode is useful for scanning through mass storage devices in an attempt to recover data from files that had previously been deleted.

**Switch**

A construct of the form /SWITCH used within a command that takes a filespecification, to modify the action of the command or attributes of the file specified. Also known as a qualifier.

**Tag**

A label specified within exclamation marks to mark a point within a TECO program. Control can be transferred to this point by the use of a GOTO (Otag) command.

**TECO**

Text Editor and CORrector program.

**TECO.INI**

A file containing TECO commands that is used as a user's private initialization file. When TECO starts up, it looks for such a file in the user's area, and if it finds one, the TECO commands in this file are executed before editing commences.

**TECO.TEC**

A TECO macro used by many operating systems to parse the user's edit-class commands.

**TECO I/O mode**

A mode of I/O operation under the RSTS/E and RSX-11 operating systems, in which the system buffers most characters and returns control to the caller (usually TECO) only when interesting characters (such as <DELETE>, <DELIM>, etc.) are typed.

**TECO's prompt**

Refers to the asterisk (\*) that TECO prints to indicate that it is ready to accept commands.

**TECO SIG**

A DECUS Special Interest Group, consisting of users who are dedicated to the spread of, improvement of, and standardization of TECO.

**Terminal characteristics flag**

A bit-encoded flag, referenced via the ET command, that contains information about the user's terminal and specifies in what manner TECO should support it.

**Text buffer**

The main buffer used by TECO to hold the text to be edited.

**Text buffer pointer**

A pointer that marks the position within the text buffer where TECO is currently 'at'. This pointer always points to positions between characters; it never points at a character. The current character is considered to be the character immediately to the right of the current position of the text buffer pointer.

**Then-clause**

The set of commands within a conditional that are executed if the condition is satisfied. In TECO, these commands immediately follow the "X at the start of the conditional. They are terminated by a | or ' character.

**Tracing**

The act of watching the command-by-command execution of a TECO program. This is accomplished by putting TECO into trace mode, via use of the ? command.

**Trace mode**

A mode of TECO wherein each command that is executed by TECO is also displayed on the user's terminal as it is being executed. This mode is useful for debugging complicated TECO programs. TECO is toggled in and out of this mode via use of the ? command. See Section 5.18.4 for more details.



**Truncate mode**

A mode of TECO's window support wherein lines that are too long to fit on a single physical line of the user's terminal are truncated when the right margin of the scope is encountered. This mode is entered by setting the 256's bit in the ET flag. Opposite of line-wrap mode.

**Type-out-time command**

A special command that makes sense only while TECO is typing out text on the terminal. These commands are <CTRL/S>, <CTRL/Q>, and <CTRL/O> and affect the type out.

**Up-arrow mode**

A standard mode of operation wherein upon type out, TECO will display control characters by typing the visible two-character sequence ^X to represent the control character <CTRL/X>. On many older terminals, the caret character (^), whose octal ASCII code is 136, prints as an up-arrow. Some control characters are not printed in up-arrow mode, notably <TAB>, <LF>, <VT>, <FF>, and <CR>.

**Upper/lower case flag**

A flag, referenced by the EU command, that specifies whether or not case flagging is to occur. If set to -1, no case flagging occurs. If set to 0, lower case characters are flagged on type out. If set to +1, upper case characters are flagged on type out.

**View all mode**

A mode of window operation in which all characters have a distinctive visible display, including characters such as <TAB>, <CR>, and <LF>, which normally do not print. Controlled by the 3:W flag (see Section 5.17). Also known as SEALL mode.

**War and Peace mode**

A mode of operation in which TECO outputs a large informative paragraph of information automatically upon encountering an error in a user's program. This paragraph of information describes the error in painstaking detail and tells the user how to recover. This mode is entered by setting the help level to 3. This mode is particularly useful to Novices and particularly obnoxious to experts.

**Window**

The portion of the text buffer that is currently being displayed on the user's terminal or auxiliary display device.

**Window support**

Assembly language code, built into TECO, that maintains a window into the text buffer on the user's terminal or auxiliary display device.

**Yank protection**

A feature of TECO wherein any Y, \_, or F\_ command that will potentially destroy valid data is aborted. This mode is normally enabled and will cause any of the aforementioned commands to abort with the ?YCA error message if there is text in the text buffer and an output file is open. This feature can be disabled by turning off the 2's bit in the ED flag.

