February 1980

This document describes the operating procedures for the TECO (Text Editor and Corrector) program. TECO is distributed with DIGITAL Operating Systems, but is unsupported by DIGITAL; TECO is Category C software.

PDP–11
TECO User's Guide

Order No. DEC-11-UTECA-B-D

SUPERSESSION/UPDATE INFORMATION: Supersedes DEC-11-UTECA-A-D

SOFTWARE VERSION: TECO-11 V36
TECO-10 V3
TECO-8 V7

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The software described in this manual is Category C software: DIGITAL assumes no responsibility to support the software nor to answer inquireis about it. If you have problems with this software, you may contact the TECO Special Interest Group of DECUS (Digital Equipment Corporation Users' Society) at the following address:

TECO SIG
c/o DECUS, MR2-3/E55
One Iron Way
Marlboro, MA 01752

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation preparing future documentation. to assist us in

The following are trademarks of Digital Equipment Corporation:

Contents

Forward

I used Google Lens to OCR the original DEC TECO manual but used a scan of the title page to retain the original look. Errors were corrected as spotted but no doubt I have missed some. Please report any you find, no matter how small, and I will make the change. Thank you!

Mike Markowski, `mike.ab3ap@gmail.com`
July 2025

Introduction

TECO is a powerful text editing program that runs under most DIGITAL operating systems. TECO may be used to edit any form of ASCII text: program sources, manuscripts, or correspondence, for example. Since TECO is a character-oriented editor rather than a line editor, text edited with TECO does not have line numbers associated with it, and it is not necessary to replace an entire line of text in order to change one character.

The versatility of TECO makes it complex. To help users deal with this complexity, this manual presents TECO in two stages. The first part (Chapter 1) contains basic information and introduces enough TECO commands to allow the novice TECO user to begin creating and editing text files after only a few hours of instruction. The introductory commands are sufficient for any editing application; however, they are less powerful, in many cases, than the advanced commands presented later.

Subsequent chapters describe the full TECO command set, including a review of the introductory 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 novice user. However, the end of Chapter 1 does include an extensive example employing most of the commands introduced up to that point. Students of TECO should experiment with each command as it is introduced, then duplicate the examples on their computer.

This manual is intended to be a reference manual and, except for Chapter 1,

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.

The following notation is used in this manual to represent special characters:

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 Robout

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.

Chapter 1

Basics of TECO

1.1 Using TECO

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

For RT-11, OS/8, TOPS-10, and TOPS-20	`R TECO`
For RSTS/E	`RUN STECO`
For RSX-11	`RUN STEC`
For VAX/VMS	`RUN SYS$SYSTEM:TECO`

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. All text arguments are terminated by a special character (usually an ESCAPE) which 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 ESCAPE characters. The ESCAPE is a non-printing character which may be labelled ESC, ALT, or PREFIX on your keyboard. TECO echoes a dollar sign ($) whenever an ESCAPE is typed. The dollar sign character is used in examples throughout this manual to represent ESCAPE. Note that the carriage return character has no special significance to TECO. Only the double ESCAPE 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 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. 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.

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 means of an ER command terminated by an ESCAPE. The ER command causes TECO to open the specified file and 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$`	General form of the ER command where "filespec" is the designation of the input file. The command is terminated by an ESCAPE, which echoes as a dollar sign.
`ERPR:$`	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 create a want to 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 an EW command terminated by an ESCAPE. 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 defined, the system's default device is assumed. The following examples illustrate use of the EW command.

Command	Function
`EWfilespec$`	General form of the EW command where "filespec" is the designation of the output file. The command is terminated by an ESCAPE, which echoes as dollar sign.
`EWSYS:TEXT.LST$`	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$$`	Open an input file INPUT.MAC to be found on DX1: and open an output file named OUTPUT.MAC. The double ESCAPE ($$) terminates the command string and causes the string to be executed. Note that the ESCAPE which terminates the EW command may be one of the two ESCAPES 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, where n must be a positive (non-zero) integer. 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 buffer. to a position between any two adjacent characters in the

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 beginnning of the nth line preceding the current position. If n is zero, the pointer is moved to the beginning of the like 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 Action Commands [not in TECO-10]

In addition, there are two special type out commands available as a convenience. They are abbreviations for two frequently used commands. These commands consist of a single character and must be the very first character typed after TECO prints its prompting asterisk. They take effect immediately; there is no need to follow these commands by an ESCAPE character. For this reason, these commands are known as immediate action commands.

Command	Function
`<LF>`	Immediately execute the 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.
`<BS>`	Immediately execute the 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:

Command	Function
`Itext$`	Where "text" is a string of ASCII characters terminated by an ESCAPE (which echoes as a dollar sign). 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 then 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 specified string of characters. for a

Command	Function
`Stext$`	Where "text" is a string of ASCII characters terminated by an ESCAPE (which echoes as a dollar sign). This command searches the text buffer for the next occurrence of the specified character string following the current pointer position. If the string is found, the pointer is positioned after the last character on the string. If it is not found, the pointer is positioned immediately before the first character in the buffer and an error message is printed.
`Ntext$`	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. You must then 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 achieve a greater understanding of user 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.

```
1< *EWDT1: FILE.TXT$$
2< *HKIMR. JOHN P. JONES
 ! COMPUTER ELECTRONICS CORPORATION
 ! BOSTON, MASAASACHUSETTS
 !
 ! DEAR MR. JONES:
 !
 ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
 ! PERTAINING TO THE NEW TECO TEXT EDITING AND CORRECTING
 ! PROGRAM.
 !
 ! ENCLOSED IS A COPY OF THE TECO USERS'S GUIDE, WHICH
 ! SHOULD ANSWER ALL OF YOUR QUESTIONS.
 !
 ! SINCERELY,
 !
 !
 !
 !
 ! $$
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,
 !
 !
 !
 !
 !
5< *EX$$
 ! (TECO is rerun, operating system dependent)
 ! *ERDT1: FILE1.TXT$EWLP:$$
6< *Y5KIMR. JAMES B. SMITH
 ! DATEK ASSOCIATES, INC.
 ! 1122 MAIN STREET WEST
 ! AUSTIN, TEXAS
 !
 ! DEAR MR. SMITH:
 ! $$
 ! *HT$$
 ! MR. JAMES B. SMITH
 ! DATEK ASSOCIATES, INC.
 ! 1122 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 ESCAPE ($$), 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 ESCAPE. 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 SETTSS 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 written onto to be the output file and closed the output file. The user then reentered TECO and reopened the file "FILEL.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 "FILEL.TXT" on DT1.

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. Readers who are first learning TECO may wish to consult the following document (available from Digital Equipment Corporation) for more basic material: DEC-10-UTECA-A-D INTRODUCTION TO TECO.

The commands described in this manual are those implemented in TECO-11 Version 36, 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.

Chapter 2

Invoking TECO

2.1 Running TECO

To run TECO with no special options or defaults, use the same on command 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, new a 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).

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 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.

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 VT100 terminals only).
	`/VTEDIT`	Load VTEDIT video terminal editor

The /SCROLL switch may take a value of the following form:

`/SCROLL:n`	Enter split screen scrolling mode, using n lines for the scrolling area.

The /VTEDIT switch may also take values (of the form :value).

`/VTEDIT:HOLD`	Start up in hold screen mode
`/VTEDIT:SEEALL`	Start up in SEEALL mode

These values can be combined, viz.: `/VTEDIT:HOLD:SEEALL`.

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!

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:

Value&l	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.

Chapter 3

Conventions and Structures

3.1 TECO Character Set

TECO accepts the full 7-bit ASCII character set; all characters have their 8th bit (the parity 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).

TECO command strings may be entered using upper case characters (as shown throughout this manual) or lower case characters. The commands MQ, mQ, Mq, and mq are treated alike. 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 alphabetics 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 immmediate 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

A TECO command consists of one or two characters, optionally preceded by a numeric argument and sometimes followed by a text argument. TECO commands may be strung together by concatenating them into a single command string. No delimiter is necessary between commands, although a single ESCAPE or caret-[may be inserted between them if desired. This ESCAPE will keep numeric values generated by the preceeding command from affecting the commands that follow the ESCAPE.

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 ESCAPE 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 occurrs.

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:

	Signed		Unsigned
System	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

Table 3-1 Restrictions on numeric arguments

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 flaq.
- 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. $N = 1$ is assumed when n is omitted.

  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:

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

Table 3-2 Default Arguments

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 an ESCAPE character (or in the case of and "A commands, by the command character). The string of ASCII characters may not include an ESCAPE, 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 arquments. Each argument must followed by an ESCAPE character, as follows: be

FSabcdef	Replace string "abc" by "def"

You can include an ESCAPE 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>.

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 any order.

3.3 Data Structures

A good way to begin the study of a programming language is the commands, for the moment, to forget 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

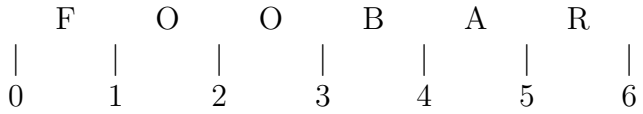
- Any character string can be divided into TECO pages by considering the page to be ended by either

  a form feed (octal 14) or
  the end of the given character string

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

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 position at 0 is the 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:



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 36 data storage registers, called Q-registers, which may be used to store single integers and/or ASCII character strings. The Q-registers have one character names: A through Z and 0 through 9. Each Q-register is divided into two storage areas. In the numeric storage area, each Q-register can store one signed integer. In the text storage area, each Q-register can store an ASCII character string which may be either text or a TECO command string.

Various TECO commands allow the storing and retrieving of numerical 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 and restored.

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.

Chapter 4

Command String Editing

While you are typing command strings at a terminal, TECO considers certain ASCII characters to have special meaning. Most of the special characters are immediate action commands, which cause TECO to perform a specified function immediately, instead of waiting for the double ESCAPE which terminates a command string. Immediate action commands may be entered at any point in a command string—even in the middle of a command or text argument.

Many immediate action commands, such as DELETE, which deletes the immediately preceding character, cannot be used as regular TECO commands. If you enter a DELETE into a command string executed from a macro, the DELETE will not delete a character as part the execution of the command string.

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

Control characters used in immediate action commands must be entered

using the CONTROL key; they will not be recognized if entered as a caret or up-arrow.

Tables 4-1 and 4-2 list the immediate action commands and explain their functions.

Table 4-1: Immediate Action Commands

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

Character	Explanation
`<ESCAPE><ESCAPE>`	The double <ESCAPE> character sequence tells TECO to begin execution of the command string just typed in. It echoes as $$ and also inserts two <ESCAPE>s into the command string. The two ESCAPES must be typed successively. If any other character is typed in between the two ESCAPES (even if subsequently DELETED), then the two ESCAPES might be treated simply as two ESCAPES to be entered into the command string rather than as an immediate action command. If you need to enter two ESCAPEs into a command line, as in the case where you want to use the `FSstring$$` command to delete a string, you can keep TECO from recognizing $$ as an immediate action command. Type `<ESCAPE><SPACE><DELETE><ESCAPE>` and then continue entering the remainder of your command string. A single ESCAPE character is not a special character and performs no immediate action.

`<DELETE>` 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.

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.

`<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.

`<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 line feed followed by a 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).

Table 4-2: Immediate Action Aids

These immediate action commands are aids to the user. They have special meaning and take immediate effect only if they are typed as the very first keystroke after TECO's asterisk prompt.

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. [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 the 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]

`<BS>` Typing this immediate action command, backspace, (as the first keystroke after TECO's prompt) causes TECO to immediately execute the 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]

`*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 <ESC><ESC>.1

Although TECO accepts all 128 characters of the 7-bit ASCII character set, whether encountered in a macro, or entered directly into a command string from the terminal, 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 must be entered indirectly, such as by entering them into a macro using the `nI$` command.

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
	^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)
`^Y`	Reboot indicator [F/B systems onlyl]

Chapter 5

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:

Table 5-1A: File Specification 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.

Table 5-1B: 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
	`/B+`	Handle a BASIC-PLUS file
	`/B2`	Handle a BASIC-PLUS-2 file
	`/CLUSTERSIZE:n`	Specifies output file cluster size
	`/MODE:n`	Use non-standard open mode
	`/n`	Handle BASIC-PLUS-2 continuation character (&) in column n
RSX-11		
VAX/VMS	`/B2`	Handle a BASIC-PLUS-2 file
	`/CR`	Implied carriage control
	`/-CR`	No (internal) carriage control
	`/FT`	FORTRAN carriage control
	`/RW`	Rewind magtape before opening file
	`/SH`	Open the file in shared mode
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 5-1C: File Close and Exit

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.) [Not 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).

`:EGtext$` Performs the same function as the EC command, and then performs an operating system specific function. 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.)

`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 ouput 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 5-1D: Secondary Stream Commands
[Not in TECO-8 or TECO-10]

Input commands (do not open or close any file; do not change the text buffer):

`EP` Switches the input to the secondary input stream.

`ER$` Switches the input to the primary input stream.

Output commands (do not open or close any file; do not change the text buffer):

`EA` Switches the output to the secondary output stream.

`EW$` Switches the output to the primary output stream.

Indirect file commands:

`EIfilespec$` 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. 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 ESCAPEs.

All commands encountered in the indirect file will have their normal TECO meaning (as opposed to any immediate action meaning). For example, a <CTRL/U> 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 <ESC><ESC> command, which directs TECO to execute the preceding command string and then return to the indirect file at the point following the <ESC><ESC>.

`EI$` 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 ESCAPE which has not yet been read is discarded. This command has no effect if no indirect file is already open.

5.1.5 Wildcard Commands

TECO-11 supports wild card file processing with set of special commands, to allow TECO programs to operate on a set of files.

Table 5-1E: Wildcard Commands

[Not in TECO-8 or TECO-10]

`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 occurences 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 occurences 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 5-8A (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 1/0 to bypass the text buffer.

Table 5-1F: Direct I/O to Q-Registers

[Not in TECO-8 or TECO-11]

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

Table 5-2: Page Manipulation Commands

Command Function

Append commands:

`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. 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]

Page Commands:

`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. O 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.
[Not in TECO-8]

.

`nP` Executes the `P` command n times, where n must be a non-zero positive integer.

`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 positive non-zero integer.

`m,nPW` Writes the contents of the buffer between pointer positions m and n onto the output file. m and in 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`.

Yank commands:

`Y` Clears the text buffer and then reads the next paqe of the input file into the buffer. As the `Y` command can result in the loss of data, it is not permitted under certain circumstances (see ED flag in 5.14).

`: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]

`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.

`: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.
[Not in TECO-8]

5.3 Buffer Pointer Manipulation Commands

Table 5-3 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 5-3: Buffer Pointer Manipulation Commands

Command	Function
Character commands:	
`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.
`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`.

Jump commands:

`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]

Line commands:

`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`.

Reverse commands:

`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 in 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. 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`.

5.4 Text Type Out Commands

Table 5-4 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 5-4: 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 the character whose ASCII code is n. This can be used to output any ASCII character to the terminal.
`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 5-4B: 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).

5.5 Deletion Commands

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

Table 5-5: Text Deletion Commands

Command	Function
Delete commands:	
`D`	Delete the first character following the current position of the buffer pointer.
`nD`	Execute the `D` command n times. If n is positive, characters the the current pointer position following n 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)
`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 <ESCAPE> 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 <ESCAPE> character.

Kill commands:

`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, then 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 5-6 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 unpredictible 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 5-6: Text Insertion Commands

Command Function

Insert commands:

`Itext$` Where "text" is a string of ASCII characters terminated by an ESCAPE. 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$` This form of the `I` command inserts the single character whose 7-bit ASCII code is n into the buffer at the current position of the buffer pointer. (n is taken modulo 128.) nI$ 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 3.1.2 above.

`n@I//` Equivalent to the `nI$` command, but does not require the ESCAPE 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 nce 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.
[Not in TECO-8]

`@FR/text/` Equivalent to "`FRtext$`", except that "text" may contain ESCAPE characters.
[Not in TECO-8]

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 search a command is specified with a null search string argument, the last explicitly defined search string will used. be 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 5-7A: Search Commands

`Stext$` Where "text" is a string of characters terminated by an ESCAPE. 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.

`nStext$` 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.

`-nStext$` Identical to "`nStext$`" 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.

`-Stext$` Equivalent to `-1Stext$`.

`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. 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.

`nNtext$` This command searches for the nth occurrance of the specified character string, where n must be greater than zero. It is identical to the `N` command in other respects.

`_text$` 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.

`n_text$` 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.

`E_text$` Same as `_text$` 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.

`nE_text$` Same as `n_text$` command except that effect^Aive `EY` (rather than Y) commands are used.

Table 5-7B: Bounded Search Commands

`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$` Compare command. The command is not a true search. If the characters in the buffer immediately following the current pointer position match the search string, the pointer is moved to the end of the string and the command returns a value of -1; i.e., the next command is executed with an argument of -1. If the characters in the buffer do not match the string, the pointer is not moved and the command returns a value of 0. Identical to "`.,.:FBtext$`".

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 5-7C: Search and Replace Commands

Search & Replace	Search Command
`FStext1$text2$`	`Stext1$`
`nFStext1$text2$`	`nStext1$`
`FNtext1$text2$`	`Ntext1$`
`nFNtext1$text2$`	`nNtext1$`
`F_text1$text2$` [not in TECO-10]	`_text1$`
`nF text1$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:

`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.

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 (ESCAPE) 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$text2$
```
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 5-8A 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 5.16 and appendices).

Table 5-8A: 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).
`<CTRL/V><CTRL/V>`	Two successive <CTRL/V> characters in a string argument indicate to TECO that all1 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 section 5.9 below. [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 5-8B lists these match control characters and their functions.

Table 5-8B: 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.

`<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 occurrences of non-zero any number 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-Registers

TECO provides 36 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: a number storage area and a text storage area. The number storage area can store one signed integer. The text storage area can store an ASCII character string which may be either text or a TECO command string. Each Q-register has a single character name which is one of the letters A to Z or one of the digits 0 to 9. Upper and lower case letters may be used interchangeably.

TECO-10 allows additional Q-registers besides the 36 described above. Consult Appendix I for further details.

Table 5-9A lists the commands which permit characters to be loaded into the Q-registers.

Table 5-9A: 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 resulting value contained in Q-register q is used as a numeric argument for the next command. If the next command does not require a numeric argument, this value is discarded.

`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 7-bit ASCII code is n into the text storage area of Q-register "q".
[not in TECO-8]

`n:^Uq$` This form of the `&Uq$` command appends the single character whose 7-bit ASCII code is n to the text storage area of Q-register "q".
[not in TECO-8]

`@^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 <ESCAPE> 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 preceeding 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. This command does not use numeric values. Numeric values are passed through this command as if it did not occur. 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]

Table 5-9B lists the commands which permit data to be retrieved from the Q-registers.

Table 5-9B: 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. The text buffer and buffer pointer are not changed by this command.

`Mq` Execute the contents of the text storage area of Q-register q as a command string. Mq commands may be nested recursively as far as TECO's push down storage will permit.

`nMq` Execute the Mq command, using n as a numeric argument for the first command contained in Q register q.

`m,nMq` Execute the Mq command, using m,n as a numeric argument for the first command contained in Q register q.

`[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 numeric values. Numeric values are passed through this command as if it did not occur, allowing macros to save temporary Q-registers and still accept numeric values. The command sequence `[A ]B` copies the text and numeric value from Q-register A to Q-register B.
[Not in TECO-8]

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 5-11, 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 5-10A lists all of the arithmetic operators that may be used in arithmetic expressions.

Table 5-10A: 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 5-10B: 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 3) 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/0) 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]

`^R` This command returns the binary value of TECO's current radix.
[TECO-11 only]

`n^R` 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.

`\` 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 + 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.

`n\` 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), unsigned hexadecimal (hexadecimal radix). Note that `\` is a "bidirectional" command. `n\` inserts a string into text while `\` (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 5-11 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.

Table 5-11: 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 an ESCAPE.

`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:

OS/8:	^B = ((month*32)+day)*8)+((year-1970)/7)+k where k = 4096 if year>1977 and k = 0 otherwise.
RT-11:	^B = (((month*32)+day)*32)+year-1972
RSTS/E:	^B = ((year-1970)*1000)+day within year
RSX-11:	^B = ((year-1900)*16+month)*32+day
VAX/VMS:	^B = ((year-1900)*16+month)*32+day
TOPS-10:	^B = (((year-1964)*12+month-1)*31+day-1)

`^E` <CTRL/E> (caret/E) is equivalent to -1 if the buffer currently contains a full page of text (which was terminated by a form feed in the input file) or 0 if the buffer contains only part of a page of text (which either filled the buffer to capacity before the terminating form feed was read or which was not terminated by a form feed.) The `^E` flag is tested by the `P` command and related operations to determine whether a form feed should be appended to the contents of the buffer on output.

`^F` <CTRL/F> (caret/F) is equivalent to the current value of the console switch register.

`n^F` `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]

`^H` <CTRL/H> (caret/H) is equivalent to the current time of day via the following equations:

OS/8:	^H	=	0
RT-11:	^H	=	(seconds since midnight)/2
RSTS/E:	^H	=	minutes until midnight
RSX-11:	^H	=	(seconds since midnight)/2
VAX/VMS:	^H	=	(seconds since midnight)/2
TOPS-10:	^H	=	60ths of a second since midnight

(or 50ths of a second where 50 Hz power is used)

`^N` <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.

`^S` <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]

`^T` <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..

`^Y` <CTRL/Y> (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 "`^YXAFR$`" to store the string in Q-register A and remove it from the buffer. You can then position the pointer to the right place and type "`GA`" to insert the string.
[TECO-11 only]

`^Z` <CTRL/Z> (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]

`^^x` 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.

Mode Control Flags

The use of these flags is described below in section 5.16.

`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 36, TECO-8 Version 7, and TECO-10 Version 3.

`ES` Equivalent to the current value of the search verification flag.
[Not in TECO-81

`ET` Equivalent to the current value of the type out control flag.

`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. performed n times. If n is greater than 0, then the iteration is

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 an ESCAPE. 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 5-13: 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]

`nOtag0,tag1,tag2,...$`
: This command causes TECO to branch to the tag specified by the *n*th 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 dictates 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, then command execution continues with the first command following the <ESC> that delimits this command.
[TECO-11 only]

`n@O/tag0,tag1,tag2,.../`
: 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]

`;`
: 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.

`n;` 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.

`:;` 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]

`n:;` 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]

`'` 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 <ESC><ESC> 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 conditional, the ?MAP error (or its equivalent) is issued. [TECO-11 only]

`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 1, 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.

`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 '
```

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 5-14: 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 of code any character that is a symbol constituent. This is usually one of the upper or lower case letters A to 7, 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 zero. is greater than

`n"L` Execute the following command string if n is less than zero.

`n"N` Execute the following command string if n is not zero. equal to

`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`.

`n"T` Execute the following Equivalent to `n"L`. command string if n is TRUE.

`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 5-15A: Retreiving Environment Characteristics

Command Function

`-1EJ` 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:

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	0	OS/8
2	DEC-10	0	TOPS-10
3	DEC-20	0	TOPS-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 console 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 5-15B: Setting Environment Information

`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 above.

Table 5-16A: Flag Manipulation Commands

`<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 5-16B: 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:

`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.

`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` Reserved for future use by TECO-11.

`ED&64` Only move dot by one on multiple occurence 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 `5SAAS` 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]

The initial value of `ED&1` is system dependent (See the appendices). The initial value of the other bits in the ED flag is 0.

`EH` The help level flag, which controls the printing of error messages and failed commands.

`EH&3` 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):

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. This mode is not implemented in TECO-11, in which case $m = 3$ is equivalent to $m = 2$.

`EH&4` 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.

The initial value of the EH flag is 0 which is equivalent to a value of 2.

`EO` Setting the value of the EO flag to n allows features that were peculiar to that version of TECO to work.
[TECO-10 only]

`ES` The search verification flag, which controls the text typed out after searches.

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 $m*256+n$. Then is the same as above. The mis 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]

The initial value of ES is 0.

`ET` The ET flag is a bit-encoded word controlling TECO's treatment of the console terminal. Any combination of the individual bits may be set. The bits provide the following functions, when set:

`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 $ (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 alphabetics to upper case on input. Setting this bit causes lower case alphabetics 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/0> on type out. Setting this bit will cancel any outstanding <CTRL/0> when the next type out occurs. After TECO has canceled the <CTRL/0>, 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).

`ET&128` When this bit is set: 1) all informational messages are supressed, 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.)

`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&32768` If this bit is set and a <CTRL/C> 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 <CTRL/C>s. In TECO-8, this bit is the 2048's bit rather than the 32768's bit.

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.

`EU` The upper/lower case flag.

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.

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 te initial value is 0. Consult the appendices for more details.

`EV` 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.

The initial value of the EV flag is 0.
[TECO-11 only]

^X The search mode flag.
[Not in TECO-8]

If n 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. Thus the lower case alphabetics match the upper case alphabetics and `"‘", "{", "|", "}", "~"` match `"@", "[", "\", "]"`, If n respectively. is -1, the search will succeed. only if the text argument is identical to text in the text buffer.

The initial value of the X flag is 0.

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 (such on. a video terminal as a VT05, VT52, or VT100), ET flag Bit 9 (value 512) will be

Table 5-17A: 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).
OW	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` command is used to interrogate and set video terminal status information, as well as implement some of the more advanced features of the video terminal "WATCH" functions.

Table 5-17B: Video Terminal Status Commands

Command	Function
`0:W`	Return a number representing the type of scope in use as the editing terminal. Current values are: 0 VT52 1 VT61 [TECO-10 only] 2 VT100 in VT52 mode 4 VT100 in ANSI mode 6 VT05
`:W`	Equivalent to `0:W`
`1:W`	Return 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.
`2:W`	Return 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.

`3:W` Returns 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` Returns 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` Returns 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.

`7:w` Returns the number of scrolling lines. If n is zero, then scrolling is turned off. When 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 window and is updated by TECO immediately before each command prompt. The scrolling feature only functions on terminals capable of split screen scrolling functions, such as the VT100.

`m,n:W` Sets the entity represented by `n:W` to `m`. Specific operating systems may put restrictions on the valid values for m.

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 5-17C: 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.

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 ofthe 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 Q-register, say Q-register C. in a 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 distinc-

tive 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 betweeen 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 5-18A: 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.
`<ESCAPE>`	An ESCAPE that is executed as a TECO command (as distinct from 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) when that value is not needed.
`^[`	Same as <ESCAPE>. Like any other TECO command that is a control character, it may be entered in up-arrow mode. In that mode, `^[` is useful on systems whose line-printer spoolers do not visibly print the ESCAPE character.
`<ALTMODE>`	On some older terminals, there is no ESCAPE key. Instead, there may be 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 5-18B: 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]
^W	<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 overriden by explicit case control within the search string. This command makes all strings behave as if they began with ^W^W. [TECO-10 only]
0^V	Returns TECO to its original mode. No special conversion occurs within strings except conversions that are explicitly specified by string build constructs located within the string. case those case ^V and ^W string build constructs located within the string. [TECO-10 only]
0^W	Same as ^V. [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,.PWO,.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 ESCAPE 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.

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:

```
[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 !
EUU2          ! 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 El 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 are 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 yo are encouraged to experiment with the individual commands on scratch files.

Example 1: Splitting, Merging, and Rearranging 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
```

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.)

```
                    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 D 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 achieve this rearrangement: to

```
                    Start TECO.
*2ED$               Allow all Y commands.
*ERFILESEWFILE1$$   Open the input file and the first output file.
*Y$$                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$$               Search for a character string in K.  After this
                    command has been executed, the buffer will contain
                    KL. No output is generated.
*EWFILE2$P$$        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 the form: files. One file is MATH.ONE, which contains information in

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

and the other is MATH.TWO, which contains:

```
MN <FF> OP <FF> OR
```

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.
```

Example 2: Alphabetizing by Interchange Sort

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.

```
!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 @0/CONT/ '     !Loop back if there is another line
                          in the buffer !
QZ"G 0UZ @0/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- .Z;>                    !Loop back if there is another line
                           in the buffer !
QZ;>                       !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.

Appendix A

Octal & Decimal ASCII Character Set

CHAR	OCT	DEC	CHAR	OCT	DEC	CHAR	OCT	DEC	CHAR	OCT	DEC
NUL	000	000	SP	040	032	@	100	064	‘	140	096
^A	001	001	!	041	033	A	101	065	a	141	097
^B	002	002	"	042	034	B	102	066	b	142	098
^C	003	003	#	043	035	C	103	067	c	144	099
^D	004	004	$	044	036	D	104	068	d	144	100
^E	005	005	%	045	037	E	105	069	e	145	101
^F	006	006	&	046	038	F	106	070	f	146	102
^G	007	007	’	047	039	G	107	071	g	147	103
BS	010	008	(	050	040	H	110	072	h	150	104
TAB	011	009	)	051	041	I	111	073	i	151	105
LF	012	010	*	052	042	J	112	074	j	152	106
VT	013	011	+	053	043	K	113	075	k	153	107
FF	014	012	,	054	044	L	114	076	l	154	108
CR	015	013	-	055	045	M	115	077	m	155	109
^N	016	014	.	056	046	N	116	078	n	156	110
^O	017	015	/	057	047	O	117	079	o	157	111
^P	020	016	0	060	048	P	120	080	p	160	112
^Q	021	017	1	061	049	Q	121	081	q	161	113
^R	022	018	2	062	050	R	122	082	r	162	114

^S	023	019	3	063	051	S 123 083	s 163 115
^T	024	020	4	064	052	T 124 084	t 164 116
^U	025	021	5	065	053	U 125 085	u 165 117
^V	026	022	6	066	054	V 126 086	v 166 118
^W	027	023	7	067	055	W 127 087	w 167 119
^X	030	024	8	070	056	X 130 088	x 170 120
^Y	031	025	9	071	057	Y 131 089	y 171 121
^Z	032	026	:	072	058	Z 132 090	z 172 122
ESC	033	027	;	073	059	[133 091	{ 173 123
FS	034	028	<	074	060	\ 134 092	| 174 124
GS	035	029	=	075	061	] 135 093	} 175 125
RS	036	030	>	076	062	^ 136 094	~ 176 126
US	037	031	?	077	063	_ 137 095	DEL 177 127

Appendix B

Error Messages

TECO error messages consist of a three letter message preceded by a question mark (?) or preceeded 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. abort the command and execution continues. These messages do not

`%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`

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

`?BNI` `> not in iteration`

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`

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

`?CON` `Confused use of conditionals`

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`

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

`?DEV` `Invalid device`

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

`?DTB` `Delete too big.`

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

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

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

`?FER  File Error`

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

`?FNF  File not found "filespec"`

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`

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`

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`

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`

An invalid F command has been executed.

`?IFN  Illegal character "x" in filename`

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`

A command of the form "`nItext$`" was attempted. This combination of character and text insertion is illegal.

`?ILL  Illegal command "x"`

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

`?ILN  Illegal number`

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

`?INP  Input error`

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`

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

`?IQC  Illegal " character`

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"`

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

`?IRA  Illegal radix argument to ^R`

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

`?ISA  Illegal search arg`

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

`?ISS  Illegal search string`

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 ^`

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

`?MAP  Missing '`

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

`?MEM  Memory overflow`

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.)

`?MLA  Missing Left Angle Bracket`

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 (`

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

`?MRA  Missing Right Angle Bracket`

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 )`

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

`?MSC  Missing Start of Conditional`

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

`?NAB  No arg before ^_`

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

`?NAC  No arg before ,`

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

`?NAE  No arg before =`

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

`?NAP  No arg before )`

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 "`

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 ;`

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`

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

`?NCA  Negative argument to ,`

A comma was preceded by a negative number.

`?NYA  Numeric argument with Y`

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

`?NFI  No file for input`

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

`?NFO  No file for output`

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

`?NPA  Negative or 0 argument to P`

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

`?NRO  No room for output`

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

`?NYI  Not Yet Implemented`

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

`?OFO  Output file already open`

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`

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`

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`

The command string has become too complex. Simplify it.

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

A J, C or R command has been executed which attempted move to 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 nA command must also be within the buffer boundary.

`?SNI  ; not in iteration`

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

`?SRH  Search failure "text"`

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`

A search or file name string is too long. This is most likely the result of a missing ESCAPE after the string.

`?TAG  Missing Tag !tag!`

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"`

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`

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`

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

`?WLO  System Device Write-Locked`

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`

An attempt has been made to execute an Y or _ search 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)

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.

Appendix C

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 he 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, these commands at your own risk and with the understanding that in future releases of TECO, these commands may change or go use 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

Appendix D

RT-11 operating Characeristics

D.1 Startup

TECO is started with the

```
.R TECO
```

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

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, 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 com-

mand 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
*ERfilespec$YHXZHKIstring$MZ$$
```

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

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 EGstrings 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

a block TECO can be a useful tool in recovering ASCII files lost on 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.

Appendix E

RSTS/E Operating Characeristics

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 is privileged, 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 specification in the standard RSTS/E format: a file

```
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 /RONLY, /MODE:n, and /CLUSTERSIZE:n can be included in a file specification. TECO automatically opens all disk input files 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.

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 a non-privileged user attempts to set Bit 6, the request is ignored and Bit 6 will read back as a 0 (assuming the job is attached). When a privileged user sets Bit 6, the job will become detached. Further reading of Bit 6 will return a 1 to indicate the detached condition.

Appendix F

RSX-11 Operating Characeristics

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 buffer space as

expand its necessary. Also, TECO'S buffer space may be explicitly allocated in the startup command

```
RUN STEC/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.

If the When TECO is initially invoked it will automatically set the ET and EU flags according to the user's terminal characteristics. 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 occurr. 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 file a 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 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 ERS 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 desireable 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 ^0), 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 is 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 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. to be

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
Fortran	Fortran
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 EIS 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 a user's TECO.TEC startup file, the final of MY$$ which causes processing of the command line is not executed. This results from the fact that the MYSS normally appears in TECO's type in after all command files have been processed. Executing the EIS 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 EIS 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.

Appendix G

VAX/VMS Operating Characeristics

TECO is implemented in VAX/VMS as a half-native, half-compatibility mode program. The command processor and editor proper is the same as TECO-11 and runs in compatibility mode. Operating system interface and file service logic run in native mode.

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.13) in this appendix.

TECO macros may also be invoked with the command

```
TECO @filespec
```

It is exactly equivalent to

```
MUNG filespec
```

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.

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 to at startup (e.g., via a TECO /VTEDIT command), it attempts to translate the logical name TEC$VTEDIT. If the name is defined, the resulting filespec is used as the file from which to load the scope editor. If the logical name is not defined, TECO defaults to SYS$LIBRARY:VTEDIT.TEC.

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 type except for EI commands which default the type field to default .TEC.

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 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 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 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!

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 VMS. (See appendix F, section F.3.)

G.4 Wild Card Lookup

Wild card file name processing supports all of the wild carding facilities of RMS-32, including embedded * and %, and multi-level directory wildcarding (not under VAX/VMS V1).

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 alphabetics, numerics, `.`, `$`, and `_`.

G.6 Exiting from TECO

This The normal method of exiting from TECO is with the EX command. 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.

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 desireable 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 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. This feature works only on VMS V2 and later systems.

G.10 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 ^0), 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 is reaches prompt (*) level.

Bit 6 (the detach flag) has no meaning in VMS.

G.11 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 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 retur / 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
Fortran	Fortran
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.

G.12 Command Line Processing

The mechanism used to process the command line in 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 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 he 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 TEC$INIT 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.

G.13 Installing TECO

TECO is distributed with VAX/VMS; the files are already in place. If TECO will receive heavy use, /HEADER it should be installed /OPEN and /HEADER_RESIDENT.

NOTE

to The half-native TECO is distributed with VAX/VMS V2. Previous versions of VAX/VMS were distributed with the compatibility mode RSX-11M TECO. Should you wish run the half-native version on VAX/VMS Vl.x systems, you must install it with CMEXEC privilege to allow its handling of TEC$MEMORY. This privilege is NOT required on VAX/VMS V2 or later..

Since the TECO commands are not part of the VAX/VMS command interpreter, each user must define the commands in his LOGIN.COM file. The following commands will define the three normal TECO invocation commands:

```
$ TECO :== $TEC TECO
$ MAKE :== $TEC MAKE
$ MUNG :== $TEC MUNG
```

One may include command switches in the command definitions. For example, one can define command to invoke TECO with VTEDIT as follows: a

```
$ VTECO :== $TEC TECO /VTEDIT
```

Appendix H

OS/8 Operating Characteristics

H.1 Startup

TECO is started with the

```
.R TECO
```

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

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 filespecl. That is, filespec2 is opened as the input file, and filespecl is created as the output file. It is equivalent to

```
.R TECO *ERfilespec2$EWfilespecl$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.

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 arqument (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 value	Initial value
1	0
2	0 (1 if terminal is a scope) 0
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 EGstrings 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 The command may

be either a a KBM or a CCL command. This command is especially useful while running under BATCH.

If TECO is exited via the EGS 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:

```
.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 desireable 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 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 for

```
ERFOO.MAC/$$
```

will open the file FOO.MA for input and put TECO into "SUPERTECO" mode. In this mode, TECO will not treat a <CTRL/2> found in a file as an end-of-file character. Instead, <CTRL/2> 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, ERS, EWS)
2. Auxiliary command streams (EI)
3. Wildcards (EN)
4. Zeroing of directories (EZ)

   Eitem Magtape commands (EM)
5. View command (nV)
6. Bounded searches 8. Anchored searches
7. Search verification (ES)
8. Command verification (EV)
9. Backward searches.
10. 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 possiblities, 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 com-

mand), 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 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 0TT 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 be used as infrequently as possible from within long-running macros. Xq is preferred to ^Uq 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
`TECOVO.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
```

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.

Appendix I

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 specifiied 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.

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 Ror 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 value	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 EGS 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 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, ERS, EWS)
2. Wildcards (EN)
3. Immediate aids LF and BS.

The following incompatibilities exist between TECO-10 and DEC's TOPS-10 TECO V24:

1. 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 qives 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$
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. in Note that these switches are needed only if a change is to be made 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 must 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 buf 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 followed 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.