



the

# MOBY MUNGER

*The official organ of the TECO Special Interest Group*

---

Issue #3 - Part 2

November 1979

---

Contributions and correspondence should be sent to the editor:

Stanley Rabinowitz  
6 Country Club Lane  
Merrimack, NH 03054

or to

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

Media intended for the newsletter editor should be sent to his work address:

Stanley Rabinowitz  
Mail Stop MK1-2/E09  
Digital Equipment Corporation  
Continental Blvd  
Merrimack, NH 03054

Camera ready copy is preferred but scribblings on the side of a napkin will be accepted. Ready to use material should be prepared on 8 1/2 x 11 inch white unlined bond paper. A one inch margin should be maintained on both sides, on the top, and on the bottom. Material should be reasonably clean, legible, and sufficiently dark copy for printing. Material intended for publication should be mailed in a large envelope so that it can be sent without the necessity of folding.

Readers are urged to submit articles, techniques, notes, hacks, ideas, comments, and bug reports to the Moby Munger for publication. This newsletter will publish on a more regular basis if more readers will submit material. It would be especially helpful if some people would volunteer to write certain columns regularly, such as the Beginner's column, or the TECO Techniques, or the NOTES FROM XXX, etc. Volunteers are urged to send their names in to the editor.

©1979, DECUS

It is assumed that all articles submitted to the editor of this newsletter are with the authors' permission to publish in any DECUS publication. The articles are the responsibility of the authors and, therefore, DECUS, Digital Equipment Corporation, and the editor assume no responsibility or liability for articles or information appearing in the document.

## NOTES FROM THE EDITOR

This is a continuation of the August issue. The index for this issue is contained in that issue. The entire issue was ready in July; however when I gave DECUS a 74-page issue, they told me that the TECO SIG was only budgeted for two 40-page issues a year. Consequently, I decided to split the issue up into two parts. Since the 'year' ends around June, this means that this will be the last issue you see for 6 months unless we can get our budget changed. We'll talk about this problem at the SIG meeting in San Diego. Part of the problem is that we had no chairman at the time of budgeting; so DECUS assumed the number of issues we wanted this year would be the same as the number we published the previous year. We wouldn't be in this problem if we had a strong leadership. Also, I was given the option of reducing the text 50% (thus printing two pages on one side of a sheet). That would have allowed us to publish another two issues. However, I did not like the looks of other newsletters that have done such a photoreduction; so without a steering committee to discuss such policy, I made the decision on my own not to go to half size. If you want more to say on such topics, volunteer to be on the SIG's steering committee.

This is the only page in this newsletter to be typed in November; the rest was typed in June. As of November 1, TECO-11 and TECO-10 are still not in the DECUS library. I don't know what the problem is; both have been done and working for over 5 months now. In any case, copies will be down at the symposium (I will hand carry them there if no one else does).

Only two solutions submitted so far for the current problem of the Month (the macro that locates itself). Deadline is not yet up though.

The symposium itself looks pretty good for TECO people. We have a lot of interesting stuff. There are a few errors in the official program; but nothing serious. The introductory TECO tutorial will be given by Pat Hall and J. Hayes. All the TECO tutorials this year are being well-prepared in advance. They should all be good - not our usual last-minute affairs. We were supposed to have two EMACS tutorials. The SIG wanted this very much - we believe EMACS to be the up-and-coming great new concept. As of November 1, I have not heard from DECUS why these tutorials did not make it into the program. If possible, I would hope to still get them scheduled (as birds of a feather session if necessary). The SIG business meeting will also be a TECO roadmap session; so be there to here about any changes in the schedule. Because meeting rooms are suppose to be locked up at 11 PM, DECUS would not let me officially schedule a TECO night owls of a feather session for midnight Thursday night; but I hope to still hold the session either in our campground (if it's open) or in my hotel room.

- Stan -

EMACS is a display editor. This means, first of all, that you see the result of each command as soon as you have finished typing it. We try to make this true of the smallest possible units of typing. So, for example, there is no "insert" command: to insert a character, you just type it, if it is a printing character (there is a command to quote other characters). As soon as you have typed it, it appears on the screen in the right place. If you make a mistake, just type Rubout, and the character before the cursor will disappear. Other commands will move backward or forward over characters, up or down lines or to the beginning or end of the current line, to the beginning or end of the buffer, etc. This much applies to all display editors.

Any display editor at all provides a tremendous advantage to the user over printing-terminal editors. Most users of display editors (EMACS, TVEDIT, Stanford ETV) find that they no longer need to make paper listings. Also, it becomes very easy to compose new programs on line.

However, EMACS offers the additional feature of self-documentation. Not only are there the obvious commands which ask what a given character does; there is the command "Apropos" which can tell you about all commands which have a given string in their names. If you have forgotten how to read in a file, just do `^X Apropos^File`. This will tell you about all commands whose names contain "File", one of which will be the one you seek. Additionally, there is an EMACS subsystem which is used for perusing tree-structured documentation files. It is called INFO; one way of entering it is to type "?" when EMACS starts up. INFO offers a tutorial on how to use INFO, as well as documentation on EMACS and whatever else you care to add to the tree.

What distinguishes EMACS from all other display editors is that it is conveniently programmable: it contains an interpreter for a powerful programming language (called TECO, but much more suitable for system programming than DEC TECO. See the appendix). This makes it very easy for new commands to be written. An individual user can add a few commands of his own, or an ambitious person can write a sharable library of useful commands which everyone can use. True, since any editor is a program, if you have the source you can modify it, but in practice a much greater sophistication is possible in EMACS programs than in assembler-language modifications to TECO.

For example, you can program EMACS to understand the syntax of your favorite programming language. At MIT we have found useful commands for editing Assembler language, text (and input to text justifiers), Lisp, TECO code (for EMACS), and PL/1. The user can select a "major mode" such as "Text Mode", "Lisp Mode", etc., and each mode redefines a few commands to be appropriate to the language (a user or site can add new major modes). In addition, there are several comment manipulation commands which can be used for any language, because the major mode tells them what strings are used to begin and end comments in the language you are using.

In "MIDAS Mode" (our assembler is called MIDAS), there are commands for moving to the next or the previous label, and for moving to the AC field or address field of an instruction. For Lisp, we have commands which indent code according to its nesting, which move over or up and down in list structure, which go to the beginning or end of the function you are working on, etc. For PL/1, there is a very hairy indentation program which figures what sort of statement or compound statement you are in the middle of and indents the line appropriately. The techniques used for PL/1 can easily be adapted to most other algol-like languages.

For editing text, we have commands that understand sentences, paragraphs, and words, as well as commands to fill and justify text. Also, there is "Auto Fill" mode, in which lines are automatically broken when you type space, to fit a specified margin. I am using auto fill mode now. For text justifiers, we have commands to insert and delete the text justifier commands that specify font selection and underlining.

To save the user from having to select the right major mode by hand, each file can specify the major mode to be used to edit it. The string "-\*-Text-\*-" at the front of this file has that purpose. It is also possible to specify in a file the value of any of the many options and parameters that control EMACS. The values thus specified will apply whenever EMACS is looking at that file. Also, a user can specify in his start-up file that certain special options should be in effect whenever he (and only he) is in a given major mode. For example, my start-up says that auto fill should be on whenever I am in text mode. If I read in this file, auto fill is turned on, but if you read it in, you won't be in auto fill unless you ask for it.

You can edit two windows at once. The screen is split across the middle, and the top and bottom can show two different files or two parts of the same file. You might want to put a list of tasks, or a FILECOM, in one part of the screen, while making the changes in the other half.

With large systems split up into many files it can be hard to remember what is in each file. With the EMACS TAGS package you don't have to. It remembers all the location of each function definition or label in one or several files, and will jump directly to the definition of the function whose name you specify, switching files as necessary. TAGS uses a special file called a "tag table", which you must construct by calling a program provided for that purpose. Since it contains the character-position in the file of each tag, tags can be found without searching the whole file. The tag table is not updated automatically when the file is edited, but this is not a problem, because a small inaccuracy mandates searching only a small part of the file, not the whole thing.

One of the most avid EMACS users at MIT has written a "word abbreviation" package. If I define "pk" to be an abbreviation for "package", then I can just type "pk" and it will expand automatically. If I type "PK", it will expand to "PACKAGE". "Pk" will expand to "Package". Some people have large files of abbreviation definitions which they always use.

Word abbreviation mode is one example of a sharable EMACS library written by a user. Several users at MIT have written libraries for general use, including the PL/1 feature described above. Every library contains documentation for each command in it, so that the self-documenting features continue to work.

EMACS is available to anyone for use on Tuenex, on a communal sharing basis. Any generally useful improvements that you make, you must give back to MIT to share with everyone else, just as we have shared with you. If you plan to make improvements, you should get in touch with MIT and other sites so that you can find out how to make them as widely useful as possible. Of course, whoever is actually willing to do the work has more say than everyone else, but it is his responsibility to consider the needs of all users.

If you would like to talk with other Tuenex users of EMACS, you can contact the Computer Corporation of America, 575 Tech Square, Cambridge, MA 02139, or Geoff Goodfellow at the Stanford Research Institute. There are also users at DEC (I hear that the maintainer of DEC PDP-10 TECO is one of them).

Some of you may have tried programming in DEC TECO. It is very tedious, since you can't have very many variables or very many subroutines, and all error conditions must be detected explicitly.

MIT TECO provides arbitrarily long names for variables and subroutines, thus allowing reasonable programming practices to be followed.

In addition, errors can be handled in several ways. First of all, you can put an error-catcher around some commands; an error inside will return to the catcher, also returning a pointer to a string containing the error message. Second of all, the user can supply an error handler which will be called in the environment of the error. In EMACS, this error handler allows you to look at the buffer and at the stack of macro calls. Finally, if you don't do anything, the local binding stack (known in former times as the q-register pdl) will be unbound, so that global values of variables are never lost. Without these error handling features, it would be impossible to write reliable system programs.

Also, data structures have been made more flexible and efficient. It may work even in DEC TECO to do Q1U2 to move a string from Q1 to Q2. This is because strings are represented by pointers. In MIT TECO we have taken advantage of this to pass pointers to strings much of the time. In addition, you can have pointers to other sorts of objects: buffers, for example (there can be any number of them), or vectors of other objects (good for symbol tables, etc). Every buffer has a moveable "gap" which is invisible to the user; this is used to make insertion and deletion in a local region of the buffer much faster. Insertion just uses up gap, while deletion increases the gap. Of course, you must move the gap to where you want to insert or delete, but that's better than moving half the buffer for each insertion or deletion. The ability to use a temporary buffer for intermediate results reduces core shuffling even more. As a result, when you edit a small part of a large file, only the part you are editing needs to be swapped in.

Finally, MIT TECO contains an integral display processor designed to display an up-to-date picture of the buffer, using as few characters as possible to update on whatever sort of terminal you have.

The best testimony to the effectiveness of MIT TECO as a system programming language is that about 8,000 lines of code have been written and debugged using it.

Historical note:

PDP-10 TECO (actually, PDP-6 TECO) was originally written at the MIT Artificial Intelligence Laboratory in 1964 by Greenblatt, Nelson and Holloway. Some time later, DEC took a copy, but development of TECO at MIT did not cease. In 1973, I added the real-time display processor, and the ability for each character to run an arbitrary program. Immediately, many users began to write packages of such programs: TECMAC, RMODE, DOC, MACROS, TMACS, Russ-mode, ... This explosion of TECO program development allowed many ideas for useful commands and command languages to be tested. Until 1976, I played no part in this work, being content to add to TECO the features requested by the maintainers of the subsystems. EMACS was begun in 1976, drawing on what had been learned from all the other TECO program packages, as well as borrowing the code for some complicated commands. Some of the older programs have been supplanted entirely; some have been re-implemented as customizations of EMACS to save their old users from having to re-learn the command set: RMODE and DOC are

## RANDOM IDEAS

by Stanley Rabinowitz

The following are my (subjective) ideas for possible future enhancements to TECO (TECO-11 in particular). Comments, improvements, and other suggestions are welcome.

1. m,n"E could mean execute the following commands if m=n.
2. m,n"N could mean execute the following commands if m≠n.
3. "X could mean execute the following commands if a numeric argument exists.
4. n"#X or perhaps n:"X (where X represents some conditional) could mean execute the following commands if n is NOT satisfied by condition X. For example, n"#D would mean execute the following commands if n is not the ASCII code for a digit.
5. Multiple I/O channels could be implemented in a more consistent and mnemonic manner than the current scheme as follows: nER\$ could mean switch to input channel n, and nEW\$ could mean switch to output channel n. Thereafter, ER's and EW's without a numeric argument would work on the current channel. In addition, nERfile\$ and nEWfile\$ would open a file for input (or output) on the specified channel.
6. n↑Atext(↑A) would read input from the terminal allowing normal editing facilities. "text" would be used as a prompt. Input is inserted into the text buffer before dot and is terminated when the character whose ASCII code is n is entered. If n=0, then input is terminated by crlf. If n=-1, then input is terminated by \$\$\$. Until the input was terminated, normal editing features, such as rubout and CTRL/U would be permitted.
7. nEC could tell TECO to shrink or expand to nK bytes (words?) of memory. ØEC would mean to shrink to the smallest possible size.
8. The match control construct CTRL/EM could mean search for multiple occurrences of the next match control construct (as in TECO-1Ø).
9. :Otag\$ could be a subroutine call. Control would transfer to the specified tag and upon execution of some as-yet undetermined command which would correspond to a RETURN command, control would resume following the :Otag\$ command. Such a construct would greatly enhance the easibility of writing good structured TECO code. The command to return from subroutine might be something like :O\$ although, technically, this should mean call the subroutine whose name is the null string (represented by a !! tag ).
- 1Ø. Frequently, at the beginning of a complicated macro, it is necessary to push away a lot of Q-registers and at the end of the macro, to pop them back. Such situations could be simplified by commands like [<A,B,C,D,...> and ]<A,B,C,D,...> which would push and pop a set of Q-registers, respectively.

In the last issue of the Moby Munger, the editor presented an editorial chiding the differences between TV and TECO. The following article is a rebuttal by Eric Osman, who reviews some of the features of TV and argues that the merits of TV override its incompatibility with TECO. Further comments, both pro and con are welcome.

## ABOUT TV

by Eric Osman

In the March 1979 issue of the MOBY MUNGER, some mention was made to TV and its relation to TECO. As the father of TV, I would like to shed some further light on TV.

TV is a TECO that runs on TOPS20. It features a window feature that shows the current buffer portion being edited, with a cursor to indicate where the pointer is. This is particularly useful on video terminals, on which TV knows how to update only the parts of the window that have changed, when a complete repaint of the screen can be avoided. Note, however, that the window feature is also available on hardcopy terminals.

I arrived at DEC five years ago and immediately was disappointed that no video TECO existed. I proceeded to create TV from VTECO. VTECO is the BBN TECO that was running on TOPS20 when I arrived at DEC.

Here are some of the major points about TV:

### BACKUP SYSTEM:

Every fifty keystrokes, every character the user has typed is appended to a file called COMMANDS.TV. If the system crashes and is restarted, the TV user can recover his work, losing at most fifty characters, merely by issuing a command telling TV to read the contents of the file as typein.

This method has the following advantages over some others:

- o Standard TECO has no automatic backup. Hence if the system crashes, you lose lots of work, unless you continually remember to "save your file" periodically. TV lets you forget about such worries and just type!
- o Some editors feature an auto-save feature, which saves a copy of the entire file periodically. This has several problems. First of all, if it's a large file, you may not have disk space for the extra copies. Also, saving an entire copy is very expensive. TV's saving of only your keystrokes during the current editing session uses much less disk space and effort.

- o Another method of backup is to map the buffer to a file. This has the problem that after a crash, you have to figure out exactly when in relation to your typing, the system crashed, so you can know which keystrokes "made it".

#### FORGETTING TO TYPE I:

All of us TECO users have at some point forgotten to type I at the beginning of an insertion string. After hitting two escapes at the end of the command string and getting an error, TECO lets you recover if you type asterisk followed by a letter. Unfortunately, if you typed one non-asterisk, you aren't given the option of deleting it and typing the asterisk. TV has the ;G (semicolon G) command which inserts the last command string into the buffer that was longer than fifteen characters. Hence short command strings or extra keystrokes following a long insertion that failed don't prevent you from recovering your typing.

#### LARGE BUFFER AND HOLE:

Instead of the standard TECO method of editing page by page, TV works most conveniently if you issue the ;Y command which yanks an entire file into the buffer, edit it, and then use ;X to exit.

This avoids several pitfalls of TECO. For instance, an inadvertent Y command will give you an error, since no file is open! In many TECOs, inadvertent Y commands cause the current page to be erased and replaced by another page! Another pitfall avoided is that of misspelling a search string. In TECO, if you say Nreciev\$, TECO shuffles through all the pages of the file, then gives an error, since you meant Nreceiv\$. Unfortunately, as minimal recovery effort, you must reopen the file. In TV, you would merely reissue the failing search, since no file is open, so the N would not clear the buffer or read in more data.

Most TECOs allow the entire file to be edited in the buffer at one time. However, this is grossly slow, since every insertion or deletion requires the "rest" of the buffer to be slid over to make room for the insertion, or close the gap after a deletion. TV solves this problem with a hole, or bubble, in the buffer. This bubble floats to the location of an insertion or deletion. A deletion causes the deleted characters to be absorbed by the bubble. An insertion causes the bubble to shrink. Hence the amount of data necessary to be shuffled is reduced to the distance between one insertion or deletion to the next. For instance, the following command string, which inserts a carriage return after every sentence, could easily run a hundred



times faster in TV than TECO:

```
J<:S.$; I
s>$$
```

This is because every time TECO inserts the next crlf, the entire buffer to the right of where the crlf is being put must be moved right two places. In TV, only the sentence and its period need be moved.

#### INCOMPATIBILITIES WITH TECO:

TV is quite similar to TECO, and anyone starting to use TOPS20 who is familiar with TECO, can learn TV by merely knowing a short list of differences.

Most of the incompatibilities are due to the fact that TV was created from VTECO, which is BBN's TECO, and already had the incompatibilities.

Although having a compatible TECO is desirable, a number of the differences, I believe, are better in TV than TECO. For instance, having the file commands be ;R and ;W etc. instead of ER and EW reduces the chances of runaway insertions causing irreparable damage. (A runaway insertion is one missing the I before it).

Another incompatibility is that searches in iterations are not assumed to be "colon" searches in TV, as they are in TECO. This is nice because in TECO if you forget a semicolon after your search command in an iteration, you get into an infinite loop. In TV, you'll get an error.

Other incompatibilities could conceivably be resolved.

All new commands that I have added to TV, I've tried to be compatible with TECO. For instance, TV now has most of the control-E search features.

The original BBN VTECO, from which TV was created, started command execution when one escape was typed instead of the two that TECO waits for. I modified TV to wait for two, as TECO does. This was done in response to many people complaining that they accidentally typed escape in the middle of a command string (such as in an iteration) and execution started prematurely. In BBN VTECO, you type CTRL/D, which echoes as \$, when you need a non-command-terminating-escape.

## FUTURE OF TV:

Although TV shows the editing buffer on the video screen, TV can not be used as a true video editor very conveniently. The true video editor allows keystrokes to cause immediate effect on the screen. For instance, if the pointer is in the middle of a line, and you type X to the true video editor, the X immediately appears in the line. Likewise, a subsequent DELETE character would cause the X to disappear, and the rest of the line would instantly slide left to fill the gap.

True video editors also allow the user to associate keystrokes with macros. For instance, you may want to be able to type CTRL/S to save the current sentence. Then you move the pointer to a new location and type CTRL/V to insert the saved sentence here. This is only one of an infinite number of examples.

A good example of a video editor is EMACS developed at MIT.

Either TV will grow the ability to do what EMACS does, or EMACS will become the popular editor. It is not clear yet which of these will happen. However, true video editors are definitely the easiest editors to use.

---

DIRECT.TEC - Installation Procedure (continued from page 20)

To invoke DIRECT.TEC, you put the following equate in your login command file:

```
DIR:=='MUNG' [loc]DIRECT,
```

where [loc] is the name of the directory in which DIRECT.TEC resides. Note that the command ends with a comma and must occur after the definition of MUNG.

To use DIRECT, you would issue a command of the form

```
DIR filespec
```

where "filespec" may contain wildcard '\*'s, '?'s, or '%'s. For example: DIR A\*B.FOR or DIR [\*]FOO??X.\* . "?" and "%" are equivalent.

# LIBERTY MUTUAL



Research Center, 71 Frankland Road, Hopkinton, Massachusetts 01748 • Tel. (617) 435-3452

Dear Stan:

April 11, 1979

In the most recent Moby Munger (#2, March 79) you published a commentary from Tom McIntyre on the advantages and disadvantages of KED and TECO. I think Tom did a good job of comparing the two from the editor designer's point of view, but he did not say much about some of the issues that I find most compelling when it comes down to picking an editor.

The most significant of these issues for me are:

1. TECO exists (as you have extensively documented) in compatible versions on all the DEC mainframes and operating systems. It seems very important here to be able to move from RT-11 to RSX-11M to RSTS to OS/8 to OS/78 without having to learn several different editors. Only masochists and gurus do that. It is particularly difficult to get compatible editors when going from -8 based systems to -11 based systems. Compatible TECOs are either distributed free with the systems or are available from DECUS at low cost for all these systems. On the other hand, KED is not. It may or may not be available for RT-11. I have had trouble finding out about cost, etc. Somewhere down the road it might be announced for the other 16/32 bit systems although I know of no DEC commitment. I expect it to be at least a few years before it is available on all the -11 systems we use. Even then we still have the issue of cost. As far as I know, there is no expectation of getting KED for the 12 bit systems. As noted above, this is a major disadvantage for me.
2. TECO runs on any terminal from a teletype to a VT-11 interactive graphics display system. KED only works on a VT-100 (it might get VT-52 support but don't bet too much on it). We have a teletype, DECwriters, a VT-52 and a VT-11 that need to be used with the editor we pick. Exactly because, as Tom points out, KED is a DEC supported product, it is not readily available to the users to modify or to add support for other devices. The decisions on which machines, operating systems and terminals will be funded for support, and when, is driven by DEC marketing considerations rather than existing users' needs.
3. KED would be, most certainly, more user oriented, more "learnable", safer and more satisfactory for the vast majority of users than TECO is. The key to this quandary for us is the TECO video terminal support that is now available on all the systems we are interested in. With it, a version of the VTEDIT macro can be used on all systems with a VT-52, VT-100 or VT-11 giving a screen oriented editor that can be learned by naive users in minutes. The new user will be able to move across all our systems with almost no further training.

When KED is available on at least all the -11 systems we use, it is low cost, and works with the terminals we have, I will give it very serious consideration as an alternative to TECO. When KED is supported in this way on our 12 bit systems (i.e., pre Omnibus included) I will almost certainly change over to it. Until then, TECO continues as the only useful, portable editor for me and my user community.

Very truly yours,

Robert H. Hunt

## MORE ERRORS IN TECO REFERENCE CARD

Since the last issue of the Moby Munger, we have noticed a few additional errors in the TECO Pocket Guide (AD-D53ØA-TK). These are described below:

5. Page 3, the explanation of the EC command, "moves the remainder of the input file to the output file ..." should read "moves the text buffer and the remainder of the input file to the output file ...".
6. Page 4, the list of legal switches under TOPS-1Ø is missing some switches. In addition to /GENLSN and /SUPLSN, TECO also accepts the switches /SCAN, /NOCREATE, /NOWRITE, /SYS, and /NEW.
7. Page 11, in the explanation of the ES command, "an effective mV command is executed when a search is completed" should read "an effective mV command is executed when a successful search at top level is completed".
8. Page 12, under CASE CONTROL, we are missing the Ø↑V and Ø↑W commands. These should be printed in red since they are implemented in DEC's TOPS-1Ø TECO only. These commands are identical and take TECO out of case conversion mode.
9. Page 13, the search string build commands CTRL/V and CTRL/W should both have a note that reads "use of these constructs in a search string causes the search to be executed in exact case mode under TOPS-1Ø".
- 1Ø. Page 14, the explanation of the ?MEU error message should read "Macro ending with ↑", not "Macro ending with U".
11. Page 16, the graphic for the ASCII character whose octal ASCII code is 14Ø should be ` , not ´. (The accent is slanted the wrong way.)

---

## ERRORS IN MOBY MUNGER

The following errors in past issues of the Moby Munger have been noticed:

- o In the NOTES FROM RT-11 on page 6 of issue #1, we said that under TECO-11 V28, the n↑W command (with n≤-2) performs a screen update. This is wrong. It merely tells TECO that |n|-1 lines have changed on the screen. The window is not actually updated until the next -1↑W command (not when TECO goes back to prompt level). Note that in versions of TECO-11 after v28, the window command is W rather than ↑W.
- o DECUS has informed us that the program, DECUS-L-125 described on page 53 of issue #2 is no longer available from the DECUS program library.
- o Issue #2, page 7, "TECO.TEC" should read "WILD.TEC".

## WPI TECO

## VERSION 1(165)

WPI TECO is a modified version of DEC's TOPS-10 TECO v23. It was modified by Al Johannesen at WACCC who is also the current maintainer. Should anyone desire a copy, Al will send a copy to anyone who supplies the media. His address is

Al Johannesen  
 Worcester Area College Computation Center  
 Worcester Polytechnic Institute  
 Worcester, MA 01609

We described many of the features of WPI TECO in the last issue of the Moby Mungler. The following new features brings us up-to-date. This information describes WPI TECO v1(165) and is current as of June 1979.

13. 1EP sets the page number display flag.
14. 64EP allows an EB command to be prematurely terminated (current output file remains as a .TMP file).
15. 128EP re-enables normal use of the Y command.
16. EY is the same as Y but must be used instead of Y at command level (unless 128EP is set).
17. n↑P pages out to page n. Truth-in-paging mode (4EP) must be set. Ø↑P pages to the last existing page in the file. Backward paging is not permitted.
18. := and == were implemented as in OS/8 TECO and TECO-11. (They suppress the crlf). :Xq was implemented as in TECO-11. (Appends to Q-register.) :T and :EQ are like T and EQ but control characters (including carriage return, line feed, and tab) will be typed out in up-arrow mode regardless of the prevailing typeout mode.
19. <CTRL/G><FORM-FEED> is an immediate mode command that re-types the entire command string. (<CTRL/G>\* on some other TECOs.)
20. nV is identical to -nT(n+1)T . V is the same as ØTT. *(I prefer V to be the same as 1V and to define nV as (1-n)TnT.)*
21. nFDtext\$ (Find and Delete) is equivalent to nFStext\$\$ . m,nFDtext\$ is equivalent to m,nFStext\$\$ . *(Andy Nourse's TECO-10 has implemented this version of the nFD command.)*
22. To save space, ↑F, ↑H, and EM were removed.
23. When using : or @ modifiers, any numeric arguments must go after the : or @. *(This seems like an unnecessary restriction.)*
24. An additional Q-register, #, has been added. This register normally contains the filename last referenced by an ER or EB command. *(This is a true Q-register, unlike the pseudo-Q-register, \*, used by TECO-11 for a similar purpose. I think that real Q-registers are better than pseudo-Q-registers. \* is a pseudo-Q-register under TECO-11 because it can only be used in limited contexts.)*
25. The source is now written in FAIL. Also many internal speed-ups have been made. (Some speed-ups, like use of AC loops are of use only on KA10s.)

## XTEC

XTEC is a TECO compiler for the DECSYSTEM-10. It runs under TOPS-10 or ITS (under the compatibility package). This program is available from the DECUS program library. Its order number is DECUS-10-264. It was written by Jack W. Krupansky and Mark R. Crispin at Stevens Institute of Technology. The version in the DECUS library is 10(427), submitted in February of 1976. As far as I know, this is the latest version.

XTEC is patterned after Stevens TECO (version 123). It is a compiler rather than an interpreter (as most other TECOs are). When you start execution of a command string, or when you execute a Q-Register, XTEC compiles the TECO code into PDP-10 machine language. Re-execution of the same macro does not cause recompilation unless the contents of the Q-Register has changed. XTEC execution is over an order of magnitude faster than Stevens TECO execution of the same TECO program.

Stevens TECO (V123) was described on pages 33 and 34 of the last issue of the Moby Munger. The differences between XTEC and Stevens TECO are described below:

1. A value cannot be returned from a macro. If you colon the macro, both the first instruction and the entire macro will be "coloned", returning a value of -1 if it succeeds, 0 if it fails.
2. Values passed to a macro must be consistent in number in all usages of that macro.
3. XTEC supports Q-Register names of up to six characters. If more than one character is used, then the name must be enclosed by parentheses. For example, the command M(PRINT) executes the macro whose name is "PRINT". Allowing symbolic names for TECO "variables" makes for much more readable code.
4. The EI command uses a temporary Q-Register and does not smash \*.
5. If a compile-time error is detected, the command is aborted without doing anything. Therefore the command string HK=\$\$ will generate an error message and will not destroy the text buffer.
6. XTEC has different filespec blocks for each I/O command (used for filespec defaulting).
7. The W command of Stevens TECO is not in XTEC.
8. The EO value of XTEC is 0 (functionally equivalent to EO level 2 of Stevens TECO).
9. The Superseding Existing File message has been changed to use the error segment. To allow usage of "/" and "?", XTEC will type a fake prompt and wait for a continuation character to continue execution of the XTEC command. This character is ignored. *(This is very confusing to the user.)*
10. XTEC supports the following switches in filespecs: /SIXBIT, /ASCII, /OCTAL, and /PROTECTION:nnn. /SIXBIT reads the file in pure SIXBIT.

## HARVARD UNIX TECO (HRSTS TECO)

v6

Harvard UNIX TECO runs under the UNIX operating system on a PDP-11. It was written by Bob Case, Peter Langston, Bruce Borden, Brent Byer, and Tucker Taft using the Harvard-Radcliffe Student Timesharing System (HRSTS). A manual, written by Daniel Waugh and Ken Strong, is available from the HRSTS System Manager. For more information, or to obtain a copy of this TECO, write to

Stephen Dyer  
 HRSTS System Manager  
 Harvard Science Center 110a  
 1 Oxford St.  
 Cambridge, MA 02138

Educational institutions can obtain this TECO for a nominal fee. Harvard UNIX TECO is based on a 1973 version of MIT TECO and is written in MACRO-11. There is a possibility that this TECO will be expanded and recoded into C in the near future.

The following list summarizes the advanced features of Harvard UNIX TECO by specifying those features that are not in other traditional TECOs such as DEC's TOPS-10 TECO.

1. LF, if typed as the first keystroke after TECO's prompt, performs an effective LT command.
2. \$ or ↑, if typed as the first keystroke after TECO's prompt, performs an effective -LT command.
3. CTRL/A immediate mode command is equivalent to RUBOUT.
4. The immediate mode command CTRL/B aborts the command string.
5. Space eats numeric arguments.
6. EDfile\$ deletes the specified file.
7. EY replaces the traditional Y (Yank) command.
8. ET (Top) repositions the input file to the beginning of the file.
9. EQ (Quit) and ↑D↑D cause an immediate exit from TECO.
10. nV is equivalent to -nT(n+1)T . V is the same as ØV .
11. :T and :V are like T and V except that the type-out shows non-printing characters. For example, space prints as <SP> and the character whose ASCII code is 1 (CTRL/A) prints as <SOH> .
12. The K command not only kills text but in addition performs an effective :X@ command. This puts the killed text into pseudo-Q-Register @. Thus, you can get the deleted text back with a G@ command.
13. E+text\$ replaces the traditional +text\$ destructive global search command.
14. nFDtext\$ finds and deletes the n<sup>th</sup> copy of the specified text.

## HARVARD UNIX TECO (CONT.)

15. Match control constructs have been shuffled around a bit. CTRL/Q matches any character, CTRL/P matches any separator, and CTRL/E is the quote character.
16. Upper and lower case Q-Register names are distinct.  
(This has proven to be a lose in other TECOs.)
17. :Xq is like Xq but the extracted text is deleted from the text buffer.
18. The Edit Macro command, EMfile\$q\$, reads the specified file into the specified Q-register.
19. The Exchange Environment command, EEq, 'moves' you into the specified Q-Register. That is, the text in that Q-Register appears to be the normal editing text buffer and the original text buffer has been saved away in a pseudo-Q-Register. While in this mode, TECO's prompt changes to q\* where 'q' is the name of the Q-Register you are editing in. The EE\$ command restores the context to the main text buffer. (A very useful command.)
20. ← returns the negative of the number of characters in the last search string.
21. Q← returns the position of the beginning of the last match.
22. Q] returns the position of the pointer before the last search.
23. Q[ returns the position of the pointer before the last insert.
24. ↑E flag is a 1 if end-of-file has been reached.
25. ↑Wtext↑W types out the specified text (traditional ↑Atext↑A).
26. ↑L clears the screen on a video terminal.
27. E!\$ (E-bang!) spawns a sub-shell. When the user logs out of the sub-shell, control returns to TECO.
28. E!cmd\$ spawns a shell to execute the single command specified and then control returns to TECO.
29. All flags are handled in a consistent manner by the commands ESf, ECf, and EVf where f is the flag name. (Each flag name consists of a single character.) ESf sets the specified flag, ECf clears the flag, and EVf returns the value of the flag. (∅ represents off and -1 represents on.)  
A description of a few of the flags follows:

Flag	Description
B	controls creation of .BAK files
C	controls whether or not typing CR on terminal enters CR/LF or LF only
E	form feed flag
L	long error message flag
T	trace mode (causes user commands to be logged in a file called TECO.TRACE)



## TV

TV is a TECO variant that runs under the TOPS-20 operating system on a DECSYSTEM-20. It has many features in common with the other TECOs available at DEC, but it differs from 'traditional' TECOs more than the other TECOs that have been described in recent issues of the Moby Munger. The main problem is that the common commands R, ;, E, and F behave radically different under TV than they do under traditional TECO.

The information I have on TV is current as of July 1978. TV contains the following features that are not in other DEC TECOs:

1. ;Y means yank the entire input file into memory.
2. @J moves the text buffer pointer to the center of the screen. (TV sort of maintains a window into the text buffer on the screen of a video terminal.)
3. ;U means finish edit but stay in TECO (equivalent to the EC command of TECO-8 and TECO-11).
4. Negative arguments to a search cause the search to proceed backward. The pointer is left positioned at the beginning of the string found.
5. ;G gets the last command string that was longer than 15 characters in length and inserts it into the text buffer.
6. ;Efile\$ causes subsequent input to TECO to come from the specified file.
7. nRa\$b\$ causes the next n occurrences of string "a" to be replaced by the string "b".
8. ↑G means exit from TECO (as in DEC's TOPS-10 TECO, EO level 1).
9. CTRL/G run-time command aborts execution of the currently executing TECO command. (*Why isn't this CTRL/C?*)
10. Backspace, if typed as the first keystroke after TV's prompt, performs an effective -LT command.
11. Linefeed, if typed as the first keystroke after TV's prompt, performs an effective LT command.  
(*The above two immediate mode commands are now in TECO-11 V33 and TECO-8 V7.*)
12. ↑L clears the screen.
13. Carriage return eats up any pending numeric value.
14. CTRL/Q is an immediate mode quote character.
15. CTRL/U if typed as the first character in a line of a command string, deletes the previous line of that command string.
16. \$ (dollar sign) clears the current numeric value.
17. ;D puts a heading at the beginning of the buffer and then issues the ;U command.
18. ;H exits from TECO (*Why isn't this ↑C?*).

## TV (CONT.)

19. ;Mqfile\$ loads the specified file into Q-Register q.
20. ;P combines the ØA command followed by a C (returning the value picked up).
21. ;S saves entire buffer then closes file. m,n;S saves the portion of the text buffer between pointer positions m and n and then closes the output file.
22. Qq;T types the string contents of Q-Register q.
23. ;Tmessage\$ types the specified message.
24. nE is the same as nL2R, that is, it is like L but is used to position yourself at the end of a line (before the CR/LF) rather than at the beginning of a line.
25. n:K is the same as nK but doesn't kill the final CR/LF.
26. :L moves the text buffer pointer to the end of the current line (before the terminating CR/LF).
27. Failing searches don't move the text buffer pointer.
28. V, nV, and m,nV are the same as the corresponding T commands, except that the specified range of text is displayed on the screen.
29. The nXq and m,nXq commands delete the extracted text from the text buffer automatically.
30. Searches in iterations are not automatically colon modified.
31. A period immediately following a digit string acts as an 'octal point', that is, it causes the number to be interpreted in base 8.
32. There is a large number of W commands, of the form Wword\$ where 'word' is mnemonic for some TV feature. For example, WVTØ5\$ tells TV that you are editing on a VTØ5 terminal. WFLAGLOWER\$ institutes lower case flagging. (ØEU on other TECOS.)

The following TV commands are exactly equivalent to (or very similar to) the corresponding TECO commands:

<u>TV command</u>	<u>TECO command</u>	<u>TV command</u>	<u>TECO command</u>
;X	EX	;U	EC
Ra\$b\$	FSa\$b\$	↑Atext\$	↑Atext (↑A)
;C	EF	;Ftext\$	+text\$
;N	\	n;N	n\
;Rfile\$	ERfile\$	;Ttext\$	↑Atext (↑A)
;Wfile\$	EWfile\$	; <space>	;
Ftext\$	Ntext\$	CTRL/R	CTRL/G<space>

One interesting difference between the ;X command and the EX command is that ;X will prompt you for an output file name if an output file is not open.

## MIT TECO (ITS TECO)

This version of TECO is incredible! It runs on a DEC-SYSTEM-10 under the ITS operating system at MIT. MIT is where TECO began; they have kept way ahead of all other TECOs.

For most other TECOs described in the Moby Munger, I complain that the author's have made incompatible changes with DEC TECOs. In this case it is just the opposite - MIT TECO existed long before DEC's versions and DEC made many incompatible changes. Our only excuse is that we did not know what MIT was doing.

At MIT, TECO has grown very powerful; it is an order of magnitude more complicated than DEC's TECOs and has an equal abundance of functionality. It is even more cryptic and packs more power per character. My only comments are: fantastic! Stupendous! and My God!

The only documentation I have on MIT TECO is an 80-page list of commands. If there is a printed manual, I would appreciate receiving a copy. The information I have is accurate as of 2-Nov-1978 and describes TECO version 709.

There is too much functionality in this version of TECO for me to possibly describe it all in this column. Instead, I will only describe a few of the features (in little detail). Future issues of the Moby Munger will describe additional commands. Inclusion of a description of a command in this column in no way is an endorsement of the merits of that command.

1. n↑@ is the same as ., .+n if n is positive.
2. ↑A is an inclusive-or arithmetic operator.
3. ↑L clears screen when executed.
4. ↑Ofilename\$ bigprints the specified filename on the device open for output.
5. ↑O in a search string is a special character signifying "OR", that is, it divides the search string into two strings either of which will satisfy the search.
6. ↑P is a sort command. The description of the arguments and how this command works fills up an entire page and is too lengthy to reproduce here. Perhaps someone would like to contribute an article entitled "SORTING WITH MIT TECO".
7. n↑Z inserts n random letters before the buffer pointer.
8. Any conditional can be reversed by preceding the " by a colon modifier. Thus 0:"E fails.
9. # is an exclusive or arithmetic operator.
- ∅. ( turns off the colon and @-sign flags. The matching ) turns them on if and only if they were on before the (, but will never turn them off.
11. : and @ are both general command modifiers. Preceding a command by either or both of these modifiers changes the command into a different (but usually similar) command.

## MIT TECO (cont.)

12. `n=` outputs the value of `n` in the current output radix. The output radix is initially decimal and is kept in a special Q-register whose name is `..E` (MIT TECO has lots of special Q-registers of this form).
13. `@>` is the same as `>` except that the iteration count is not decremented. `!<!@>` is a good way to jump back to the start of the iteration without decrementing the count. The reason for the angle bracket in the comment is so that if the iteration should end earlier (say by a `Ø;` command), then TECO won't branch to this `>`, since it will see the `<` (MIT TECO doesn't ignore angle brackets found in strings or tags).
14. `n:A` appends `n` lines of the file (but won't append beyond a page boundary).
15. `:C` is like `C`, but returns `-1` ordinarily. It returns `Ø` if `C` without the colon would cause an error. `:J` and `:R` act in a similar manner. (`:D` doesn't seem to be implemented though.)
16. `EDfile$` deletes the specified file.
17. `E[` pushes the input channel if any. Saves the current input file and position in it, or saves the fact that no file is open.
18. `E]` pops the input channel. If any input file was open, the rest of it is flushed.
19. `E\` pushes the output channel.
20. `E↑` pops the output channel. If an output file is open, it is closed without being renamed.
21. `m,nF↑@` returns two values, which are `m` and `n` in numerical order.
22. `F↑Y` returns a value saying how many args were given it (in the low order 2 bits). The 4's bit is turned on if a `:` preceded the `F↑Y` and the 8's bit is turned on if an `@` preceded the command.
23. Within a macro, `F↑X` returns as its value, the arguments that were given to the macro. This command is frequently followed by a `F↑Y` to determine the number and nature of those arguments.
24. `FQq` returns the number of characters in Q-register `q`. `-1` is returned if the Q-register contains a number.
25. `m,nGq` inserts only a part of the text in the specified Q-register. Specifically, the range from `m` to `n-1` inclusive is inserted.

In this article, we have only been able to touch upon a very brief sampling of the commands in MIT TECO. We will continue this article in the next issue of the Moby Munger.

## MULTICS TECO

MULTICS TECO is a version of TECO that runs under the MULTICS operating system on a Honeywell computer. It is of interest to us because it is based on an early version of MIT TECO and consequently, its advanced features highlight those features of MIT TECO that were deemed most useful. Multics TECO was first written late in 1970 by Richard H. Gumpertz, then working for MIT. Later, Peter Bishop, also working for MIT became the maintainer. In 1972, David Levin at Honeywell became the principal maintainer and augments until 1977 (or 1978). In 1976, Richard G. Bratt added a slew of features derived from ITS TECO. In addition, in the mid 1970's, several other people at the Cambridge Information Systems Lab (the Multics System Software arm of Honeywell), such as Mike Grady and Larry Johnson, added certain additional features.

TECO's status in Multics is that of a 'tool'; it is not part of the standard service system and is used primarily by sophisticated systems programmers.

My thanks go to Bernie Greenberg who provided me with the information for this article which comes from Publication AZ03, "Multics System Programming Tools". (Incidentally, Bernie is the main implementor for a Multics version of EMACS written in LISP.)

A significant difference between Multics TECO and the TECOs that you may be familiar with is that Multics TECO does not use any special characters like ESCAPE or control characters. This is probably because early Multics users ran Multics on terminals that could not transmit these characters. Consequently, all strings are essentially @-sign modified, although no @-sign is required. Thus, to insert some text, you type I/text/ and to search for a string, one would type S/string/ where "/" denotes any non-alphanumeric character. In addition to the /text/ form of a quoted string, the construct Qq may also be used, where q represents the name of a Q-register. Thus SQA searches for the string that is stored in Q-register A. Command string execution is begun by ending a line with a dollar sign (\$) (followed by a newline). Other than these differences, Multics TECO looks very much like traditional TECO.

The following features are extensions to 'traditional' TECO:

1. There are 95 Q-registers, one for each printing ASCII character.
2. | is used as the boolean 'or' operator.
3. Input and output is a bit different because complete files are read into the text buffer so there is no need for paging; consequently no need for the A, P, Y, or PW commands. A file is read in with the EI/file/ command and written out with the EO/file/ command. EO can also take a numeric argument or argument pair representing a range of characters to output (as in the T command).
4. m,nS/string/ is like nS/string/ however only m lines are searched. (This is a bounded search command). If m is negative, previous lines are searched. If n is negative, the search proceeds backwards and a successful search leaves the pointer at the beginning of the string found.

5. N/string/ searches for a regular expression, string.
6. m,n= types out the two numeric values separated by a space.
7. n:= types out the value of n in octal.
8. EQ (External Quit) exits TECO.
9. :Iq/string/ or n:Iq inserts the specified text into Q-register q.
10. n:L is equivalent to nLR (i.e. it moves to the end of the line rather than the beginning). Note that in Multics, lines are ended by a single newline character.
11. n:J, n:C, and n:R are like nJ, nC, and nR except that if the text buffer pointer would be moved out of bounds, no error is generated and the text buffer point is set to B or Z as appropriate.
12. m,nUq is equivalent to nUqm .
13. m,nPq is like m,nXq except that the text is appended to the specified Q-register. *(The letter P was free in this version of TECO, however, for TECOs that do paging, I recommend the :X command to be the append to Q-register command.)*
14. :< and n:< are like < and n< except that errors that occur within the iteration group just terminate the group and the > returns a value. -1 is returned if no errors occurred, 0 is returned if the iteration terminated due to an error.
15. ;; is like ; except that the sense of the test is reversed. n; exits the current iteration if n<0.
16. If the target of an O command is not found in the current macro level, the previous macro level is searched. This continues back to command level. Although a GOTO can exit a macro, it cannot be used to exit an iteration. *(A funny mixture of extension and restriction.)*
17. :Mq, if issued from within a macro, causes a return from that macro to occur when macro q is returned from.
18. EM/file/ executes the TECO commands stored in the specified file.
19. String arguments may be passed to a macro. The general call to a macro has the following form:

m,nMq/arg<sub>1</sub>//arg<sub>2</sub>//arg<sub>3</sub>/...

The command :Xq, if issued from a macro, puts the next argument into the specified Q-register. (The first :Xq gets the first argument, the next :Xq accesses the second argument, etc.) The macro and the caller must agree on the number of arguments, since when the macro is exited, control resumes with the first character after the last string argument that had been fetched.

20. `m,n"E..."` executes the commands if `m=n`.  
`m,n"N..."` executes the commands if `m≠n`.
21. `"M/string/..."` executes the commands if the specified string appears in the text buffer just to the right of dot.  
`:"M` is like `"M` except the sense of the test is reversed.
22. `:'` transfers to the next `'`. This is useful in creating if-then-else statements, e.g. `n"Xthen:'else'`.
23. `VW` reads a character from the terminal and returns its ASCII value. (*We are familiar with this as a CTRL/T command.*)
24. `:VWq` reads a line of text from the terminal and puts it into the specified Q-register.
25. `EC/string/` passes the string to the Multics request processor (command processor) for execution.
26. `EAq/string/` passes the specified string to the command processor's active function application entry. The result is returned in Q-register `q`.
27. `nA` returns the ASCII code for the  $(.+n)^{\text{th}}$  character in the buffer. `lA` gets the value of the current character (the one just to the right of the text buffer pointer).  
*(This is compatible with MIT TECO as opposed to TECO-11.)*
28. `:\` is like `\` except that radix 8 (octal) is used.
29. `m,n\ inserts the decimal interpretation of m into the text buffer (before dot). It is padded on the left to be at least n characters wide.`
30. `$` (embedded dollar sign) throws away any numeric arguments.
31. Special loop facilities are provided for throwing and catching values. `F<` and `>` define an iterative group like `<` and `>`. From the time that the `F<` iteration group is entered until the time it is exited, `F<` sets up a handler to "catch" values thrown by the `F;/label/` request. If no `F;/label/` request with matching string argument is executed before the `F<!label!` group is exited, the iteration group returns -1 as a value. If, however, an `F;/label/` request is executed (where the `/label/` matches one that immediately follows the `F<`, i.e. `F<!label!`), then the execution of all macros and iteration groups encountered since the `F<!label!` is abandoned, and the `F<` iteration group returns the numeric argument of the `F;` request as a value. `:F<!label!` is like `F<!label!` except that if an error is encountered during the execution of the `:F<` group, the latter returns `∅` as value. These requests provide a method of exiting several nested loops at once. Execution of a `F;` command terminates the `F<` loop as well as any contained loops.
32. The Q-register `"` always contains the last quoted string that has been referenced by TECO.
33. `:T/text/` types out the specified text on the terminal.
34. A period at the end of a digit string is an "octal point".

## TECO NOW RUNS ON YOUR HOME COMPUTER!

## TEC65

## TECO ON 6502 MICROCOMPUTER SYSTEMS

A subset of TECO developed to run on microcomputer systems using the 6502 microprocessor chip is available for systems using TIM, SYM, and KIM monitors. It is also soon to be available for APPLE II systems.

TEC65 (as it is named) was written by Larry Fish of Denver, Colorado who became familiar with TECO implemented on a PDP-10. TEC65 requires just under 4K bytes of RAM. Total RAM required depends on the amount of text you wish to handle at one time. SYM and KIM versions allow saving and loading of text via the cassette ports provided. The new APPLE II version will make use of APPLE II Disk hardware.

A catalog describing this and other 6502 software is available for \$1.00 from the 6502 PROGRAM EXCHANGE, 2920 W MOANA, RENO, NV 89509.

David P. Marsh  
Vice President  
The 6502 Program Exchange  
2920 Moana  
Reno, NV 89509

Remarks from the editor:

Yes, it's really true! To quote from the catalog, "There is no reason why you shouldn't have one of the finest text-editors on the market today".

I've got a copy of the TEC65 manual, and indeed, they have a real TECO, although it's a subset of the traditional TECO for the -10.

The following commands are implemented:

C, D, FSa\$b\$, Gq, Itext\$, J, K, L, Mq, T, Xq, =, <...>, Stext\$,  
;, Ntext\$, \*q, EX, ., H, B, Z, and TABtext\$.

There are also a few other I/O commands that are not compatible with traditional TECO, so I won't go into them here.

Q-register names are limited to 0 through 9. There are also the usual immediate action commands such as rubout and CTRL/U, as well as LF which performs an effective LT if typed as the first character after TECO's prompt. Also CTRL/G which executes an effective 0LT and ESC which executes an effective -LT. The match control construct CTRL/X is also permitted in search strings.



## QUESTIONS AND ANSWERS

This column will attempt to answer questions submitted by readers on any TECO-related topics. Queries should be submitted to the editor, Stan Rabinowitz, 6 Country Club Lane, Merrimack, NH 03054.

Q4. by Ronald J. Price (U.S. Environmental Protection Agency, Houston, Texas)

I am presently using version 3 of DEC's OS/8 TECO, but I have version 4. How can I get information concerning the differences in the two versions?

A4. V4 came out with a new release of OS/8 and has no new features but contains bug fixes. You should be using V4 rather than V3 because it is solidier. In fact, you should get the latest version (V5) since it has considerably more functionality than V4.

Q5. by Ronald J. Price

I am using OS/8 TECO on a PDP-8/E with 32K memory. How can I increase the Q-register size from a maximum of 2000 characters to something larger?

A5. On 8K machines, you get 2000<sub>10</sub> characters in Q-register space. On machines with 12K or more, you get a maximum of 2944<sub>10</sub> characters in Q-register space (although no single Q-register may contain more than 2047 characters). I know of no way to increase this limit of 2944. Allowing Q-register space to cross field boundaries is a very tricky operation; although with 32K you have lots of room to write tricky code. If anyone can provide a patch to OS/8 TECO to allow expanded Q-register space, please write in.

Q6. by Dr. R. N. Caffin (Division of Textile Physics, C.S.I.R.O., Ryde, N.S.W.)

I have several questions on VT52 support under RT-11. The Backspace (BS) key is meant to move the cursor to the end of the current line. However, under certain conditions after editing within the line, it fails to do so: it ends up some characters short of the end. Does anyone have a cure for this?

A6. I have never encountered any problems with the BS function. Please check that you have an uncorrupted version of VT52 (better yet - switch to VTEDIT). If the problem recurs, please try to send circumstances which can be duplicated which cause the error to happen. Note that the implementation of this function is not too swift; it simply executes the TECO commands L2R. This means that if the line ended with something other than CARRIAGE-RETURN / LINE-FEED, typing the BS key may not do what you expect. This could happen if the line ended with FF or end-of-buffer.

Q7. by Dr. R. N. Caffin

Could we arrange to have future releases of VT52.TEC to omit the screenful of keypad layout, assume that all necessary files (except for EDIT.TEC and the text file) are on SY:, and to allow lower case handling automatically?

A7. The source of VT52.TEC is distributed in all releases. This allows users to make their own custom modifications. We can't please everyone, so we release it with the options that we feel are most useful. You should feel free to make your own modifications (but be sure to document them; and don't send in any bug reports unless the bugs occur in the macro as distributed by us).

Q8. You mentioned in MOBY MUNGER #2 the existence of VTEDIT. An interesting extension of VT52, with its word orientation, but how does one get a copy?

A8. By the time you read this, VTEDIT should have been submitted to the DECUS library, or will be shortly. Wait for announcement of availability from DECUS, then order it from DECUS. Copies were also given out at the last DECUS symposium.

Q9. What is the chance of including officially, in VT52, the command CTRL/P for "scroll one page-full forward" and (say) CTRL/Q for "scroll one page-full backwards"? It's a very useful facility for browsing through a long listing or a piece of text.

A9. The latest version of VTEDIT (V1.00) already has this feature (but with a different syntax). See elsewhere in this issue for the functionality of the latest version of VTEDIT.

Q10. Is anyone working on an extension to VTEDIT which will allow direct preparation of text, bypassing the use of RUNOFF? The main features required are the automatic adjustment of words in a paragraph following an insertion or deletion, right margin justification and listing with page numbers.

A10. I know of at least two people at DEC who have macros that do this kind of thing. I can only hope that they will submit them to the DECUS program library.

Q11. Is it possible for TECO to perform direct output to another terminal or line printer?

A11. Yes. Unless the operating systems denies you access (say if the line printer is assigned to another job) there is no reason why you can't perform an EW command to such a device. If you are using TECO-11 and you are in the middle of an EB, you can open the line printer (or terminal) for output on the secondary output channel by issuing a command such as EA EWLP:\$ and then issuing P commands. The EW\$ command then switches you back to the primary output channel.

Q12. by Dr. R. N. Caffin

In humble ignorance, what is the meaning and origin of MUNG?

A12. MUNG is a recursive acronym for Mung Until No Good. This is what you typically do to files with TECO. I believe the word originated at MIT but am not sure. Better information concerning the origin of this word would be appreciated.

Q13. by Chester Wilson (Charleville Australia)

I have the DECUS TECO release for RT-11 and am having a difficult time working out such things as changing the default ET and ED values. Is there a simple way to do this?

A13. RT-11 TECO V34 (soon to be available from DECUS) supports user initialization via a file called TECO.INI. The TECO commands in this file will be executed before TECO starts up.

Q14. by J. M. "Pete" Asensio

As newcomers to TECO, we find it somewhat difficult to decipher involved programs, such as the scope macros, when no comments are present in the distributed version. Is it possible to obtain a commented listing of VT52.TEC or VTEDIT.TEC? We wish to patch these macros for use with non-DEC terminals, specifically the Infoton I-100.

A14. no answer supplied. The authors were "too engrossed" in getting this marvel to work that they didn't bother to include any comments. Anyone who has commented it should send us a copy. We hope that other readers who send us useful TECO programs will properly comment them.

Q15. by Nick Carr (DEC Europe)

My main use of TECO is with RSTS/E. Unfortunately, these scope macros run rather slowly with RSTS/E. It is possible to type ahead several characters (apparently between run bursts) and then TECO will suddenly catch up. Would it be feasible to increase the speed of these macros, possibly at the expense of some functionality?

A15. We are always trying to increase the speed of VTEDIT. Suggestions for speed improvement are always welcome. If the latest version of VTEDIT seems too slow for you, you might try switching back to VT52.TEC - it should run faster but has considerably less functionality. It is always possible for you to go through VTEDIT removing features that you don't want. You can also shuffle the code around. Putting those features that you use most frequently at the beginning would speed these features up.

Q16. by H. S. Hopkins Jr (General Latex and Chemical Corp. of Ohio)

Let's say I'm in the middle of the third page of a 5 page TECO source, and want to bring in (insert) several lines or all of another source file. I keep losing 1) the rest of the 5 pages, 2) my place in the source, or 3) everything except the backup file. Is there a convenient way to make this type of insertion?

A16. If you know in advance that you want to make these insertions, you should open the file containing the material to be inserted first. Read the file in, store the material to be later inserted into a Q-register, and then begin your edit. If you are using TECO-11, another solution is possible (this method is not available with OS/8 TECO or TOPS-10 TECO): Store away the current text buffer (HXAHK), open up the other file using a secondary input stream (EPERfile\$Y), locate the material to be inserted and store it away in some Q-register, delete the unwanted text and switch back to the primary input stream (HKER\$GA) and restore the text buffer. Then resume your editing.

Q17. Conversely, I would like to take 15 or 20 lines from page 4 and put them somewhere on page 2. How can this be done?

A17. Some TECO variants allow reading the entire file into memory to let you do editing like this. If you don't have such an editor, there are several ways you can proceed. The simplest is to move the text into a Q-register, finish your edit without leaving TECO, and then do another EB and insert the text where desired. Another solution (which only works if you have enough memory and the two pages are near each other) is to read several pages into core at once; enough to include the two pages in question. When doing this, don't forget to re-insert the form feeds that the Append command removes when appending pages together. For example, if you were on page 2, instead of doing a P when you were done, you would issue a AZJ12I\$A command. Pages 2, 3, and 4 would now be in the text buffer and you would have no trouble moving text from page 4 to page 2. Another solution, for longer shuffles, as you say later in your letter, is to break the source up into several files, edit these as necessary, and then recombine the files in the desired order.

Q18. by Ray Newman (North Brisbane, Australia)

We have changed the effect of the CTRL/Z command (in the VT52 macro under RSTS/E) to do an EX. Unfortunately, this will, if and only if the private default run-time system is BASIC, leave the terminal with the echo turned off. How can we get around this problem?

A18. This problem has been solved for you in VTEDIT. With that macro, you use the ESC CTRL/Z to perform a clean exit. You probably could have solved the problem by explicitly turning echo back on via the command (-8-1)&ETETEX .

19 by Ian Calhaem (Board member, DECUS Australia)

I have made modifications to VT52.TEC (as supplied in RT-11 V3) to enable use with a VT100. How can I ensure that a RESET escape sequence is output after the command EX\$\$ ? How can I convert from 80 to 132 columns and reverse?

19. Get and use VTEDIT when it becomes available. This is an expanded version of VT52 which fully supports the VT100. This macro supports 132 column mode and other VT100 features (for example, it displays your 'cut buffer' in reverse video. To quit editing, instead of going back to TECO and issuing an EX command, you type the ESC CTRL/Z command to VTEDIT. VTEDIT then sends the proper escape sequences to your terminal (if it is a VT100) to put it back in a nice mode, and then exits TECO.

Q20. Can RT-11 TECO V28 be made to run as a foreground job? I would like to run in foreground and background at the same time.

A20. Yes. It is very easy to get TECO to run in the foreground under RT-11. All you have to do is link together the object modules (TECO.OBJ and TECOIO.OBJ) using the /FOREGROUND switch. (You should also specify a 100-word stack.) The result is TECO.REL which can be run in the foreground via the .FRUN TECO command. Mark Bramhall reports that there were some bugs in version 28 of TECO for RT-11 which might cause problems with running TECO in the foreground. All these bugs have been fixed in V34. In fact, running TECO in the foreground under V34 is even simpler, since no re-linking is necessary; a prelinked TECO.REL is included in the V34 TECO kit.

Q21 by Ray Newman (North Brisbane, Australia)

Under RSTS/E, two of the internal flags behave in different ways depending on whether the user is privileged or not. On scope terminals the ET flag and the EU flag are both 0 for non-privileged users. For privileged users these flags take their correct values. How do we get around this?

A21 You are using an old version of TECO. This problem has been fixed long ago. The TECO that was distributed with the 6C release of RSTS/E does not exhibit this failure. Use that version, or use V34 when it becomes available from the DECUS library.

Q22. by Stanley Rabinowitz, DEC Merrimack

What is the origin and derivation of the term "Q-register"?

A22. No answer supplied. Responses are solicited.

## THE SEWERS OF PIING POONG

Ned Freed has written in to tell us about the many TECO programs that he has. These were written for TEXAS TECO version 124(344). I hope that he will submit them to the DECUS program library. In the mean time, he will send them to anyone who sends him a magtape. The two he sent to me were too big to print. One, called TECINI, was 4 solid pages of TECO code and is a utility to allow quick construction of good TECO.INI files. It allows TECO novices to have very sophisticated macros at their disposal, and for experts to generate them more quickly. The other program is called SEWER. It was originally written by Darryl Morrell and modified by Charles Carvalho. I was quite impressed by this game. It is a mini-ADVENTURE and was almost 5 pages long, including such informative messages as YOU ARE SITTING IN YOUR LITTLE MUD HUT IN PIING POONG EATING YOUR BOWL OF RICE WHEN ALL OF A SUDDEN A TYPHOON BLOWS IN. THE RESULTING FLOOD WASHES YOU OUT OF YOUR HUT AND INTO ... THE SEWERS OF PIING POONG, and YOU ARE IN AN ANCIENT CATACOOMB WITH PILES OF BONES EVERYWHERE. A LARGE VAULTED PASSAGE WEST IS NOW BLOCKED BY A CAVE-IN. THERE IS A SMALL HOLE IN THE EAST WALL. THE SKELETONS ARE WEARING VARIOUS PIECES OF JEWELRY, ..., and YOU ARE ON THE SOUTH EDGE OF A STREAM OF SLUDGE .... The object is to get out of the sewers. Ned's address is:

Ned Freed  
Seaver Computer Center  
The Claremont Colleges  
Claremont, CA 91711

---

## HELP PROGRAM FOR RSTS/E

Ray Newman (of North Brisbane, Australia) sent along the program HELP.TEC (printed on the next page). This program responds to the RSTS/E CCL command HELP. If there is no following argument, it lists topics for which help is available. Arguments that follow the command are used to select information from one of two help files and display it on a VT52. It first searches the users PPN for the file HELP.HLP. If it fails to find it there, it searches [1,2]. The format of the file is: One topic per page and all topics and sub-topics enclosed in matched braces. Ray says he is indebted to RT11 for the idea, but not the code which is of the home-grown variety.

## HELP.TEC

=====

```

-----|
| Responder to the HELP CCL |
-----|
| Set up the various Q res |
@^US\UT ET#1ET 27 CT QTCT ET&(1^_)ET\
@^UC\^)\
@^UZ\ZJ ZU1 GA Q1J :@S/ /
  *T Q1,.-1XD .,ZXA Q1,ZK / , \
OUB OUB -1EU -1^X @^UE/
      / @EW// G*
| Close off command file |
| and check for correct CCL |
@EI// J :@S/HELP/*F @^A/?Illegal CCL command
/ ^C/
| Clean up and store command |
O..K ^D ZJ @I/ / J
!REM! :@S/^ES/ *T -D @O!REM!
< @S/ /;-DJ > HXA HXD HK
| Try for private help file |
| if that fails go straight |
| to the system file [1,2] |
:@ER/HELP.HLP/ *F
!SYSFIL! ID JA 1U9
:@ER/[1,2]HELP.HLP/ *F @^A/
?No help file available
/ ^C // Y
| Check for command if none |
| then just list topics |
QB *N @O!TC! / MZ *E @^A/
      Help is available for the following topics:

/
OU1 !TC! < J D @S/^S/ B,.-1T .-1^1-60 *G @^A/
/ OU1 / < -(Q1&15); @^A/ / 1^1 > ^N^L; Y >
Q9 *E 1U9 @O!SYSFIL! / @^A/
/Q9 ^C/
| The main loop starts here |
OU1 < OJ < !ARRGH! -1^X @S/^/;
.U2 1U4 < :@S/^EGC/ *F @^A/
?Corrupt HELP.HLP file
/ ^C / 1^4 0-1A-125*E 0-2^4 *E .U3 @O!CONT! / / >
!CONT! Q2J 0^X :@S/^EQB/ *T @S/^S/ Q2,.-1;XE :@^UE/ /
  MZ *N Q3-1,ZK B,Q2K J @O!ARRGH! /
  Q3-1,ZK B,Q2K
!TYPE IT ! J < @S/^EGC/;-D >!* ET&512 *!1*N 72MS 74MS 91MS /
!GE @^A/

/ ET&512 HT @^A/
/!*ET&512*!1 *N 92MS / ^C / Q2J Q1 *E Q3J / >
  ^N^L; Y > Q9 *E @O!SYSFIL! / @^A/
%No help for / :GD @^A/
/ ^C $$$

```

## SQU,TEC = the TECO macro squisher

Mark Bramhall

This macro will take a nice, readable TECO-11 macro and squish it so that it is as small and as fast as possible (and is completely unreadable). SQU is distributed with the TECO-11 kits and will run under any operating system that supports TECO-11.

You invoke SQU with the MUNG command. For example:

MUNG SQU

SQU then enters into a dialogue with you as described below. All questions contain the allowed legal responses enclosed in parentheses and the default you get if you just type RETURN in angle brackets. In the discussion below the term "lexical" refers to characters that TECO would have interpreted as TECO commands; all characters inside string arguments are not lexical characters. For example, the ABC of SABC\$ are not lexical characters while the S and \$ are. Similarly, the ABC and DEF of #FS/ABC/DEF/ are not lexical while the #, F, S, /'s, and \$ are.

Delete CR/LF (Y/N) &lt;N&gt;?

A Y response will delete all lexical RETURNS and LINE FEEDS. As this usually results in one very long "line", an answer of N is suggested (but, see the question below).

Set line lengths (Y for 70, N, or length) &lt;N&gt;?

This question is asked if you responded N to the previous question. If a positive response is given, SQU will initially delete all lexical CR/LFs, then, as a post-process, insert the CR/LF combination wherever it's needed so that no line is longer than the length specified. This neatly formats the squished TECO macro. The most commonly used length is 70 so a response of Y results in a length of 70.

Delete blank lines (Y/N) &lt;N&gt;?

This question is asked only if you responded N to both of the questions above. A Y answer results in all lexical blank lines being deleted.

Delete lexical TABs and FORM FEEDs (Y/N) &lt;N&gt;?

Even though lexical TABs and FORM FEEDs have meaning to TECO (TAB starts an insert, FORM FEED outputs a FORM FEED on the terminal), some TECO macro writers use them simply to format their macro code. Of course, their macros won't correctly execute without removing those lexical TABs and FFs. A Y response will delete all lexical TABs and FORM FEEDs. If your macro runs unsquished, you certainly want to answer N.

Delete comments (Y for SP/TAB, N, or set) &lt;N&gt;?



The trouble here is that both comments and labels (tags) use the same construct. To be able to differentiate between the two, some sort of coding convention must be used. SQU will check the character immediately following the "!" that starts the comment or label. If that character matches the one of the set of "comment characters" specified, SQU will delete the construct. Using a Space or TAB immediately after the leading exclamation point seems to be the most flexible for formatting comments so a Y response results in the "comment character" set being Space and TAB.

Watch progress (Y/N) <N>?

If your terminal is supported by the W (scope watch) commands, this question will be asked. At the cost of SQU running more slowly, you can watch its progress. Useful for debugging, not for production squishing...

Automatic mode (Y for %, N, or set) <N>?

This is the most confusing question. The problem here is that SQU normally won't touch characters inside string argument strings, but the %U command is frequently used to load TECO macro code into Q-registers and, of course, you wish that code to also be squished. On the other hand, all %U commands aren't necessarily loading TECO macro code. There are two forms of the %U command: %Uxxxxs and @%U%xxxx%. The rule SQU uses is that any %U command that is of the @%U%xxxx% form and whose delimiting character is a member of the "non-squishable %U command" character set shouldn't be sub-squished; every other %U command's string argument is assumed to be TECO macro code and is sub-squished. The convention we've adopted is that the quoting character "%" is to be used for textual (i.e., non-code) %U command arguments. Therefore, a Y response sets the "non-squishable %U command" character set to %. You may enter any set of character(s) you want. For example, a reply of %/\ means that any %U string delimited by a percent sign, slash, or backslash character should not be sub-squished. An N response places you in manual mode. Whenever SQU encounters a %U command, it will pause, ring the terminal's bell, and await a response. A Y says to go ahead and sub-squish the string argument; an N means to skip it. Obviously, you want to start coding your macros according to some convention so that you don't need manual mode.

Allow adjacent ESCapes (Y/N) <N>?

The process of squishing a macro can result in two adjacent ESCAPE characters where there wasn't before. For example, if the command @FS/ABC// was used, SQU will turn it into FSABCSS. If the macro will be read via an ER command, yanked into the text buffer, copied to a Q-register, and executed this poses no problems. On the other hand, if the macro is to be EI'd, the adjacent ESCAPes will trigger command execution prematurely. An N response will direct SQU to never generate adjacent ESCAPes

File <\*.TEC>?

SQU is now asking you for the file specification of your TECO macro and indicating that it will default the extension to .TEC. After your reply, SQU will take off and start squishing the macro. When it's done, it will simply return to TECO's \* prompt. You can now write out the squished macro however you want.

If you invoke SQU as

```
MUNG SQU,filnam
```

it will enter its dialogue for the squishing options, but will start squishing immediately after asking about adjacent ESCapes then return to the \* prompt.

You can invoke SQU specifying all of the squishing options as switches. The switches are:

```
/D      Delete CR/LF
/L      Set line lengths
/B      Delete blank lines
/T      Delete lexical TAB/FFs
/C      Delete comments
/W      Watch progress
/A      Automatic mode
/E      Allow adjacent ESCapes
```

Each switch should be expressed as /x:y where "x" is the switch and "y" is either Y or N. A non-standard line length can be expressed as /L:nn where "nn" is some non-zero digit string. Non-standard character sets can be specified for the /C and /A switches, but the character set is limited to a single character. For example, /C:\* sets the comment character set to "\*" and /A:~ sets the non-squishable "U" command character set to "~".

You can also supply SQU with both output and input file specifications. For example:

```
MUNG SQU,outfil=infil
```

SQU will enter its dialogue for the squishing options, then will immediately do the squishing, write out the result, and exit.

If you have specified everything SQU needs (an input file, an output file, and all of the options as switches), it will do the whole squishing operation and exit without further intervention. A convention used at DEC (and recommended by the TECO SIG) is that if a TECO macro is intended to be squished, the unsquished version be given a .TES extension (TECO Source) and the squished version be given a .TEC extension. A file with a .TEC extension must always be a runnable TECO macro (or program).

## TECO-RELATED DECUS SUBMISSIONS

The following items of interest to TECO users are available from the DECUS program library:

<u>Number</u>	<u>Description</u>
DECUS-8-873	RSTS Terminal Monitor on a PDP-8 Written by Andras Nagy. This package lets you connect together OS/8 and RSTS/E. Its goal is to synthesize the powers and versatilities of the RSTS and OS/8 operating systems. The source language consists of a mixture of assembly code and TECO macros.
DECUS-10-116	QED Written at the University of Utah. This is an implementation of the well-known line-oriented editor known as QED. It is written in TECO and runs under a slightly modified version of a 1970 copy of DEC's TOPS-10 TECO.
DECUS-10-148 } DECUS-11-53 }	PDP-11/10 Loader Written by H. L. Farnsworth and R. B. Fleisher. Can be ordered under either number. Consists of two TECO programs that allow full duplex conversation between TOPS-10 and a PDP-11 terminal. Also lets a PDP-11 program to be assembled on the -10 and loaded directly into PDP-11 memory.
DECUS-11-360	RSX TECO with Buffered Typeout and VT52 Window Support, plus TECO DOCTOR Game Written by G. Everhart. This consists of two parts. The first part is a video editor (for a VT52) written in TECO. It runs under a modified version of TECO-11 which is enclosed. That version of TECO is now obsolete since the latest TECO-11 for RSX supports VT52s properly. The second part of this submission consists of the well-known DOCTOR program (ELIZA) written in TECO. It is slightly less powerful than the original LISP program. It allows the computer to respond to your statements as would a psycho-analyst, and attempts to carry out a conversation with the user. Should run under most TECOs. What does that suggest to you?
DECUS-11-265	DOS-11 TECO Written by Glenn Everhart. It is a copy of version 15 of TECO-11 which has been interfaced to a new I/O module to allow it to run under the DOS-11 operating system.



DIGITAL EQUIPMENT COMPUTER USERS SOCIETY  
ONE IRON WAY, MR2-3/E55  
MARLBORO, MASSACHUSETTS 01752

**MOVING OR REPLACING A DELEGATE?**

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

- ( ) Change of Address
- ( ) Delegate Replacement

DECUS Membership No.: \_\_\_\_\_

Name: \_\_\_\_\_

Company: \_\_\_\_\_

Address: \_\_\_\_\_

\_\_\_\_\_

State/Country: \_\_\_\_\_

Zip/Postal Code: \_\_\_\_\_

Mail to: DECUS - ATT: Membership  
One Iron Way, MR2-3  
Marlboro, Massachusetts 01752 USA

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.