September 1977

This document describes the operating procedures for the TECO (Text Editor and COrrector) program. TECO is distributed with the IAS, RSX-11D, RSX-11M, and RT-11 operating systems, but is unsupported by DIGITAL; TECO is Category C software.

PDP-11
TECO User's Guide

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

SUPERSESSION/UPDATE INFORMATION: This manual includes Update Notice No. 1 (DEC-11-UTECA-A-DN1).

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 be used or copied only 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 inquiries about it. If you have problems with this software, you may contact the RT-11 Special Interest Group of DECUS (Digital Equipment Corporation Users' Society) at the following address:

RT-11 Special Interest Group
c/o DECUS
Digital Equipment Corporation
129 Parker Street PK3-1/E55
Maynard, Massachusetts 01754

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECSYSTEM-20	TYPESET-11

TABLE OF CONTENTS

INTRODUCTION

TECO is a powerful text editing program that runs under most PDP-11 operating systems. TECO may be used to edit any form of ASCII text such as program listings, manuscripts, correspondence and the like. Since TECO is a character-oriented editor rather than a line editor, text edited with TECO does not have line numbers associated with it, nor is it necessary to replace an entire line of text in order to change one character.

Because TECO is very versatile, it is necessarily complex. This manual is, therefore, divided into two parts. Chapter I 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 convenient, in most cases, than the advanced commands presented later.

Chapter II introduces the full TECO command set, including a review of the introductory commands presented earlier. This chapter also introduces the concept of TECO as a programming language and explains how basic editing commands may be combined into "programs" which are sophisticated enough to handle the most complicated editing tasks.

Specific examples of the use of TECO commands have been de-emphasized throughout this manual. This was done because all of the TECO commands have a consistent, logical format which will quickly become apparent to the novice user. However, each chapter of the manual is concluded with one or more elaborate examples which employ most of the commands introduced up to that point. Users who are learning TECO commands should experiment with each command as it is introduced, then duplicate the examples on their computer.

CHAPTER I

INTRODUCTORY COMMANDS

1.0 FUNDAMENTALS

TECO considers text to be any string of ASCII codes. Text is broken down into units of pages, lines, and characters. A page of text consists of all the ASCII codes between two form feed characters, including the second form feed. A character is one ASCII code. Thus, every page of text contains one form feed character, which is the last character on the page. Every line of text contains one line feed, which is the last character on the line.

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

Line feed and form feed characters are inserted automatically by TECO. A line feed is automatically appended to every carriage return entered into the buffer, and a form feed is appended to the content 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. That is, all text preceding the form feed will appear on one page, and the text following the form feed will appear on the next page.

Finally, TECO also maintains an input file and an output file, both of which are selected by the user through the use of file specification commands. The input file is any device from which text may be accepted. For example, if a block of text is stored in a disk file, the disk file would be specified as an input file when the text is edited.

The output file is any device on which edited text may be written. If the disk file mentioned above were to be edited, it could be written, for example, onto another disk file.

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 a portion of text has been written to the output file, it cannot be accessed again without closing the output file and re-opening it as an input file.

TECO may be called from command level by typing:

.R TECO	For RT-11
RUN $TECO	For RSTS/E
TECO	For RSX-11

(terminated with a carriage return). TECO will respond by printing an asterisk at the left margin to indicate that it is ready to accept user commands. At this point, one or more commands may be typed at the terminal, and TECO will execute the commands upon receipt of two consecutive ESCAPE characters. The ESCAPE is a non-printing character which may be labelled ESC, ALT, or PREFIX on some keyboards. TECO echoes a dollar sign ($) whenever an ESCAPE is received. The dollar sign character is used in examples throughout this manual to represent ESCAPE.

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 might be used to indicate such things as the number of times a command should be executed. A text argument is a string of ASCII characters which might be words of text, for example, 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.

If more than one command is typed in response to the asterisk generated by TECO, the command string will be executed from left to right until either all commands have been executed or a command error is recognized. If an error is encountered, a message is printed and the rest of the command string is ignored. In any case, TECO prints another asterisk at the left margin as soon as it finishes execution of a command string, so that additional commands may be entered.

The extensive text editing capability of TECO implies a large and versatile command set. However, the novice TECO user will find that little more than a dozen basic commands suffice for most editing requirements. The following section introduces the basic TECO commands. The full command set will be described later in this manual.

1.1 FILE SELECTION COMMANDS

Input and output files may be specified to TECO in several ways. The following commands permit flexible file selection with TECO.

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.

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 write output text onto any 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 (e.g., a disk), a file name and extension (if any) 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, which has the same format as the ER 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 a dollar sign.
EWSY:TEXT.LST$	Prepare to write output file TEXT.LST on SY:.
EWPROG$	Prepare to write output file PROG on the system's default device.
EWDX1:TEXT.LST$	Prepare to write output file TEXT.LST on DX1:.

It is not always necessary to specify an input file. If the user desires to create a file without using any previously edited text as input, he 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 content of the buffer onto an output file. Since all input is supplied from the keyboard, no input file is necessary.

An output file is unnecessary if the user desires only to examine an input file, without making permanent changes or corrections. In this case, the content 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.

When the user is finished editing a file, he may 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. Note that the EX command takes no arguments.

COMMAND: EX

FUNCTION: Move the remainder of the current input file to the current output file, close the output file, then return to the monitor.

COMMAND STRING: ERDX1:INPUT.MAC$EWOUTPUT.MAC$$

FUNCTION: 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.

COMMAND STRING: ERFILE.MAC$EWCOPY.MAC$EX$$

FUNCTION: 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.

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. The EX command or one of the other file closing commands presented later may be used to close the output file.

1.2 INPUT AND OUTPUT COMMANDS

The following commands permit pages of text to be read into the TECO text buffer from an input device or written from the buffer onto an output device. 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 (see ED flag in 2.14).
P	Write the content 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 an integer in the range 1<=n<=65535. If n is not specified, a value of 1 is assumed.

1.3 POINTER POSITIONING COMMANDS

The buffer pointer provides the only means of specifying the location within a block of text at which insertions, deletions or corrections are to be made. The following commands permit the buffer pointer to be moved to a position between any two adjacent characters in the buffer. TECO positions the pointer before the first character in the buffer after every Y or P command.

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 may be any 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 line on which it is currently positioned.
C	Advance the pointer forward across one character.
nC	Execute the C command n times, where n must be an integer in the range -32768<=n<=32767. A positive value of n moves the pointer forward across n characters (carriage return/line feed counts as two characters). A negative value of n moves the pointer backward across n characters. If n is zero, the pointer position is not changed.

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 the command "-10000L" would position the pointer before the first character in the buffer. The command "10000L" would position the pointer after the last character in the buffer. No error message is printed in either case.

1.4 TYPE OUT COMMANDS

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

COMMAND	FUNCTION
T	Type the content 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 must be an integer in the range -32768<=n<=32767. 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 content of the buffer from the beginning of the line on which the pointer is located up to the pointer is typed. This facilitates locating the buffer pointer.
HT	Type the entire content of the text buffer.
V	Type the current line. Equivalent to the sequence "0TT".

The 0T command is particularly useful for determining the position of the buffer pointer. This command should be used frequently to determine that the pointer is actually located where the user expects it to be.

1.5 TEXT MODIFICATION COMMANDS

The following commands permit the user to insert or delete text from the buffer.

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, with the pointer positioned immediately after the last character of the insertion.
K	Delete the content of the text buffer from the current position of the pointer through and including the next line feed character.
nK	Execute the K command n times, where n may be any integer in the range -32768<=n<=32767. 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 content 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 content of the text buffer.
D	Delete the character following the buffer pointer.
nD	Execute the D command n times, where n may be any integer in the range -32768<=n<=32767. A positive value of n causes the n characters following the pointer to be deleted. A negative value of n causes the n characters preceding the pointer to be deleted. If n is zero, the command is ignored.

Like the L command, the D and K 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 across a page boundary will produce an error and the command is ignored.

1.6 SEARCH COMMANDS

The following commands may be used to search for a specified string of characters which may occur somewhere in the input file. They cause the buffer pointer to be positioned immediately after the last character in the specified string, if found.

COMMAND	FUNCTION
Stext$	Where "text" is a string of ASCII characters terminated with 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 and it is necessary to close the output file and reopen it as an input file before any further editing may be done on that file.

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 continues 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 content 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.

1.7 SUMMARY

At this point, the basic TECO commands have been introduced. Recall that TECO indicates it is ready to accept user commands by printing an asterisk (*). Once TECO has printed an asterisk, one or more commands may be typed at the terminal. Errors may be corrected by typing the DELETE key to delete characters. The DELETE key may be labeled DEL or RUBOUT on some keyboards. Each depression of the DELETE key deletes one character, beginning with the last character typed, and then prints the deleted character at the terminal. An entire command string may be deleted in this manner, if necessary. Once the correct command(s) have been entered, typing a double ESCAPE ($$) causes TECO to execute the command(s) in the order in which they were entered, and to print another asterisk so that additional commands may be typed. Note that this manner of operation is different from most other editors. In particular, carriage return has no special significance to TECO. Only the double ESCAPE forces execution of the command string.

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 self explanatory message relating to the command that generated the error. Every error message is followed by an asterisk at the left margin, indicating that TECO is ready to accept additional commands. If the first command entered after a TECO-generated error message is a single question mark character (?), TECO will print the erroneous command string up to and including the character which caused the error message. This facilitates locating errors in long command strings and determining how much of a command string was executed before the error was encountered.

At the conclusion of an editing job, the user may type EX to exit TECO. If an input and output file are open at the time the EX command is encountered, the remainder of the input file, including the current contents of the text buffer, is copied to the output file, and the output file is closed before TECO exits.

1.8 SAMPLE EDITING JOB

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

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 then 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 then typed -20L to move the pointer to the beginning of the buffer and SETTS$ to position the pointer immediately after the character string "ETTS" which terminates the word "MASSACHUSETTS". He then used an I command to insert one space and a five-digit zip code. A second S command positioned the pointer after the word "INFORMATION". The 2C command moved the pointer to the beginning of the next line (carriage return and line feed count two characters), and the user deleted the words "PERTAINING TO" and replaced them with the word "REGARDING".

4) The user continued editing by positioning the pointer after the word "GUIDE". He then deleted this word, and replaced it with the word "MANUAL". Finally, he searched for the word "SINCERELY", typed 0T to determine that the pointer was correctly positioned between the Y and the comma which follows it, and typed 0K to delete everything on the line except the comma. He then inserted "VERY TRULY YOURS" in place of the word "SINCERELY". An HT command caused the edited text to be printed at the terminal.

5) The command string EX$$ caused the content of the buffer to be written onto the output file and closed the output file. The user then reentered TECO and reopened the file "FILE1.TXT" as

an input file and specified the line printer as an output file.

6) This command string reads the first (and only) page of "FILE1.TXT" into the buffer, deleted the first 5 lines, replaced them with a different address and salutation, then printed the content of the buffer on the terminal for verification and finally printed the new version of the letter onto the line printer. Note that the previous version of the letter still resides in file "FILE1.TXT" on DT1.

```
1< *EWDT1:FILE1.TXT$$
2< *HKIMR. JOHN P. JONES
 ! COMPUTER ELECTRONICS CORPORATION
 ! BOSTON, MASAASACHUSETTS
 !
 ! DEAR MR. JONES:
 !
 ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
 ! PERTAINING TO THE NEW TECO-11 TEXT EDITING AND CORRECTING
 ! PROGRAM.
 !
 ! ENCLOSED IS A COPY OF THE TECO-11 USERS'S GUIDE, WHICH
 ! SHOULD ANSWER ALL OF YOUR QUESTIONS.
 !
 ! SINCERELY,
 !
 !
 !
 !
 ! $$
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-11 TEXT EDITING AND CORRECTING
 ! PROGRAM.
 !
 ! ENCLOSED IS A COPY OF THE TECO-11 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.
 ! 122 MAIN STREET WEST
 ! AUSTIN, TEXAS
 !
 ! DEAR MR. SMITH:
 ! $$
 ! *HT$$
 ! MR. JAMES B. SMITH
 ! DATEK ASSOCIATES, INC.
 ! 122 MAIN STREET WEST
 ! AUSTIN, TEXAS
 !
 ! DEAR MR. SMITH:
 !
 ! I WAS PLEASED TO RECEIVE YOUR REQUEST FOR INFORMATION
 ! REGARDING THE NEW TECO-11 TEXT EDITING AND CORRECTING
 ! PROGRAM.
 !
 ! ENCLOSED IS A COPY OF THE TECO-11 USER'S MANUAL, WHICH
 ! SHOULD ANSWER ALL OF YOUR QUESTIONS.
 !
 ! VERY TRULY YOURS,
 !
 !
 !
 !
 !
 ! *EX$$
```

CHAPTER II

COMMAND SUMMARY

The remainder of this manual is devoted to a detailed description of the full TECO command set. It is assumed that the reader is familiar with the elementary TECO commands presented earlier. The commands described in this chapter are the commands implemented in TECO version 27. Some of the I/O related commands may not be present under some operating systems.

2.0 TECO CHARACTER SET

TECO accepts the full 7-bit ASCII character set, which is presented in Appendix D. All characters have their 8th bit (the parity bit) trimmed off. Most terminals will not transmit and receive all of the ASCII codes; however, characters that are not available on the user's terminal may be inserted into the TECO text buffer by means of special commands which will be presented later in this chapter.

TECO command strings may be entered using upper case characters, as indicated throughout this manual, or by using the corresponding lower case characters. A file which contains upper and lower case text may be edited in the same manner as a file which contains only upper case text, although dealing with lower case text from an upper case only terminal is inconvenient. TECO normally converts lower case alphabetics to upper case as they are typed in. Commands to enable lower case type-in are presented later in this chapter.

TECO considers certain ASCII characters to have special meaning. Most of the special characters are immediate action commands. Typing these characters in a command string causes TECO to perform a specified function immediately, without waiting for the double ESCAPE which terminates the 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. For this reason, the special characters should not be used in text arguments, except where specifically indicated throughout this manual.

Table 2-0 lists the special characters, their functions and the restrictions associated with each character.

TABLE 2-0: RESTRICTIONS ON SPECIAL CHARACTERS

CHARACTER	RESTRICTION
ESCAPE	The ESCAPE character is a command terminator. It may not be used in the argument of any command except where noted specifically throughout this manual. TECO echoes a dollar sign when an ESCAPE is received. ESCAPE may be labelled ALTMODE or PREFIX on some terminals. A double ESCAPE initiates execution of the command string.
DELETE	Typing a DELETE character deletes the last character typed. The DELETE key may be labeled DEL or RUBOUT on some keyboards. Typing several consecutive DELETEs deletes one character for each DELETE typed beginning with the last character typed. TECO echoes the deleted character whenever a DELETE is typed (except in "scope" mode, see 2.14).
Control-C	Control-C, produced by striking the CONTROL key and the C key simultaneously. The action of the Control-C key depends on the operating system being used (See Appendices).
Control-U	Control-U, produced by striking the CONTROL key and the U key simultaneously, causes the current line of the current command line to be deleted. TECO echoes the character as ^U followed by carriage return and line feed (except in "scope" mode, see 2.14).
Control-G<Control-G>	Typing two consecutive Control-G characters, produced by holding the CONTROL key depressed while striking the G key twice, 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 Control-G character is not a special character.
Control-G<space>	Control-G, produced by striking the CONTROL key and the G key simultaneously, followed by a space, causes the line currently being input to be retyped.
Control-G<asterisk>	Control-G, produced by striking the CONTROL key and the G key simultaneously, followed by an asterisk, causes all the lines typed by the user from the last TECO prompt (the asterisk) to be retyped.

The Control-Z character is used as an end-of-file terminator in some contexts. 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).

TECO also attaches special significance to the carriage return, line feed, space, and null characters. A line feed is appended to every carriage return typed. Thus, it is necessary to type a carriage return and then a DELETE in order to enter a carriage return which is not followed by a line feed.

Carriage return, line feed, and space characters are ignored between commands in a command string; they may be inserted for clarity or convenience whenever necessary. The null character (control-shift-P or control-@) is ignored by all TECO input commands including the reading of data files.

Control characters which are not special characters (i.e., immediate action commands) may be included in the text argument of any TECO command. When used in this manner, the control character must be produced by striking the CONTROL key and a character key simultaneously. TECO will echo a caret (or uparrow) followed by the character which was typed whenever most control characters are entered; the others, such as control-L (form feed) or control-G (bell) echo as the function they perform.

Many control characters are also TECO commands. When a control character is entered as a command, it may be produced by striking the CONTROL key and the character key simultaneously or else by typing a caret (uparrow) followed by the desired character. This is advantageous because all control characters echo normally when typed in the caret/character format and it is essential that some TECO commands such as caret/C (^C) never be typed inadvertantly in their control format.

Several of the commands in TECO require a text string argument. The insert and search commands are examples of these. The TECO command language contains a general alternate form for all the commands which take a text argument as follows:

```
@command/text/
```

A delimiting character (shown as a slash here) must precede and follow the text argument. Command is the actual TECO command to be executed (e.g., I, S, ER, ^A, !). This alternate form is frequently used to imbed ESCAPE's in the text argument or to avoid the use of control characters as in the ^A command. Of course, the text argument itself may not contain the delimiting character.

2.1 FILE SPECIFICATION COMMANDS

An input file must be specified whenever TECO is requested to accept text from any source except the terminal. An output file must be specified whenever a permanent change is made to the input file. Input and output files are selected by means of file specification commands.

File specifications are operating system dependent. The Operating Characteristics Appendices at the end of this manual fully describe file specifications for each operating system.

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 only keep one input file and one output file selected at a time.

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

Table 2-1 lists the full file specification command set. Unless otherwise noted, all file specification commands leave the text buffer unchanged.

TABLE 2-1: FILE SPECIFICATION COMMANDS

COMMAND	FUNCTION
ERfilespec$	Opens a file for input on the currently selected input stream. The "filespec" is the file specification and "$" signifies an ESCAPE.
EWfilespec$	Opens a file for output on the currently selected output stream. The "filespec" is the file specification and "$" signifies an ESCAPE.
EBfilespec$	The EB command may be used for files on file-structured devices only. It opens file "filespec" 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 in a

	system dependent fashion (See Appendices).
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.
EC	Moves 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.
EX	Performs the same function as the EC command, but then exits from TECO.
EGtext$	Performs the same function as the EC command, but then exits from TECO and passes "text" to the operating system as a parameter (see Appendices).
Control-C	The Control-C (caret/C) command, when not executed from within a macro, causes an immediate abort and exit from TECO. Currently open files are not necessarily closed. See the Appendices. Section 2.11 describes the Control-C command when it is executed from within a macro.
EK	Kill the current output file on the currently selected output stream. This command purges the output file without closing it. This command is useful to abort an undesired edit from becomming permanent. 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.)
:ERfilespec$	Performs a similiar function as the ER command, but returns a numeric value. A -1 indicates success and 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.

:EBfilespec$	Perform a similiar function as the EB command, but returns a numeric value. See the :ER command.
EP	Switches the input to the secondary input stream. This command does not open or close any file and does not change the text buffer.
ER$	Switches the input to the primary input stream. This command does not open or close any file and does not change the text buffer.
EA	Switches the output to the secondary output stream. This command does not open or close any file and does not change the text buffer.
EW$	Switches the output to the primary output stream. This command does not open or close any file and does not change the text buffer.
EIfilespec$	Opens a file as an indirect command file. Any further TECO requests for terminal input will come from this file. At end-of-file, the file will be closed and terminal input will again come from the terminal. Any error message will also close the indirect command file and switch input 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. End-of-file in the indirect command file does not automatically start execution of commands. The indirect file must have two adjacent ESCAPEs to start its execution.
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 which has not yet been read is discarded. Otherwise, this command has no effect.
ENfilespec$	This command presets the "wild card" lookup filespec. This is the only filespec that can contain any wild card notations. See the Appendices for the

allowed wild fields in each operating system. This command is only a preset; it does not open, close or try to find any file.

EN$ — Each occurence of this command will load the filespec buffer with the next occurence of the preset wild card lookup filespec. The G* command (see 2.9) can be used to retrieve the fully expanded filespec. If no more occurences of the wild card filespec exist, the ?FNF error is returned.

:EN$ — Performs a similiar function as the EN$ command, but returns a numeric value. A -1 indicates 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 can use the string building characters described in Table 2-8A (see section 2.8). The Control-EQ* construct is especially useful.

Many editing jobs are most conveniently accomplished by using the EB (Edit Backup) command to open the designated input and output file, then terminating the job with either an EC command, which returns control to TECO, or an EX command, which exits from TECO. The EB command is recommended for normal editing.

2.2 PAGE MANIPULATION COMMANDS

In the sections following, the letters "m" and "n" are used in command formats to indicate numerical arguments. These may be either simple integers or expressions of arbitrary complexity (explained later).

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

TABLE 2-2: PAGE MANIPULATION COMMANDS

COMMAND	FUNCTION
A	Appends the next page of the input file to the content of the text buffer, thus combining the two pages of text on a single page with no intervening form feed character.
Y	Clears the text buffer and then reads the next page 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 2.14).
PW	Write the content 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 an integer in the range 1<=n<=65535.
m,nPW	Writes the content of the buffer from the m+1th character through and including the nth character onto the output file. m and n must be integers in the range 0<=n<=32767 and m should be less than n. A form feed is not appended to this output, nor is the buffer cleared. The pointer position remains unchanged.
HPW	Equivalent to the PW command except that a form feed character is not appended to the output. (See 2.13 for the definition of H.)
P	Writes the content 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.
nP	Executes the P command n times, where n must be an integer in the range 1<=n<=65535.

m,nP Equivalent to m,nPW.

HP Equivalent to HPW.

All of the input commands listed in Table 2-2 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 two thirds full. Special techniques for handling pages in excess of the buffer capacity will be developed later in this chapter.

2.3 BUFFER POINTER MANIPULATION COMMANDS

Table 2-3 summarizes the complete buffer pointer manipulation command set. 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 either 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 in this manner, the pointer is positioned at the boundary which would have been exceeded and no error message is printed.

TABLE 2-3: BUFFER POINTER MANIPULATION COMMANDS

COMMAND	FUNCTION
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.
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.
-C	Equivalent to -1C.

R — Moves the pointer backward across one character.

nR — Executes the R command n times. If n is positive, the pointer is moved backward across n characters. If n is negative, the pointer is moved forward across n characters. If n is zero, the position of the pointer is not changed.

-R — Equivalent to -1R.

L — Advances the pointer forward across the next line terminator[1] 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.

2.4 TEXT TYPE-OUT COMMANDS

Table 2-4 summarizes the commands which may be used to type out part or all of the content of the buffer for examination. These commands do not move the buffer pointer.

---

[1]. Line terminators are line feed, vertical tab, and form feed.

TABLE 2-4: TEXT TYPE-OUT COMMANDS

COMMAND	FUNCTION
T	Types out the content 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, the n lines following the current position of the pointer are typed. If n is negative, the n lines preceding the pointer are typed. If n is zero, the content of the buffer from the beginning of the line on which the pointer is located up to the pointer is typed.
-T	Equivalent to -1T.
m,nT	Types out the content of the buffer from the m+1th character through and including the nth character in the buffer. M should be less than n.
.,.+nT	Types out the n characters immediately following the buffer pointer. N should be greater than zero. (See 2.13 for the definition of .)
.-n,.T	Types the n characters immediately preceding the buffer pointer. N should be greater than zero (i.e., -n should be less than zero).
HT	Types out the entire content of the buffer.
V	Types out the current line. Equivalent to 0TT.
nV	Types out n-1 lines each side of the current line. Equivalent to 1-nTnT.
n^T	Types out the ASCII character whose value is n. This can be used to output any ASCII character to the terminal.
^Atext^A	Types "text" on the terminal. While the command may be entered as Control-A or Caret/A, the closing character must be a Control-A.
@^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 Control-A.

Users may stop the output of any typeout command by typing control-O at the keyboard. Typing control-O stops the output of the current command string. When used in this manner, the control-O must be entered while TECO is actually in the process of typing out at the terminal.

2.5 DELETION COMMANDS

Table 2-5 summarizes the text deletion commands, which permit deletion of single characters, groups of adjacent characters, single lines, or groups of adjacent lines.

TABLE 2-5: TEXT DELETION COMMANDS

COMMAND	FUNCTION
D	Delete the first character following the current position of the buffer pointer.
nD	Execute the D command n times. If n is positive, the n characters following the current pointer position are deleted. If n is negative, the n characters preceding the current pointer position are deleted. If n is zero, the command is ignored.
-D	Equivalent to -1D.
m,nD	Equivalent to m,nK.
K	Deletes the content of the buffer from the current position of the buffer pointer through and including the next line terminator character.
nK	Executes the K command n times. If n is positive, the n lines following the current pointer position are deleted. If n is negative, the n lines preceding the current pointer position are deleted. If n is zero, the content of the buffer from the beginning of the line on which the pointer is located up to the pointer is deleted.
-K	Equivalent to -1K.
m,nK	Deletes the content of the buffer from the m+1th character through and including the nth character. M should be less than n. The pointer moves to position m.
.,.+nK	Deletes the n characters immediately following the buffer pointer. N should be greater than zero.
.-n,.K	Deletes the n characters immediately preceeding the buffer pointer. N should be greater than zero.
HK	Deletes the entire contents of the buffer.

2.6 INSERTION COMMANDS

Table 2-6 lists the full text insertion command set. All text insertion 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 echo ^G (Bell) after each character that is typed. Attempting to enter too many characters into the current command string causes TECO to delete them as they are entered. Use of the DELETE key should be attempted to shorten the command to permit its termination.

With the exception of the nI$ command, insertion command arguments must not contain special characters (see Table 2-0). The nI$ command will insert any character into the buffer, including the special characters.

TABLE 2-6: TEXT INSERTION 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 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. (i.e. n is taken modulus 128.) It may be used to insert characters that are not available on the user's terminal or special characters such as DELETE which may not be inserted with the standard I command.
<tab>text$	Where <tab> is a tabulation, produced by striking the TAB key or by pressing the CONTROL key and the I key simultaneously. The TAB character echoes as one to eight spaces on most terminals. This command is equivalent to the I command except that the tabulation is part of the text which is inserted into the buffer.
@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. This delimiter may be any non-special character which does not appear in the insertion. This @ and alternate delimiter construct may be used in all commands which take a string argument.

n@I// Equivalent to the nI$ command, but does not require the ESCAPE character. Any delimiting character (shown as slash here) can be used.

FRtext$ Equivalent to "-nDItext$", where "n" is obtained from the most recent occurrence of the following: (a) the length of the most recent string found by a successful search command, (b) the length of the most recent text string inserted (including insertions from the FS, FN, or FR commands), or (c) the length of the string retrieved by the most recent "G" command. In effect, the last string inserted or found is replaced with "text", provided that the pointer has not been moved.

@FR/text/ Equivalent to "FRtext$", except that "text" may contain ESCAPE characters. The delimiter (shown as a slash here) must not be present in "text".

2.7 SEARCH COMMANDS

In many cases, the easiest way to position the buffer pointer is by means of a character string search. The 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 always begins at the current position of the pointer and proceeds either in a forward direction or in a reverse direction within the current buffer or in a forward direction through the file.

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

Normally searches are "unbounded". That is, they search from the current position to the end of the text buffer. A bounded search 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.

TABLE 2-7: 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 must be 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 and the pointer is positioned after the last character in the string.

-Stext$ — Equivalent to -1Stext$.

m,nStext$ — Performs the same function as the nS command, but m serves a bound limit for the search. If the search string can be found without moving the pointer more than ABS(m)-1 places, the search succeeds and the pointer is repositioned to immediately after the last character of the string. Otherwise, the pointer is left unchanged. (See 2.13 for a description of the ^Q operator. It converts line moving arguments to character moving arguments.)

0,nStext$ — Performs the same function as the nS command, but the pointer position will remain unchanged on search string failure. (Essentially an unbounded search with no pointer movement on failure.)

m,Stext$ — Identical to "m,1Stext$".

m,-nStext$ — Performs the same function as the m,nS command, but searches is the reverse direction.

m,-Stext$ — Identical to "m,-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 (backarrow) 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 ED flag in 2.14).

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.
FStext1$text2$	Executes an Stext1$ command, then deletes "text1" and replaces it with "text2".
nFStext1$text2$	Executes an nStext1$ command, then deletes "text1" and replaces it with "text2".
FNtext1$text2$	Executes an Ntext1$ command, then deletes "text1" and replaces it with "text2".
nFNtext1$text2$	Executes an nNtext1$ command, then deletes "text1" and replaces it with "text2".

The FS and FN commands above can also be reverse searches (n<0) or bounded searches (m,n argument).

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.

Any of the search commands listed above may be preceded by a colon (:). The colon is a search command modifier which supresses error message generation and causes the search command to "return a value" instead. That is, the next sequential command is executed with an argument of zero if the search fails. If the search succeeds, the next sequential command is executed with an argument of -1. If the next sequential command belongs to the class of commands which require a positive argument, the -1 is interpreted as a positive 65535. If the next sequential command does not require an argument, it is executed as it stands. If an unmodified search command in a command loop (section 2.10) is immediately followed by a semi-colon (section 2.12), it is treated as if it were a colon-modified search. 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 (or 65535, if it is a command which must have a positive argument). 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.

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

The @ character is another search command modifier. Inserting an @ character between the numeric argument of any search command and the command itself causes TECO to accept the first character following the command as a delimiting character which will also be the command terminator. This character may be any character which does not appear in the search command argument, except for special characters. When the @ command is used, search command arguments may contain ESCAPE characters. The following examples illustrate use of the @ command modifier.

COMMANDS:
```
n@S/text/
m,n@S/text/
n@N/text/
n@_/text/
n@FS/text1/text2/
m,n@FS/text1/text2/
n@FN/text1/text2/
   etc.
```

FUNCTION: In each case, execute the search command with text string "text" as an argument. This argument must be preceded and followed by a delimiting character which does not appear in the argument (a slash is

shown here). The search command argument (and the replacement string in FS and FN) may contain ESCAPE characters, as long as two consecutive ESCAPES are not struck.

Needless to say, these constructs may be combined.

COMMANDS:

```
n:@S/text/
m,n:@S/text/
n:@N/text/
n:@_/text/
n:@FS/textl/text2/
m,n:@FS/textl/text2/
n:@FN/textl/text2/
   etc.
```

2.8 SEARCH ARGUMENTS

TECO builds the search string by loading its search string buffer from the supplied search command argument. To facilitate the entry of special characters or frequently used character sequences, the argument may contain special string building characters. Table 2-8A lists the string building characters and their functions. (The notation Control-Ex means the Control-E character followed by the character x.)

TABLE 2-8A: STRING BUILDING CHARACTERS

CHARACTER	FUNCTION
^	A caret (uparrow) character in a search command argument indicates that the character following the caret is to be used as its control character equivalent (i.e. octal 100 to 137 is used as octal 0 to 37). Combination with other string building or match control characters is legal. For example, the combination Caret/Q/Caret is the same as Control-Q/Caret which means literally Caret. This string building character can be disabled by using the ED flag (see 2.14 and Appendices).
Control-Q	A control-Q character in a search command argument indicates that the character following the control-Q is to be used literally rather than as a match control character.
Control-^	A control-^ character in a search command argument indicates that the character following the control-^ 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).
Control-EQq	Control-EQq 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 2.9 below.
Control-EQ*	Control-EQ* indicates that the string stored in the filespec buffer is to be used in the position occupied by the ^EQ* in the search string. Q registers are discussed in section 2.9 below.
Control-EQ_	Control-EQ_ indicates that the string stored in the search string buffer is to be used in the position occupied by the ^EQ_ in the search string. Q registers are discussed in section 2.9 below.

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. These characters alter the usual matching process that occurs when a search is executed. Table 2-8B lists the match control characters and their functions.

TABLE 2-8B: MATCH CONTROL CHARACTERS

CHARACTER	FUNCTION
Control-X	A control-X character indicates that this position in the character string is unimportant. TECO accepts any character as a match for control-X.
Control-S	A control-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 control-S.
Control-N	TECO accepts any character as a match for the control-N/character combination EXCEPT the character which follows the control-N. Other combined Control-N/special character are legal also. For example, the combination Control-N/Control-ED means match anything except a digit in this position.
Control-EA	Control-EA indicates that any alphabetic character is acceptable in this position. TECO accepts any letter (upper or lower case A to Z) as a match for ^EA.
Control-EC	Control-EC indicates that any RAD50 character 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 ^EC.
Control-ED	Control-ED indicates that any digit is acceptable in this positon. TECO accepts any digit (0 to 9) as a match for ^ED.
Control-EL	Control-EL indicates that any line terminator is acceptable in the position occupied by ^EL in the search string. Line terminators are line feed, vertical tab, and form feed.
Control-ER	Control-ER indicates that any alphanumeric character is acceptable in this position. TECO

accepts any letter (upper or lower case A to Z) or digit (0 to 9) as a match for ^ER.

Control-ES — Control-ES indicates that any non-null string of spaces and/or tabs is acceptable in the position occupied by ^ES.

Control-EX — Equivalent to ^X.

2.9 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. In the number storage area, each Q-register can store one integer in the range -32768<=n<=32767. In the text storage area, each Q-register 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. In this manual, a Q-register name is indicated by a lower case "q", which stands for any one of the 36 Q-registers.

Table 2-9A lists the commands which permit characters to be loaded into the Q-registers.

TABLE 2-9A: Q-REGISTER LOADING COMMANDS

COMMAND	FUNCTION
^Uqstring$	This command inserts character string "string" into the text storage area of Q-register "q". ^U must be specified in the caret/U format, since the control-U character is the line erase character.
@^Uq/string/	Equivalent to the ^U command except that the character string to be inserted into Q-register q may contain ESCAPE characters. The insertion must be delimited before and after by any character (a slash is shown here) which does not appear in the insertion.
nUq	Put n in the number storage area of Q-register q.
n%q	Add n to the content 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.
%q	Equivalent to 1%q.
nXq	Move n lines into Q-register q, where n is an integer in the range -32768<=n<=32767. 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 content 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.
Xq	Equivalent to 1Xq.
-Xq	Equivalent to -1Xq.
m,nXq	Copy the content 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.

]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.
:]q	Just like the]q command except that a numeric value is returned. 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 (and Q-register q was not modified).

Table 2-9B lists the commands which permit characters to be retrieved from the Q-registers.

TABLE 2-9B: Q-REGISTER RETRIEVAL COMMANDS

COMMAND	FUNCTION
Gq	Copy the content 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 content of the text storage area of Q-register q on the terminal. The text buffer and buffer pointer are not changed by this command.
Qq	Use the integer stored in the number storage area of Q-register q as the argument of the next command.
Mq	Execute the content 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 content of the text storage area of Q-register q as a command string and use n as a numeric argument for the first command in this string.
m,nMq	Execute the content of the text storage area of Q-register q as a command string and use m,n as a numeric argument for the first command in this

string.

[q — Copy the content 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. This command does not use numeric values. Numeric values are passed through this command as if it did not occur. This allows 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.

G* — Get most recent filespec string. The asterisk is not actually a Q-register but represents TECO's filespec string area. Copy the contents of the filespec string area into the buffer at the current position of the buffer pointer, leaving the pointer positioned after the last character copied. The filespec string area contains the fully expanded filespec of the last E command (see Appendices).

:G* — Print the contents of the filespec buffer on the terminal.

G_ — Get most recent search string. The underscore (backarrow) is not actually a Q-register but represents TECO's search string area. Copy the contents of the search string area into the buffer at the current position of the buffer pointer, leaving the pointer positioned after the last character copied.

:G_ — Print the contents of the search string buffer on the terminal.

2.10 COMMAND LOOPS

The user may 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 which designates the number of iterations. Iterated command strings are called command loops. Loops may be nested in such a way that one command loop contains another command loop, which, in turn, contains other command loops, and so on. The maximum depth to which command loops may be nested is determined by the size of TECO's push-down list (system dependent), but is always greater than 10.

The general form of the command loop is:

n<command string>

where "command string" is the sequence of commands to be iterated and n is the number of iterations. If n is not supplied then no limit is placed on the number of iterations. If n is 0 or less than 0 then the iteration is not executed at all; command control skips to the closing angle bracket. If n is greater than 0 (1 through 32767) then the iteration is done n times.

Search commands inside command loops are treated specially. If a search command which is not preceded by a colon modifier is executed within a command loop and the search fails, a warning message is printed, the command loop is exited immediately and the command following the right angle brackets 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 (section 2.12), it is treated as if it were a colon-modified search.

2.11 BRANCHING COMMANDS

TECO commands may be combined in sophisticated command strings which are capable of solving even the most complex editing problems. In this respect, TECO should be considered a programming language which accepts an input file as data and processes this input to produce an output file. As with most programming languages, TECO provides an unconditional branch command and a set of conditional execution commands.

To provide for branching within a command string, there must be some means of naming locations inside the string. TECO permits location tags which have 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 for special characters. 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 named location elsewhere within the command string and "$" signifies an ESCAPE. When an O command is executed, the next command to be executed will be the command following the tag referenced by the O command, and command execution continues normally from this point.

The use of the O command is subject to two important restrictions. 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.

Secondly, 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.

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

A special purpose branching command is Control-C (caret/C). If the Control-C command is executed from within a macro, the execution of TECO commands is aborted and TECO returns to the top level (asterisk prompt). No cleanup of push-down lists, flag settings, etc. is done. This command allows a macro to abort TECO's command execution.

2.12 CONDITIONAL EXECUTION COMMANDS

All conditonal execution commands are of the form:

n"Gcommand string'

where "n" is a numeric argument on which the decision is based, "G" may be any of the conditional execution commands listed in table 2-12, and "command string" is the command string which will be executed if the condition is satisfied. If the condition on n is not satisfied, the command string will not be executed. Note that the numeric argument is separated from the conditional execution command by a double quote (") and the command string is terminated with an apostrophe (').

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 2-12 lists the conditional execution commands. Each conditional execution command must be followed by a command string (not shown in Table 2-12) which will be executed only if the condition is satisfied. This command string must be terminated by an apostophe. If the condition is not satisfied, the first command following the apostrophe will be the next command executed.

TABLE 2-12: CONDITIONAL EXECUTION COMMANDS

COMMAND	FUNCTION
n"G or n">	Execute the following command string (terminated by an apostrophe) if n is greater than zero.
n"L or n"<	Execute the following command string (terminated by an apostrophe) if n is less than zero.
n"E	Execute the following command string (terminated by an apostrophe) if n is equal to zero.
n"C	Execute the following command string (terminated by an apostrophe) if n is the ASCII code of any character which is one of the upper or lower case letters A to Z, one of the digits 0 to 9, or period, or dollar sign.
n"N	Execute the following command string (terminated by an apostrophe) if n is not equal to zero.
n"T	Execute the following command string (terminated by an apostrophe) if n is TRUE. Equivalent to n"L.

n"S Execute the following command string (terminated by an apostrophe) if n is SUCCESSFUL. Equivalent to n"L.

n"F Execute the following command string (terminated by an apostrophe) if n is FALSE. Equivalent to n"E.

n"U Execute the following command string (terminated by an apostrophe) if n is UNSUCCESSFUL. Equivalent to n"E.

n"A Execute the following command string (terminated by an apostrophe) if n equals the ASCII code for an alphabetic character (upper or lower case A to Z).

n"V Execute the following command string (terminated by an apostrophe) if n equals the ASCII code for an upper case alphabetic character (upper case A to Z).

n"W Execute the following command string (terminated by an apostrophe) if n equals the ASCII code for a lower case alphabetic character (lower case A to Z).

n"D Execute the following command string (terminated by an apostrophe) if n equals the ASCII code for a digit (0 to 9).

n"R Execute the following command string (terminated by an apostrophe) if n equals the ASCII code for an alphanumeric (upper or lower case A to Z or 0 to 9).

There is one further conditional execution command which is not related to the commands listed in Table 2-12. The n; command, where n is any integer or numeric expression, may be used in an iterated command loop. It has the general form:

m<string1 n; string2>string3

where "m" is the iteration count, "string1", "string2", and "string3" are command strings and "n;" is the conditional exit command. When the n; command is executed, it will cause TECO to exit the command loop so that "string3" will be executed next if n is greater than or equal to zero. If n is less than zero, the loop is not exited and "string2" is executed next. N may be any argument such as Qq (the value of the numeric part of Q-register q), a colon-modified search, or any other command that returns a value.

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.

2.13 NUMERIC ARGUMENTS

Almost all TECO commands may be preceded by a numeric argument which generally indicates a buffer pointer, an ASCII character value, the number of iterations, or how many times the command should be executed. Some numeric arguments must be positive, while others may be negative or zero. In any case, every numeric argument is stored as a single 16-bit word.

This leads to an important restriction on the maximum size of any numeric argument. Commands which require positive arguments must have an argument in the range 0<=n<=65535, since 65535 is the largest number which may be stored in one 16-bit word. Commands which have positive or negative arguments require an argument in the range -32768<=n<=32767, because -32768 is the smallest number which may be stored in 16 bits using 2's complement notation, while 32767 is the largest number which may be stored in this manner.

TECO maintains several internal variables which record conditions within TECO. Each of the variables has a name which is equivalent to the current contents of the variable. These characters may be entered as numeric arguments to TECO commands. When the command is executed, the current value of the designated variable is substituted for the character and used in 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 (twice) for the "dot". The addition is then carried out, and the command is executed as though it were of the form "m,nT".

Table 2-13A lists all of the characters which have special numeric values. Any of these characters may be used as numeric arguments in place of the values they represent.

TABLE 2-13A: CHARACTERS ASSOCIATED WITH NUMERIC QUANTITIES

CHARACTER	FUNCTION
B	Always equivalent to zero. Thus, B represents the position at the beginning of the buffer, preceding the first character in the buffer.
Z	Equivalent to the number of characters currently contained in the buffer. Thus, Z represents the position at the end of the buffer, 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. For example, 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.
Mq	The Mq command (execute the content 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	:Qq is equivalent to the number of characters in the text storage area of Q-register q.
\	A backslash character which is not preceded by a numeric argument is equivalent to the value of the digit string (if any) that begins with the character immediately following the buffer pointer and is terminated by the next character that is not a valid digit for the current radix. The first character may be a digit or + or -. As the backslash command is evaluated, TECO moves the buffer pointer to a position immediately following the digit string. If there is no digit string following the pointer, the backslash is equivalent to zero and the pointer position remains unchanged. The digits 8 and 9 will stop the evaulation if TECO's current radix is octal.
ED	ED is equivalent to the current value of the edit level

flag. The use of the ED flag is described in section 2.14.

EH — EH is equivalent to the current value of the help level flag. The use of the EH flag is described in section 2.14.

ES — ES is equivalent to the current value of the search verification flag. The use of the ES flag is described in section 2.14.

ET — ET is equivalent to the current value of the type-out control flag. The use of ET flag is described in section 2.14.

EU — EU is equivalent to the current value of the upper lower case flag. The use of the EU flag is described in section 2.14.

EV — EV is equivalent to the current value of the edit verify flag. The use of the EV flag is described in section 2.14.

^B — Control-B (caret/B) is equivalent to the current date via the following equations:

```
RT-11:    ^B = (((month*32)+day)*32)+year-72
RSTS/E:   ^B = ((year-1970)*1000)+day within year
RSX-11:   ^B = ((year-1900)*16+month)*32+day
```

^E — Control-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 content of the buffer on output.

^F — Control-F (caret/F) is equivalent to the current value of the console switch register.

^H — Control-H (caret/H) is equivalent to the current time of day via the following equations:

```
RT-11:    ^H = 0
RSTS/E:   ^H = minutes until midnight
RSX-11:   ^H = (seconds since midnight)/2
```

^N — Control-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 Control-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., one simply types "^SC".

^T Control-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 2.14) for variations.

^V Control-V (caret/V) is equivalent to the version number of the version of TECO which is currently being run. This manual describes TECO version 27.

^X Control-X (caret/X) is equivalent to the current value of the search mode flag. The use of the ^X flag is described in section 2.14.

^Y Control-Y (caret/Y) is equivalent to ".+^S,.". This is the n,m numeric argument spanning the text just searched for or inserted. For example, this value may be used to recover from inserting a string in the wrong place. One types "^YXAFR$" to store the string in Q-register A and remove it from the buffer. One may then position the pointer to the right place and type "GA" to insert the string.

^Z Control-Z (caret/Z) is equivalent to the number of characters presently stored in the Q-register storage area, including requirements for the command string currently being executed.

^^x The combination of the Control-Caret (double caret or double uparrow) followed by any character is equivalent to the value of the ASCII code for the character. The "x" in this example may be any character that can be typed in to TECO.

The numeric argument of a TECO command may consist of a single integer, any of the characters listed in Table 2-13A, the numeric contents of any Q-register, or an arithmetic combination of integers, Q-registers, and the characters listed in Table 2-13A. If an arithmetic expression is supplied as a numeric argument, TECO will evaluate the expression. All arithmetic expressions are evaluated from left to right. 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 performing operations outside the parentheses. Parentheses may be nested, in which case the innermost expression contained by parentheses will be evaluated first. Table 2-13B lists all of the arithmetic operators that may be used in arithmetic expressions.

TABLE 2-13B: 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.
^Q	n^QC=nL	The number of characters between the buffer pointer and the nth line separator (both positive and negative). This operator converts line oriented command argument values into character oriented argument values. Used after an expression.

The following commands affect the radix in which TECO processes numbers, which may be either octal or decimal. All numbers typed in, typed out, or handled by the \ and n\ commands are interpreted in the current radix.

TABLE 2-13C: 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. TECO's radix is unaltered.
n==	This command causes the value of n to be output at the terminal in octal followed by a carriage return and line feed. Octal numeric conversion is unsigned. For example, the unsigned number 65535 (decimal) will output as 177777. TECO's radix is unaltered.
n:=	This command causes the value of n to be output at the terminal in decimal and leaves the carriage positioned at the end of the output.
n:==	This command causes the value of n to be output at the terminal in octal and leaves the carriage positioned at the end of the output.
^O	Control-O (caret/O) causes all subsequent numeric input to be accepted as octal numbers. The digits 8 and 9 become illegal as numeric characters when the radix is octal. Numeric conversions using the \ or n\ commands will also be octal. The octal radix will continue to be used until the next ^D command is executed.
^D	Control-D (caret/D) causes all subsequent numeric input to be accepted as decimal numbers. This is the initial setting.
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) or unsigned octal (octal radix). Note that \ is a "bidirectional" command. n\ inserts a string into text while \ (no argument) returns a numeric result.

Some TECO commands generate numeric values which they pass on to subsequent commands. An example is any colon-modified search command, which causes the next sequential command to be executed with an argument of -1 or 0, depending upon the outcome of the search. Commands of this sort are very useful, but occasionally it may be undesirable to have arguments passed in this manner. Note that many TECO commands perform significantly different functions, depending on whether no, one, or two arguments are present. A single ESCAPE character may be inserted between any two commands in a command string to absorb unwanted values. This ESCAPE has no effect on the individual commands, however a numeric argument will never be passed across an extraneous ESCAPE.

2.14 MODE CONTROL FLAGS

TECO has flags which control various aspects of its operation. These flags are referenced with the commands "^X", "ED", "EH", "ES", "ET", and "EU", "EV", and are called the ^X, ED, EH, ES, ET, EU, and EV flags. A flag's current setting may be interrogated 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 with a preceding numerical argument; the flag is set to the value of the argument. The flags have the following functions:

TABLE 2-14: MODE CONTROL FLAGS

COMMAND	FUNCTION
n^X	Sets the search mode flag to n. 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 "@", "[", "\", "]", "^" respectively. If n 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.
nED	Sets the value of the edit level flag to n. If n is -1 the Y (Yank) command and _ (underscore or backarrow) command work unconditionally as described earlier in the manual. If n is +1 or 0 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 from bringing in successive pages if an output file is open.

If n is -1 or 0 the character caret in a search argument will convert the character following it to a control character. If n is +1 the character caret has no special meaning in search arguments.

The initial value of ED is system dependent (see Appendices).

nEH Sets the value of the help level flag to n.

If n is equal to 1, error messages are output in abbreviated form ("?XXX"). If n is equal to 2, error messages are output in normal form ("?XXX Message"). If n is equal to 3, error messages are output in normal form, then the failing command is 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.)

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

nES Sets the value of the search verification flag to n.

If n is equal to 0, nothing is typed out after searches. If n is -1, the current line is typed out when a search 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 more than one line of type-out is desired, the form m*256+n can be used. The n is the same as above. The m is the number of lines of view. For example, 3*256+^^! would be 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.

The initial value of ES is 0.

nET Sets the type out control flag to n.

The ET flag is a bit-encoded word that controls TECO's treatment of the console terminal. Any combination of the individual bits may be set as the user sees fit. The bits have the following functions:

Bit 0 (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 is suppressed. This mode is useful for driving displays. It should be used with caution, especially if the user is talking to TECO over a dial-up line.

Bit 1 (2) Process DELETEs and Control-Us 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.

Bit 2 (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).

Bit 3 (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.

Bit 4 (16) Cancel Control-O on typeout. Setting this bit will cancel any outstanding Control-O when the next typeout occurs. After TECO has canceled the Control-O, it will automatically reset the bit.

Bit 5 (32) Read with no wait. This enables the ^T command to test to see 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.

Bit 6 (64) Detach flag (see Appendices).

Bit 7 (128) When this bit is set: 1) all informational messages are supressed, 2) any Control-C causes the immediate termination of TECO, and 3) any error causes the termination of TECO after the error message is printed.

Bit 8 (256) If this bit is set, all lines output to the terminal are truncated to the terminal's width if needed. (RSTS/E and RSX-11 only.)

Bit 15 (32768) If this bit is set and a Control-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 Control-Cs.

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.

nEU Sets the value of the upper/lower case flag to n.

If n is -1, no case flagging of any type is performed on typeout, 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 operating system dependent (see Appendices).

nEV Sets the value of the edit verify flag to n.

The edit verify flag's value 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.

2.15 PROGRAMMING AIDS

Text Formatting

The characters carriage return, line feed, vertical tab, and space are ignored in command strings, except when they appear as part of a text argument. 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.

Comments

One of the most powerful features of TECO is the 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!
```

An important pitfall to avoid is the use of tab characters to format long command strings. Only space, carriage return, and line feed must be used to format command strings since TAB is an insertion command.

Messages

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

```
^Atext^A
```

or

@^A/text/

where 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 control-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 occurence 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^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.

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 (TECO prompts with *).

Command Editing

If an error is typed while entering a command string, the error may be corrected at any time before the double ESCAPE which terminates the command string is typed. Characters may be deleted individually by striking the DELETE key. Each depression of the DELETE key deletes one character, beginning with the last character typed, and causes the deleted character to be printed at the terminal. An entire line of a command string may be deleted by typing control-U. If an entire command string is deleted in one of these manners, TECO responds by printing a new asterisk at the left margin.

Typing Control-G<space> causes TECO to retype the line currently being input. Typing Control-G<asterisk> causes TECO to retype all the lines from the last command execution point. The caret form of the Control-G may not be used for these commands.

Typing two successive control-G characters causes the current command string to be erased completely. The double control-G command must be produced by holding the control key depressed while striking the G key twice (if the terminal has a bell, it will ring). The caret form of the control-G may not be used for this command.

After successful command completion, after an error, or after ^G^G to cancel a command, the last command buffer contents

may be recovered by immediately typing "*q", where "q" is any Q-register name. This places the command buffer contents into the named Q-register. No ESCAPE is required after the *q.

2.16 ERROR MESSAGES

When TECO encounters an illegal command or a command that cannot be executed, an error message is printed at the terminal. Error messages are of the form:

?XXX Message

where "XXX" is an error code and "Message" is an explanatory message. When an error message is generated, the command to which it refers is not executed, the rest of the current command string is ignored, and TECO prints an asterisk at the left margin to indicate that it is ready to accept further commands.

In some cases it may be difficult to determine which command in a long command string resulted in an error message. Typing a question mark immediately after the TECO-generated error message causes TECO to print the current command string up to and including the erroneous character. When used in this manner, the question mark must be the first character typed after the error message is printed. It is not necessary to follow the question mark with an ESCAPE.

2.17 MANIPULATING LARGE PAGES

TECO is designed to operate most efficiently when editing files that contain no more than several thousand characters per page.[1] If any page of an input file is too large to fit in the text area, the various 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. One may make room by positioning the pointer past a portion of text at the beginning of the buffer and moving it out with the commands

```
0,.PW0,.K
```

---

[1]. 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.

2.18 TECHNIQUES AND EXAMPLES

TECO may be used in three ways. The most elementary application, described in Chapter I of this manual, involves using TECO to create and edit 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, an entire job may be accomplished with one long command string consisting of all of the short command strings placed end to end with the intervening double ESCAPE characters removed. This leads to the concept of a TECO editing program, which is simply a long command string that performs a certain editing task. 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 (where "q" is the Q-register name).

This is fine for clear-cut editing assignments, such as converting from one format to another or editing certain characters out of a file, but many editing jobs are so complex that a given editing program will only solve a small class of problems. The solution, in this case, is to write specialized "editing subroutines." TECO subroutines might 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, the right combination of subroutines may be loaded into various Q-registers, augmented with additional commands if necessary, and called by a "mainline" command string.

Editing subroutines are essentially macros; that is, sequences of commands which perform commonly required editing functions. Thus the third and most powerful application of TECO is the creation and use of a macro library. As each editing job is undertaken, the user may look for sequences of operations which might be required in future editing assignments. All of the TECO commands required to perform such an operation may be loaded into a Q-register and executed by means of an Mq or nMq command. When the job is finished, the content of any Q-register which contains a useful macro may be written onto an output file (via the buffer) and saved in the macro library. The nMq and m,nMq commands, which were designed to facilitate use of macros, permits 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 (decimal) or 8 (octal) into Q-reg 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 and saves Q-register storage space. A TECO command file retrieved with an EI command may be in one of two basic forms: The file may contain a TECO command that loads the macro into a Q-register for later use. Or, the file may simply be the macro itself (in which case it must be retrieved with EI each time it is used).

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

EXAMPLE 1: SPLITTING, MERGING, AND REARRANGING FILES

Assume that the user has a file named PROGRAM.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.)

	Call TECO.
*-1ED$$	Allow all Y commands.
*EBPROG.DAT$Y$$	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 content is overwritten).
*Y$$	Delete MN and read OP
*J20X3$$	Save all of O in Q-register 3.
*20K$$	Delete O from the buffer.
*P$$	Write P onto the output file, leaving the buffer cleared (the input file is exhausted).
*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 mentioned earlier, containing data in the form:

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

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

	Call TECO.
*-1ED$$	Allow all Y commands.
*ERFILE$EWFILE1$$	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.
*EX$$	Close the output file and exit.

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

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

and the other is MATH.TWO, which contains:

MN <FF> OP <FF> QR

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

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

	Call TECO.
*-1ED$$	Allow all Y commands.
*ERDK1:MATH.TWO$$	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.

*HPEX$$	Write KL onto the output file (without appending a form feed) and close the file, then exit.

EXAMPLE 2: ALPHABETIZING BY BINARY SEARCH

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. This command string groups 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 binary 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 @O/CONT/ '           !Loop back if there is another line
                                 in the buffer !
QZ"G 0UZ @O/START/ '            !Repeat if a switch was made on the
                                 last pass !
$$
```

APPENDIX A

RT-11 OPERATING CHARACERISTICS

Startup

TECO is started with the

```
.R TECO
```

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

The EDIT command

```
.EDIT/TECO filespec
```

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

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

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

```
.SET EDITOR TECO
```

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

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

```
.EDIT filespec
```

can be used to edit an already existing file.

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

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

Another option, /EXECUTE, is also available:

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

The /EXECUTE option causes TECO to process the filespec (assumed

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

```
.R TECO
*ERfilespec$YHXZHKIstring$MZ$$
```

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

Startup Conditions

The initial value of the ET flag is 0.

The initial value of the EU flag is -1.

The initial value of the ED flag is 0.

File Specification

The file access commands ER, EB, 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, 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.

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.

Exit and Go

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

Control-C

The action taken when the user types Control-C depends on what TECO is doing. At command level Control-C is a valid input character and has no special meaning. Note though, that if TECO executes Control-C as a command from command level, TECO is aborted.

If TECO is executing commands or doing I/O, a double Control-C will stop the operation and generate the ?XAB error message.

A ^T command within a macro can read a Control-C as its input character, but another Control-C will normally abort the macro.

Sometimes it is desireable for a TECO macro to detect when a double Control-C was typed. By detecting the double Control-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 double Control-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
```

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.

File Recovery

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

```
ERdev:$
```

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

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.

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. TECO provides two commands to interact with the display:

Command	Function
^W	Cause an immediate display screen update with the current window into the text buffer. This command allows the creation of interactive display screen macros.
n^W	Cause TECO to display the n lines preceding and following the current position of the pointer on the display screen. The window size is set to n.

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 B

RSTS/E OPERATING CHARACERISTICS

Startup

TECO is started with the

```
RUN $TECO
```

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

The CCL command

```
TECO filespec
```

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

```
RUN $TECO
*EBfilespec$Y$$
```

The CCL command

```
MAKE filespec
```

is used to create a new file with TECO. It is equivalent to

```
RUN $TECO
*EWfilespec$$
```

One more CCL command exists for TECO. It is

```
MUNG filespec
     --or--
MUNG filespec,text
```

This is equivalent to

```
RUN $TECO
*Itext$EIfilespec$$
```

In other words, the MUNG command will process the filespec as a TECO indirect command file passing an argument of text in the text buffer. This is a convienent way to invoke TECO macros.

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+3K initially (TECO's default startup size is 3K).

Startup Conditions

The initial value of the ED flag is always 0.

When TECO is initially invoked it will automatically set the ET and EU flags as described below.

The ET flag is set to 0 for non-scope terminals without lower case input, to 2 for scope terminals without lower case input, to 4 for non-scope terminals with lower case input, and to 6 for scope terminals with lower case input. Note: The actual ET flag value will be 128, 130, 132, or 134 (Bit 7 on in addition to the above) when TECO initially starts. TECO automatically clears Bit 7 every time it reaches prompt (*) level.

The EU flag is set to 0 (flag lower case) for upper case only terminals. It is set to -1 (no flagging) for lower case terminals.

File Specification

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

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

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

The file specification switches /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.

Editing BASIC-PLUS Programs

The line feed, carriage return, null combination that signals a continuation line to BASIC-PLUS can cause problems when those files are edited with TECO. To overcome this, TECO has a special "BASIC-PLUS file" edit mode. Simply append a slash to the file specification.

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

This changes TECO's handling for file input and/or output as follows.

On input, TECO will strip off and ignore all nulls (Octal 0, Decimal 0) and carriage returns (Octal 15, Decimal 13). For every line feed (Octal 12, Decimal 12), TECO will automatically insert a carriage return preceeding the line feed. In this way the text buffer has all lines ending with the carriage return, line feed combination. Editing of the text buffer is now easy.

On output, TECO will ignore all carriage returns. For every line feed TECO do one of the following:

1) If the character two positions before the line feed was the character ampersand (&), TECO will output carriage return and line feed. In other words, the text buffer sequence &, CR, LF will be output unchanged. This is the BASIC-PLUS "EXTEND" mode convention for line continuation.

2) If the character following the line feed is a digit (i.e. the start of a line number) or no characters follow the line feed, TECO will output a carriage return, line feed.

3) If the character following the line feed is not a digit (start continuation line with a tab to ensure this) or the line feed is the last character but a form feed is to be output, TECO will output a line feed, carriage return, null sequence.

Editing RSX Run-Time System Generated Files

TECO will correctly read RSX Run-Time System generated files of type 1 (fixed length records), type 2 (variable length records), and type 4 (ASCII stream). All types are converted to ASCII stream format in the text buffer. All TECO output files are ASCII stream.

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.

Wild Card Lookup

The EN command will process wild card lookups on RSTS/E. To preset the wild card lookup file specification, use the standard RSTS/E format

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

The device name must reference a file structured disk device, be the public structure (SY:), or be omitted which defaults to the public structure. The [p,pn] field cannot be wild. It defaults to the user's current logged in account. The filename field must be specified and can be explicit, fully wild (i.e. *), or partially wild (i.e. containing one or more ?'s). If the .ext field is omitted, only files with no extension will be looked for. Otherwise, the extension field can be explicit, fully wild (*), or partially wild (?'s).

Exit and Go

If TECO is exited via the EGstring$ command, the most recently created output file's extension is checked. If the extension matches an extension in TECO's built in table, the associated CCL command is executed.

```
.MAC for command "MACRO string"
.RNO for command "RNO string"
.CTL for command "SUBMIT string"
.FOR for command "FORTRAN string"
```

If no match occurs, TECO simply exits.

Control-C

The action taken when the user types Control-C depends on what TECO is doing.

If TECO is executing commands, the ?XAB error occurs.

If TECO is in the midst of doing I/O, the error "?ERR Programmable ^C trap" occurs.

If TECO is at command level or asking for a character with the ^T command, TECO simply restarts and reprompts its asterisk.

If two Control-Cs are typed to TECO when it is at command level, it will exit.

Sometimes it is desireable for a TECO macro to detect when a Control-C was typed. By detecting the Control-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 Control-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
```

ET Flag Handling

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

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

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.

Installing TECO

The following commands will install TECO on your RSTS/E V06B

system. (Note: dev: is your distribution medium.)

```
RUN $PIP
(PIP's header line)
#$TECO.TEC/CO:T=dev:$TECO.TEC
#[0,1]TECO.RTS/CL:32/CO:T=dev:[0,1]TECO.RTS
#=$TECO.TEC<232>/RE
#^Z

RUN $UTILTY
(UTILTY's header line)
? NAME TECO=$TECO.TEC
? ^Z
```

Inclusion of TECO on System Startup

The following commands will include TECO on your RSTS/E V06B system at system startup time.

```
RUN $UTILTY
(UTILTY's header line)
? ADD TECO
? CCL MAK-E=$TECO.TEC;[PRIV ]n
? CCL MU-NG=$TECO.TEC;[PRIV ]n
? CCL TE-CO=$TECO.TEC;[PRIV ]n
? ^Z
```

The n above is a memory usage limit. If n is 0 then the TECO will expand its memory on demand up to the user's memory limit. If n is non-zero then TECO will not expand its memory over nK. N will be 0 for most installations.

If PRIV is included, TECO will expand its memory beyond the user's private memory limit (but never beyond the limit set by n if any). Most installations will not include PRIV in their CCL definitions.

APPENDIX C

RSX-11 OPERATING CHARACTERISTICS

Startup

TECO is started with the

```
TECO
```

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

The command

```
TECO filespec
```

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

```
TECO
*EBfilespec$Y$$
```

The command

```
TECO filespec1=filespec2
```

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

```
TECO
*ERfilespec2$EWfilespec1$Y$$
```

The command

```
MAKE filespec
```

is used to create a new file with TECO. It is equivalent to

```
TECO
*EWfilespec$$
```

One more command exists for TECO. It is

```
MUNG filespec
     --or--
MUNG filespec,text
```

This is equivalent to

```
TECO
*Itext$EIfilespec$$
```

In other words, the MUNG command will process the filespec as a TECO indirect command file passing an argument of text in the text buffer. This is a convienent way to invoke TECO macros.

If the MUNG command is not installed, TECO macros may also be invoked with the command

```
TECO @filespec
```

It is equivalent to

```
TECO
*EIfilespec$$
```

In systems supporting dynamic task expansion, TECO will expand its buffer space as necessary, up to a limit defined in the task build command file. (As supplied, this limit is roughly 7K words.) Also, TECO'S buffer space may be explicitly allocated in the startup command

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

If a larger text buffer is needed than the expansion limit, TECO should be started up with a /INC allocation greater than the limit; it will then expand to the maximum size the system will allow.

Startup Conditions

The initial value of the ED flag is always 1.

When TECO is initially invoked it will automatically set the ET and EU flags according to the user's terminal characteristics. If the terminal supports CRT style rubouts, then bit 1 of the ET flag is set to do the same in TECO. If the terminal supports lower case type in, then bit 2 of the ET flag is set and the EU flag is set to -1 to turn off case flagging. Thus the initial value of the ET flag will be either 0, 2, 4 or 6. 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.

File Specification

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

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

in which dev: is a physical device name or a logical device name; if dev: is not specified, SY: is assumed. If [p,pn] is

not specified, the user's current default directory is assumed. The filename field must be specified whenever the device name references a file structured device. The typ field is a file type and must be explicitly given if used. There is no default type except for EI commands which default the .typ field to .TEC.

The 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* (returning 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 one opens a file and then supersedes it, one should not switch the input channel away from it with an EP or ER$ command, since it

will not be possible to open the file again.

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.

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. To protect against accidental loss of text typed in, the EX command will not allow TECO to exit if there is text in the buffer and no open output file. To exit after just looking at a file, one must use the sequence HKEX.

The EG command is identical in action to the EX command, since RSX-11 has no facility for passing control from one program to another.

The Control-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 Control-C command is roughly equivalent to EKHKEX.

Control-C

The action taken when the user types Control-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, the control-C is simply entered into the command string. Note that execution of the Control-C command causes instant exit from a main level command string.

Sometimes it is desireable for a TECO macro to detect when a Control-C was typed. By detecting the Control-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 Control-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 Control-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.

ET Flag Handling

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

In addition, TECO always turns off Bit 7 (exit on error, etc.) every time 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 Control-C interrupts. Setting bit 6 of the ET flag causes TECO to run with the terminal detached. All commands function normally, except that typing Control-C causes the MCR to be activated, allowing other tasks to be run from the same terminal concurrently with TECO. It is, of course, the user's problem to sort out the confusion that will arise if both TECO and another task request input from the terminal at the same time.

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. The one exception to this processing is a record containing just a form feed. This is interpreted as an end of page mark; it stops the read operation but is not entered in the buffer. 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 sequences. Each carriage return / line feed delimits the end of an output record. If the output file is implied or Fortran carriage control, the carriage return and line feed are not output with the record; if the file is internal carriage control, they are. Form feed characters in the buffer are always output as a single character record. Note that solitary carriage returns and line feeds, and line feed / carriage return sequences do not delimit records, but remain embedded in the records being output.

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

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 Control-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.

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 uses of TECO.TEC are limited only by the user's imagination. They range from simply loading a few utility macros into Q-registers, through interpreting and perhaps modifying the command line (available in Q-register Z) or the command processing macro (in Q-register Y), to loading and starting up a higher level editing macro which handles the command line itself and takes its own commands from the terminal.

Installing TECO

The first step in installing TECO is to copy the files from the distribution medium into an available directory on the system disk. TECO distributed on Magtape or DECtape is in DOS format and must be read with FLX; disks are in Files-11 format. The command line thus is of the form

```
FLX SY:=MT:[200,200]*.*
```

or

```
PIP SY:=DK1:[200,200]*.*
```

One must then build TECO with the command

```
TKB @TECBLD
```

Note that TECO uses the Utility Library. In RSX-11M this is located in the file [1,20]PIPUTL.OLB in the system distribution kit; in RSX-11D it is located in the file [11,5]UTLLIB.OLB. These must be available to build TECO. Users with unmapped RSX-11M systems will need to edit the TKB command file to alter the PARTITION directive to fit their system. RSX-11M users should also note that the size of TECO is about 7.5K words,

excluding its text buffer. Thus, TECO requires a minimum partition of about 10K words to be useful.

Once task built, TECO needs to be installed under three names to make all of the commands available:

```
INS TEC              default name is ...TEC
INS TEC/TASK=...MAK for the MAKE command
INS TEC/TASK=...MUN for the MUNG command
```

In IAS, TECO is installed instead under the task names $$$TEC, $$$MAK, and $$$MUN to implement the three commands.

TECO makes use of some of the advanced features of RSX-11M V3 and RSX-11D V6.2, such as dynamic task expansion, the read with special terminators terminal function (i.e., TECO mode), and unsolicited character AST's. TECO may be reconfigured to do without some of these features to allow it to run on subset or older systems; instructions on this are to be found in the assembly and TKB command files, and in the assembly prefix file TECPRE.MAC.

APPENDIX D

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
^H	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
ALT	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 E

ERROR MESSAGES

TECO error messages consist of a three letter message preceded by a question mark. 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 also produces two warning messages. These messages do not abort the command and execution continues.

%Superseding existing file
: Indicates that the file to be created as the result of an EW command already exists. If the output file is closed the old copy of the file will be deleted. The EK command may be used to "take back" the EW command.

%Search fail in iter
: Indicates that a search command has failed inside iteration brackets. A ; (semi-colon) command immediately following the search command can typically be used to suppress this message.

ERROR MESSAGES

?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.)

?CPQ Can't pop Q-reg
: A] command has been executed and there is nothing saved on the Q-register push down list.

?DEV Invalid device (RT-11 only)
: A file specification string in an E command contains an unknown device name.

?DIO Directory I/O error (RT-11 only)
: A file creation command has probably been attempted to a write locked device.

?DTB Delete too big
A D 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.

?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 file full (RT-11 only)
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 then be concatenated when the space problem is alleviated.

?ICC Illegal " character
One of the valid " commands did not follow the ". Refer to Section 2.12 (conditional exectution commands) for the legal set of commands.

?IEC Illegal E character
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 F character
An invalid F command has been executed. The only valid F commands are FN, FR, and FS.

?IFN Error in file name (RT-11 only)
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
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 (RT-11 only)
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, the user may have to return to a backup file.

?IQR Illegal Q-reg
An illegal Q-register name was specified in one of the Q-register commands.

?ISA Illegal search arg
The argument preceding a search command is 0. This argument must not be 0.

?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. (Section 2.17.)

?MFN Missing file name (RT-11 only)
An ER or EW command has been executed in which the file name has been left out.

?MRP Missing)
A command string has been executed which contains a parenthetical expression that is not closed by a right parenthesis.

?NAB No arg before ^_
The ^_ command must be preceded by either a specific 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 value.

?NFI No file for input
Before issuing an input command, such as Y, it is necessary to open an input file by use of a command such as ER or EB.

?NFO No file for output
Before issuing an output command such as N search or P it is necessary to open an output file by use of a command such as EW or EB.

?NRO No room for output (RT-11 only)
The output device is too full to accept the requested output file.

?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 (RT-11 only)
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.

?PDO Push-down list overflow
The command string has become too complex. Simplify it.

?POP Pointer off page
A J, C or R command has been executed which attempted to move the pointer off the page. The result of executing one of these commands must leave the pointer between 0 and Z, inclusive.

?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.

?UEA EA not implemented (RT-11 only)
The EA command has no meaning in this version of TECO.

?UEI Edit Indirect not implemented (RT-11 only)
The EI command has no meaning in this version of TECO.

?UEN Edit Next not implemented (RT-11 only)
The EN command has no meaning in this version of TECO.

?UEP EP not implemented (RT-11 only)
The EP command has no meaning in this version of TECO.

?UTC Unterminated command
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 Control-C.

?YCA Y command aborted
An attempt has been made to execute an Y or search command with an output file open, that would cause text in the text buffer to be erased without outputting it to the output file. The ED command (section 2.14) 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 F

INDEX TO TECO COMMANDS AND SPECIAL CHARACTERS

Command Char	Page	Function
ESC	16	String and command terminator.
^\ (FS)	-	Not a TECO command.
^] (GS)	-	Not a TECO command.
^^x (RS)	47	ASCII value of next char in command string.
^^x (RS)	34	(String build char) Use next char as lower case.
^_ (US)	48	Ones complement.
SP	17	Ignored in commands.
!	41	Define label.
"	42	Start conditional.
n"A	43	Test for alphabetic.
n"C	42	Test for RAD50.
n"D	43	Test for digit.
n"E	42	Test for equal to zero.
n"F	43	Test for false.
n"G	42	Test for greater than zero.
n"L	42	Test for less than zero.
n"N	42	Test for not equal to zero.
n"R	43	Test for alphanumeric.
n"S	43	Test for successful.
n"T	42	Test for true.
n"U	43	Test for unsuccessful.
n"V	43	Test for upper case alphabetic.
n"W	43	Test for lower case alphabetic.
n">	42	Test for greater than zero.
n"<	42	Test for less than zero.
#	48	Logical OR.
$	-	Not a TECO command. (Dollar symbol not ESCAPE.)
n%q	37	Add n to Q-register q.
&	48	Logical AND.
'	42	End conditional.
(	47	Expression grouping.
)	47	Expression grouping.
*	48	Multiplication.
*q	56	Put last command in Q-register q.
+	48	Addition.
,	-	Argument separator.
-	48	Subtraction or negation.
.	45	Current pointer position
/	48	Division.
0-9	-	Digit.
:	31	Make next command return a value.
::	32	Make next search a compare.
:=	49	Type in decimal, no carriage return.
:==	49	Type in octal, no carriage return.
:Gq	38	Type Q-register q on terminal.
:Qq	45	Size of text in Q-register q.

Command Char	Page	Function
n;	43	Exit iteration if n>=0.
n<	40	Iterate n times.
=	49	Type in decimal.
==	49	Type in octal.
>	40	End iteration.
?	55	Toggle trace mode.
?	56	Type command string up to error.
@	17	Use alternate string delimiters.
A	22	Append to buffer.
nA	45	ASCII value of char in buffer.
B	45	0 - beginning of buffer.
nC	23	Advance n characters.
nD	26	Delete n characters.
EA	20	Select secondary output stream.
EB	18	Open input and output.
EC	19	Close out (copy input to output and close).
ED	50	Edit level flag.
EF	19	Close output file.
EG	19	Close out and exit.
EH	51	Edit help level.
EI	20	Open indirect command file.
EK	19	Kill output file.
EN	20	Wildcard lookup.
EP	20	Select secondary input stream.
ER	18	Open input file.
ES	51	Search verification flag.
ET	52	Type out control flag.
EU	53	Case flagging flag.
EV	53	Edit verify flag.
EW	18	Open output file.
EX	19	Close out and exit.
nFN	31	Global string replace.
FR	28	Replace last string.
nFS	31	Local string replace.
m,nFS	31	Bounded local string replace.
Gq	38	Get string in Q-register q into buffer.
G*	39	Get last filespec string into buffer.
G_	39	Get last search string into buffer.
H	45	Equivalent to "B,Z".
I	27	Insert text.
nI	27	Insert ASCII character "n".
nJ	23	Move pointer to "n".
nK	26	Kill n lines.
m,nK	26	Delete between m and n.
nL	24	Advance n lines.
Mq	38	Execute string in Q-register q.
nN	30	Global search.
O	41	Go to label.

Command Char	Page	Function
nP	22	Advance n pages.
m,nP	23	Write out chars m to n.
nPW	22	Write buffer n times.
m,nPW	22	Write out chars m to n.
Qq	38	Number in Q-register q.
nR	24	Back up n characters.
nS	29	Local search.
m,nS	30	Bounded local search.
nT	25	Type n lines.
m,nT	25	Type from m to n.
nUq	37	Put number n in Q-register q.
nV	25	Type n current lines.
W	-	Not a TECO command.
nXq	37	Put n lines into Q-register q.
m,nXq	37	Put characters m to n into Q-register q.
Y	22	Read into buffer.
Z	45	End of buffer value.
[q	39	Q-register push.
\	45	Value of digit string in buffer.
n\	49	Convert n to digits in buffer.
]q	38	Q-register pop.
^	17	Interpret next command char as a control char.
^	34	Interpret next search char as a control char.
n_	31	Global search without output.
^_	-	Not a TECO command.
a-z		Treated the same as upper case A-Z.
{	-	Not a TECO command.
|	-	Not a TECO command.
}	-	Not a TECO command.
~	-	Not a TECO command.
DEL	16	Delete last character typed in.