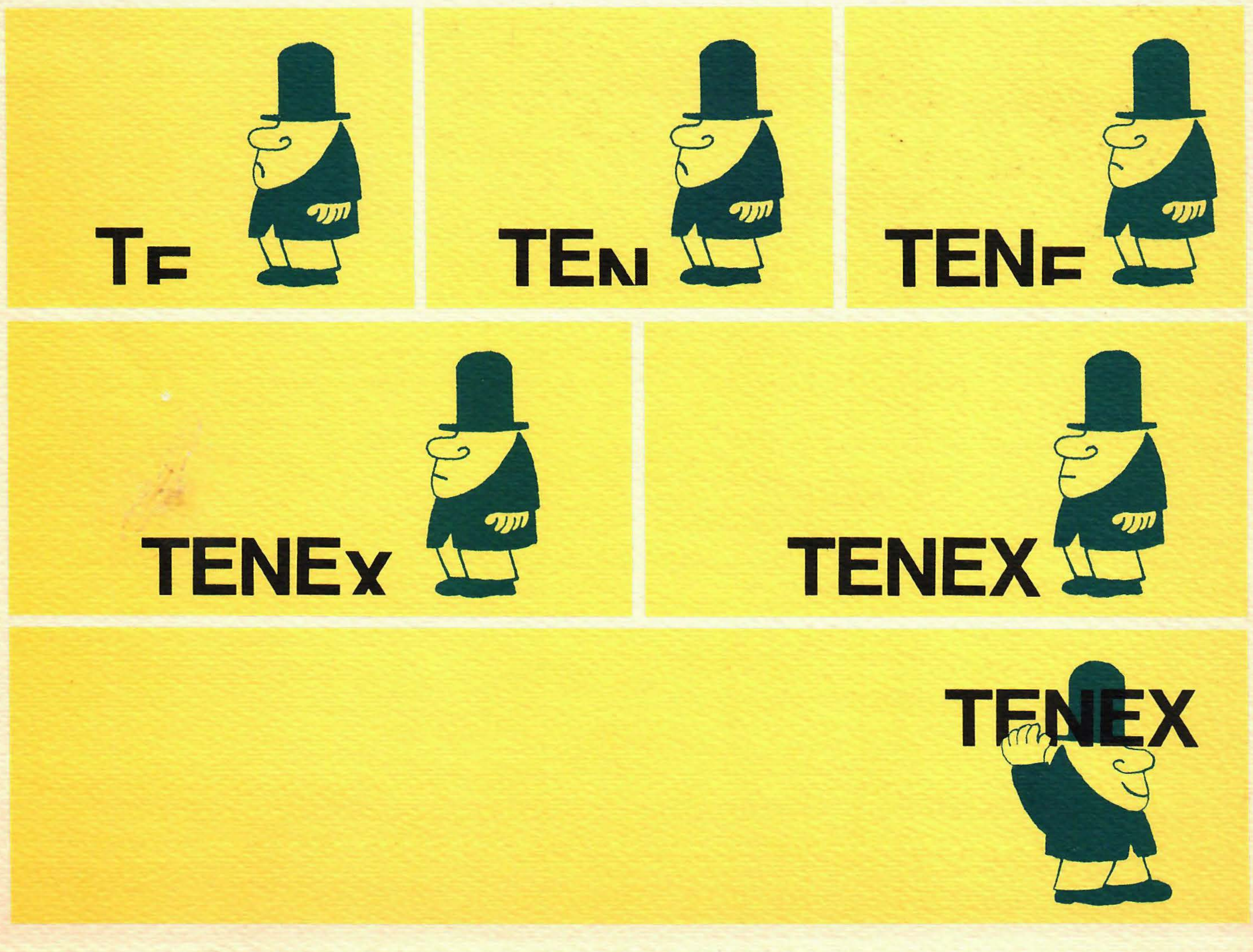
Bolt Beranek and Newman Inc.

users' guide



TENEX USERS' GUIDE

Bolt Beranek and Newman Inc.

5Ø Moulton Street

Cambridge, Massachusetts Ø2138

ADDENDUM - October 1977
2nd Revision - January 1975
1st Revision - January 1973
Copyright - May 1971

INTRODUCTION

New Additions or Changes to TENEX Users Guide

For users of this guide the most recent additions or changes since 1975 will be flagged on the replacement page on the right side with the character "|". Date of replacement is shown on page for your convenience in updating the manual.

A user will find NETWORK related subsystems in the concluding section of the TENEX USERS' GUIDE.

If there are any questions regarding any of the subsystems mentioned in this manual, please contact the Operations Manager at Bolt Beranek and Newman, (617) 491-1850 Ext. 484, or 617-491-6169 for a direct line to the Research Computer Center.

TENEX SUBSYSTEMS

Name	Page #	Description
ALGOL	1	An implementation of the ALGOrithmic Language ALGOL-6Ø. See DECsystem1Ø ALGOL Manual.
BASIC	3	A conversational problem-solving language. See DECsystem1Ø BASIC Manual.
BCDTAP	6	BCDTAP reads or writes a BCD magnetic tape, file by file.
BCPL	--	BCPL is a simple recursive programming language designed for compiler writing and system programming. For more information see the TENEX BCPL Manual.
BDDT	--	A debugger for BCPL, see the TENEX BCPL Manual.
BEDIT	1-1	A Binary File Editor
BINCOM	--	DEC version of Binary Compare.
BLISS1Ø	11	Compiler for system implementation.
CACCT	--	CACCT is a program for systematically changing account numbers for files.
CALENDAR	2-1	A subsystem to help people keep track of schedules and daily tasks. CALENDAR can also serve as a reminder for birthdays, appointments, anniversaries, etc.
CCL	--	The Command Control Language is intended to speed program development by simplifying communication between a user and the system programs. See NCCL

COBOL	2Ø	(COmmon Business Oriented Language) is a large DEC-supplied compiler and operating system.
COPYM	29	Program designed to facilitate copying groups or lists of files.
CREF	31	CREF is a program to make cross reference listings.
DDT	37	An interactive debugger, also see SDDT, IDDT, and UDDT. For additional information on DDT, please refer to DECsystem 2Ø User's Guide.
DELVER	41	A program to assist in the management of file versions.
DFTP	3-1	A user invoked program which stores and retrieves local files on the Datacomputer. See Network Section.
DO	42	A subsystem for passing a parameterized text string from a specified file to the EXEC for execution.
DTACPY	44	Copies full dectapes to full dectapes
DUMPER	--	A program for dumping and restoring files on magtape.
ECAP	45	An Electronic Circuit Analysis program. See IBM 162Ø ECAP manual GH2Ø-Ø17Ø-2.
F4Ø	--	The DEC FORTRAN-4 compiler, no longer current. See FORTRAN for more details. The current FORTRAN compiler is FORTRA.
FAIL	--	An advanced assembler (Modified Stanford version). See SRI Op. Note No. 26.1

GRIPE	93	A subsystem where a user can "gripe" about other subsystems.
GRPSTS	--	A subsystem which prints the current status of all the pie-slice groups and a per group summary of currently logged-in jobs. GRPSTS may be run at any time without affecting a user's current program.
HERMES	5-1	A message system for sending and receiving messages.
HG	6-1	A message reading program.
HOSTAT	256	Obtains the network site status information maintained by the network survey site (MIT-DMS as of Dec. 1, 1973). It then types this out in columnated form, grouped by status. See NETWORK Section of the TENEX Users' Guide.
HTYPE	7-1	A program for printing files on the Xerox 17ØØ printer.
IDDT	94	This is an extended version of DDT which debugs programs in an inferior fork.
IMGPTP	1Ø4	Copies binary file to paper tape.
LBLOCK	1Ø5	LBLOCK is a subsystem to perform "line-blocking" of text files.
LINK1Ø	1Ø6	A program which loads relocatable binary files. The current loader.
LISP	--	Symbol manipulating language. See INTERLISP Reference Manual.
LOADER	1Ø9	A program (no longer current) which loads relocatable binary files. DEC distribution, modified by BBN.

LOGOS	110	An interpreted procedure-based language with strings as its fundamental data type.
LPTPLOT	113	A program which permits a fairly coarse X-Y plot of data contained in any ASCII disc file to be prepared for listing on the line printer.
MACRO	115	DEC Assembler modified by BBN. For further information on MACRO refer to DECsystem-10 MACRO Manual and TENEX Users' Guide.
MAILBOX	8-1	A mailbox finder to find the appropriate user name and site an individual's mail should be sent.
MAILER	116	The subsystem that delivers queued mail.
MAILSTAT	118	Lists all queued mail in connected directory.
MAILSYS	--	A New mail system soon to be documented.
MIDAS	120	Project MAC Assembler. See Project MAC AI-Memo #90.
MINCOP	121	A program for copying MINIDUMPER (same as DUMPER) format tapes with optional listing of tape contents and optional comparison of the copies.
MRUNOFF	9-1	A program that will produce a formatted manuscript from a source file.
MSG	10-1	A program for reading, writing message file format.
MTACPY	122	A Magtape producing tape image files.

NCCL	--	The Command Control Language is intended to speed program development by reimplifying communication between a user and the systems programs. See <DOCUMENTATION> NCCL.INFO. NCCL defaults to current versions of all programs. CCL defaults to F40, the old FORTRAN-4, but is otherwise identical to NCCL.
NETDMP	--	A modified version of DUMPER which dumps and loads to a file which may be a network file.
NETED	258	An ARPA Network common editor which new users can pick up quickly. See NETWORK section of the TENEX Users' Guide.
NETSTAT	269	Program to print information about the status of the ARPA Network. See NETWORK section of the TENEX Users' Guide.
NOTIFY	125	Sends messages to selected teletypes.
PA1050	126	Compatibility -- Simulates 10/50 Monitor
PAL10	128	PDP-8 Assembler.
PAL11X	--	PDP-11 Cross Assembler for TENEX for further information, see the PAL11X Manual.
PCSAMP	131	A program to measure the operation of other user programs.
PPL	132	PPL is an interactive, extensible programming language. Reference PPL USER'S MANUAL, 6 Sept 1972, Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory.
PTIP	11-1	Instructions on use of a terminal that is connected to the BBN PTIP.

RD	135	Subsystem used to look at and manipulate the mail entries in your MESSAGE.TXT file (see also READMAIL & SNDMSG).
READMAIL	137	A subsystem which allows you to read your MESSAGE.TXT files.
RELRIM	138	Converts absolute REL file to RIM1ØB paper tape.
RENBR	139	Renumbering program.
REPORT	148	Allows interrogation of a data base which contains a running commentary on the status of the BBN-TENEX system.
RJS	--	Remote Job Service.
RSEXEC	28Ø	Experimental multi-computer Executive Program. See NETWORK section of the TENEX Users' Guide.
RUNFIL	149	Program to use a file instead of teletype to supply an input command stream.
RUNOFF	151	A document preparation program.
SDDT	--	A version of DDT which is called into service by the EXEC @DDT command when no user-defined symbols are available. See DDT writeup.
SECURE	--	A subsystem which allows users opportunity to lock their terminal when away from their office, so no one can destroy their programs.
SNDMSG	165	Sends message to any user.
SNOBOL	171	SNOBOL is a PDP-1Ø version of the string processing language, SNOBOL4.

SORT	172	SORT is a column-oriented text file sorter. See also the COBOL section of the User Guide for further details.
SOS	12-1	A line number oriented editor for text files.
SPELL	13-1	A program designed to read text files and check them for correctness of spelling. In addition to the spelling check, the program provides a means for correcting words that it thinks are misspelled.
SRCCOM	183	A program to compare edited files.
SYSDPY	--	Displays system statistics, WATCH, etc.
TAINT	184	This is not a TENEX program. It is a TOPS1Ø program which reads TENEX MINI-DUMPER tapes. It is supplied to installations running a TOPS1Ø system with a need to read such tapes. Complete documentation on TAINT is supplied in RUNOFF format, with the program itself. TAINT's operation is conversational and self-explanatory.
TAPCNV	187	Converts IBM formatted card image tapes to be compatible with TENEX.
TAPRD	--	A magnetic tape program for reading tapes.
TECO	189	TENEX TECO is a BBN created editor. See TENEX TECO Manual for further information.
TELNET	296	Telnet is a TENEX subsystem which provides a user with access to host computers via the ARPA network. It provides the user with the capability to communicate with a remote computer as if he were using a terminal at the remote site. See NETWORK section of the TENEX Users' Guide.
TIPCOPY	316	A subsystem which provides a means of sending TENEX text files to a TIP port via a separate data connection. See NETWORK Section of the TENEX Users' Guide.

ALGOL

The ALGOL Compiler responds by typing an asterisk on the user´s terminal. The user then types a command string to the compiler, specifying the source file(s) from which the program is to be compiled, and the output files for listing and output of relocatable binary. The command string, in common with other PDP-10 compilers, takes the form:

```
     OUTPUT-FILE,LISTING-FILE=SOURCE-FILES
```

A file takes one of the forms

```
     DEVICE:FILE-NAME.FILE-EXTENSION
```

or

```
     DEVICE:FILE.NAME
```

for directory devices (disk and DECtape)

or

```
     FILENAME.FILE-EXTENSION
```

or

```
     FILE-NAME
```

where DSK is assumed to be a default device.

In the case of non-directory devices, the format is simply

```
     DEVICE:
```

In cases where no FILE-EXTENSIONS are specified, the standard defaults REL for the relocatable binary output file, LST for the listing file, and ALG for the source file are assumed.

SOURCE-FILES

consists of one file or a list of files separated by commas. If a DEVICE is specified for the first file, and not for succeeding files, the second and following files are taken from the same device as the first.

Example:

```
     EULER,TTY:=EULER
```

[read source from DSK:EULER.ALG, write relocatable binary on DKS:EULER.REL, and listing on the user´s terminal].

```
     MTAØ:,DSK:SIM26=SIM26,PARAM.TST
```

[read source from DSK:SIM26.ALG, DSK:PARAM.TST, write relocatable binary on device MTAØ, and listing on file DSK:SIM26.LST].

Certain switches may be set by the user in the command string. These are:

L	list source program
N	no listing of source program
nD	(where n is an unsigned decimal integer) set the dynamic storage region for own arrays etc. (known as the "heap") to n words.

These switches are set by preceding them with a / after a file, for example:

```
     PROD,PROD/1ØØØD=PROD1/L,PROD2/N
```

causes file PROD1.ALG to be compiled with listing, file PROD2.ALG to be compiled without listing, and causes the size of the heap to be set to 1ØØØ words.

The ALGOL compiler reports all source program errors both on the user's terminal and in the listing device (if it is other than the terminal). After compiling a program the compiler returns with another asterisk, whereupon the user may compile another program, or type ^C to return to monitor level.

BASIC

The following is a short description of some of the most commonly used BASIC commands. For more information, see the DECsystem10 BASIC manual.

BYE Logs the user's job off the system.

CATALOG DEV:

Lists onto the user's terminal the names of the user's files which exist on the specified device. Ex: CATALOG DTA4:

COPY DEV:FILENM.EXT > DEV:FILENM.EXT

Copies the first file onto the second.

DELETE LINE NUMBER ARGUMENTS

Erases the specified lines from core. For example:

```
    DELETE 11,44-212,13
```

erases line 11, lines 44 through 212, and line 13. If no arguments are specified, an error message is returned. (It is not necessary to use the delete command to erase lines. You can erase a line simply by typing its line number and then depressing the return key.)

LIST LINE NUMBER ARGUMENTS

Lists the specified lines of the program currently in core onto the user's terminal. The line number arguments are of the form described under the DELETE command. If no arguments are specified, the entire program is listed.

NEW DEV:FILENM.EXT

The program currently in core is erased from core and the specified FILENAME is established as the name of the "PROGRAM CURRENTLY IN CORE". N.B., at the end of execution of this command, no lines exist in core.

OLD DEV:FILENM.EXT

The program currently in core is erased from core and the specified file is pulled into core to become the new "PROGRAM CURRENTLY IN CORE".

QUEUE FILENM.EXT

Queues the specified file from the user´s disk area for output to the line printer. Two optional switches available with this command are /UNSAVE and /&COPIES, where & is a number from 1 to 63. The switches follow the FILENM.EXT argument; for example:

```
QUEUE OUT.A/UNSAVE/2COPIES
```

REPLACE

See SAVE.

RESEQUENCE

Changes the line numbers of the program currently in core to 10,20,30,.... (line numbers within lines (as, GO TO 1000) are changed appropriately.).

RUN

Compiles and executes the program currently in core.

SAVE DEV:FILENM.EXT

Writes out the program currently in core as a file with the specified name. BASIC will return an error message if SAVE attempts to write over an existing file; to write over a file you must type "REPLACE" instead of "SAVE".

SCRATCH

Erases from core the program currently in core.

SYSTEM

Exits from BASIC to MONITOR level. N.B., the contents of core are lost.

UNSAVE DEV:FILENM.EXT

Deletes the specified file. More than one file can be specified; for example:

```
UNSAVE DSK:ONE.F4, DTA4:TEST.BAK
```

In the commands above which accept such arguments, if "DEV:" is omitted, "DSK:" is assumed; if ".EXT" is omitted, ".BAS" is assumed; if "EXT" is omitted, a null extension is assumed. The

# SAVE, REPLACE, and UNSAVE commands allow the "FILENM" part of the
# argument to be omitted, in which case ".EXT" must be omitted also
# and the name and extension of the program currently in core are
# assumed.
#
# The keywords of commands (CATALOG, LIST, ETC.) may be abbreviated
# to their first three letters. Only the three letter abbreviation
# and the full word form are legal; intermediate abbreviations
# such as CATAL, for example, are not allowed. If an intermediate
# abbreviation is used, the extra letters will be seen as part of
# the command argument (because BASIC does not see blank spaces or
# tabs at command level.). For example: CATAL DSKB: would be
# seen as a request to catalog the device ALDSKB. An example of a
# legal abbreviated command is: CAT DTA4:
#
# Whenever BASIC finishes executing a command, it types "READY".
# It does not answer "READY" after deleting a line by the alternate
# method described under the DELETE command or after receiving a
# line for the program currently in core from the user's terminal.
(To insert or replace a line in the program currently in core,
# simply type the line and then depress the return key. BASIC
# distinguishes between lines (which must be stored or erased) and
# commands (which must be processed) by the fact that a line always
# begins with a digit (part of the line number) whereas commands
# never do.).

BCDTAP

BCDTAP reads or writes a 7 track, even parity BCD magtape file-by-file. BCD here means the code produced by an IBM Ø26 keypunch or equivalent. The program first asks:

```
MAGTAPE UNIT NO.=
```

to which the user replies "Ø" or "1" depending on the unit being used. The magtape on that unit is moved to its load point, then the program asks,

```
USE 556 BPI?(Y OR N)
```

A response of "Y" sets tape density to 556 bits per inch, while a reply of "N" causes the further question:

```
DESIRED DENSITY(2ØØ OR 8ØØ):
```

when response is complete and density is set, the program asks:

```
TO OR FROM MAGTAPE?(T OR F):
```

if the direction is <u>from</u> magtape, ("F" response) the program requests a target file name by:

```
OUTPUT FILE:
```

The correct response here is any writable ASCII file. Each record read is converted from Ø26 BCD code to ASCII, trailing blanks are suppressed and carriage return-line feed is appended, then the line is written to the output file.

If the file name supplied above is null, (just carriage return) then one file is skipped over on the magtape and the program again asks,

```
OUTPUT FILE:
```

At the completion of each file transfer, BCDTAP types out the number of characters read from the magtape.

On encountering two successive end-of-file marks, the program types out:

```
LOGICAL END OF TAPE.
```

It then rewinds the tape and exits.

If the specified direction of information transfer is <u>to</u> the magtape, the program requests a source file by:

```
     INPUT FILE:
```

the correct response here is any readable ASCII file. Each line is read from the input file, tabs are converted to spaces, the codes are converted from ASCII to Ø26 BCD, the line is filled out to 8Ø characters, and the result is written on the magtape as one record.

When the entire file is transferred, the program types out the number of characters written on the magtape.

If a null file name is supplied for the input file, (just carriage return) the program asks:

```
     DONE?(Y OR N)
```

If the response is "N", the program again asks for an input file. If the answer is "Y", the program writes eight successive end-of-file marks, rewinds the tape, and exits.

BEDIT

BEDIT is a program which allows a user to examine, modify, and create files which are interpreted as strings of bytes of arbitrary size. The "random byte I/O" nature of BEDIT operations restricts its use to disk files.

BEDIT has no in-core buffer as many text editors do. All access to files is by means of pointers into the files themselves. In TENEX files, the bytes are numbered 1, 2, 3,... The BEDIT byte pointers (one each for input and output files) refer directly to the byte to be referenced, i.e., set to 1 to have the next operation refer to byte 1. (Notice that this contrasts with the usual TENEX byte pointer convention, which is that a byte pointer points to the byte before the one to be referenced. The difference is not profound, just easier to use when confronted with specific byte numbers.)

Commands to BEDIT are single letters. Most of them require arguments, which are explicitly asked for by the program. Arguments are numbers or single letters. Numbers should be terminated by carriage return. Numbers are interpreted as decimal unless prefixed by a sign, which means octal. The RUBOUT key may be used at any time to return to the command input level. Typing a "?" at the command input level or at some other places in the program will produce a typed summary of what typeins are appropriate at that point.

I: Specify input file. If an input byte size has not been specified, it is requested. If an input file is already open, it is closed. On commands which imply an input file (C, T, L), if an input file has not been specified, this command is automatically performed.

O: Specify output file. If an output byte size has not been specified, it is requested. If an output file is already open, it is closed. On commands which imply an output file (C, E, Z), if an output file has not been specified, this command is automatically performed.

BEDIT

BEDIT is a program that allows a user to examine, modify, and create files which are interpreted as strings of bytes of arbitrary size. The "random byte I/O" nature of BEDIT operations restricts its use to disk files. BEDIT operates on both TENEX and TOPS-20, but editing your typein is a little odd on TOPS-20. On TOPS-20, file name type-in is edited with RUBOUT and ^U (for character-delete and line-delete), as is the usual TOPS-20 convention; however numbers typed to BEDIT are edited with ^A and ^Q, according to the (old) TENEX convention.

BEDIT has no in-core buffer as many text editors do. All access to files is by means of pointers into the files themselves. In BEDIT, the bytes are numbered slightly differently from normal TENEX usage, in that the bytes of the file are numbered 1, 2, 3... (The usual TENEX byte pointer convention is that the first byte is number 0.)

Commands to BEDIT are single letters. Most of them require arguments, which are explicitly asked for by the program. Arguments are numbers or single letters. Numbers should be terminated by carriage return. Numbers are interpreted as decimal unless prefixed by a # sign, which denotes octal. ^E may be used at any time to return to the command input level. Typing a "?" at the command input level or at some other places in the program will produce a summary of the typeins that are appropriate at that point.

I: Specify input file. If an input byte size has not been specified, it is requested. If an input file is already open, it is closed. On commands that imply an input file (C, T, L, X, P), if an input file has not been specified, this command is automatically performed.

O: Specify output file. If an output byte size has not been specified, it is requested. If an output file is already open, it is closed. On commands which imply an output file (C, E, P), if an output file has not been specified, this command is automatically performed.

S: Set byte size or pointer. Choice of Input file, Output file, or Both. Choice of Size or Pointer. The effect on the byte pointer of changing the byte size is as described in the TENEX-4 system memo. Failing that, try it yourself and see. For pointer typein only, user may specify "N", ".+N", or ".-N", where N is an integer. In the first case, the byte pointer is set to N. In the latter two cases, "." is interpreted as "current value of the byte pointer," so the effect is to skip it forward or back by N bytes. A byte pointer specified as -1 means to the end of file.

C: Copy bytes from input file to output file. User must specify number of bytes to be copied (0 or just a carriage return means until end of input file).

E: Enter bytes from TTY into output file. User must specify if the bytes to be typed in are to be interpreted as Floating point (only legal for byte size of 36), Decimal integer (here also, a # preceding the number will cause interpretation as octal), or Octal. Numbers may be separated by carriage return, line feed, space tab, comma, slash, or semicolon. Enter mode is terminated by typing alt mode (escape). In addition to typing in the values of individual bytes, the Enter command also permits a number of bytes of the same value to be specified by means of the construction "n@value", where "n" is a decimal integer (or octal if preceded by #), and "value" is a number in the Mode specified. For instance, to specify 19 bytes of pi, type 19@3.14159.

X: Search for a byte sequence. First, it asks you to specify the byte sequence, which you do in the same manner as the Enter command. Then it scans the input file, starting at the current position of the byte pointer. If the sequence is found, the pointer is left positioned BEFORE that sequence. If not found, the pointer is left at its original position.

P: Overlay. Identical to Enter, except that the input byte pointer is moved forward by the number of bytes put in the output file (or to the end of the input file, if that's encountered first). This command makes it easier to change one or more bytes of a file.

T: Type bytes from input file. User must specify mode (Floating point, Decimal integer, Octal, or Text) and number of bytes to be typed out. If the number is 0 (or just a carriage return), the typeout will continue until end of file is reached or a ^E is typed. This command does not change the input file byte pointer.

In Text mode, bytes greater than 177 octal are typed out in octal. If the user's terminal does not have lower case, then lower case characters are printed as upper case prefixed with a period. The following characters are indicated as shown below.

11	tab	*t	37	TENEX EOL	*n
12	line feed	*l	4Ø	space	*s
14	form feed	*f	177	rubout	*r
15	car. rtn.	*c			

L: List bytes from input file onto a listing file. Same as Type, except output to any TENEX file or device.

K: Klose output file. Must be confirmed by a carriage return.

F: File status. Gives the name, byte size, length (in bytes) of the given size), and value of the byte pointer for each file.

V: Verbose typeouts (initial setting).

N: Nonverbose typeouts where practical.

Q: Quit, closing input and output files. Must be confirmed by a carriage return. Returns to TENEX Exec. Typing CONTINUE will resume BEDIT.

BLISS-10 COMMAND STRINGS

BLISS-10 does not use the standard -10 command scanner. However, command string interpretation is similar to that of other -10 CUSPS.

```
        @BLIS10
         *RELFIL,LSTFIL_SRC1,SRC2,...
```

1. Each file descriptor [RELFIL,LSTFIL,SRC] has the form: DEVICE:FILENAME.EXT

2. RELFIL receives the machine code generated by the compiler. If no code is desired, leave this position empty in the command string. If FILESPEC_ appears, FILESPEC is assumed to be the RELFIL spec and no listing output is generated.

3. LSTFIL receives the program listing produced by the compiler. If no listing is desired, leave this position empty in the command string.

4. SRC1,SRC2,... are the BLISS-10 source files which when concatenated together form one BLISS-10 module.

5. If "DEVICE" is omitted "DSK" is assumed.

6. If "EXT" is omitted, ".REL" is used for the RELFIL, ".LST" is used for the LSTFIL, and ".B10," ".BLI", and the null extension in that order are tried for the source file until either a file is found or all three defaults have failed.

7. SWITCHES: (-) implies that (switch name) will result in the opposite of the switch action. *indicates that the switch is assumed on by default.

#	NAME	ACTION
#	C	Generate a cross referenced listing.
#	E(-)	Expand all MACROs in the listing.
#	F(-)	Set up stack frame register (SREG) on every routine entry.
#	G(-)	All routines are to be made GLOBAL ROUTINES.
#	H	This entire module is to be loaded into the high segment.
#	I(-)	Generate special inspection word immediately prior to each routine or function body.
#	K	Disable listing of the source text (same as -L).
#	*L(-)	Enable listing of the source text.
#	M(-)	Enable listing of the machine code generated.
#	N(-)	Do not print error messages on the controlling TTY.
#	*O(-)	Optimize subexpressions across all ";"s.
#	R(-)	Do not save all declarable registers around an EXCHJ.
#	S	Output routine names as compiled and compilation statistics to TTY.
#	T(-)	Generate calls to a timing routine at the start and end of each routine.
#	U	Do not optimize across ";"s (same as -0).
#	V	Entire module is to be loaded into the low segment.
#	X	Perform a syntax only (no code generation) compilation.

TENEX CALENDAR SUBSYSTEM

Introduction

The CALENDAR subsystem is a program which is especially for people who like to keep lists of things to do daily. It is also useful for people who need reminders of things like birthdays, appointments...

Commands

The CALENDAR subsystem gets its commands from the terminal. The commands are very simple and are designed to handle those operations most frequently performed on the calendar data. They do not constitute a general set of primitives which one might consider the basis for programs to operate on calendar data. CALENDAR is not intended to be a programming language.

Entering Appointments, Deadlines, Things to Do

There are two primary commands for entering appointments, deadlines, and things to do in the data base; the Enter command and the Appointment command.

When the Enter command is invoked (by inputting E on the terminal), CALENDAR asks for the date which is terminated by carriage return. Dates are read by the TENEX time and date monitor calls which permit relatively free format to be used. If the date input is null (only a carriage return is typed), the current date is assumed. Whenever CALENDAR asks for a date, the preceding comments apply. Next CALENDAR types a task number (maximum of 256 tasks per day) to be assigned to this task or thing-to-do. This number is useful only for addressing the task at a later time, you need not remember it because it is always output when CALENDAR types a reminder or a task. Next calendar waits for the user to input an arbitrary length description of the task. Editing may be done on this description with control A or control Q consistent with standard TENEX editing. A carriage return may be put into the description for multi-line tasks. Tasks are terminated by Z or ESC (altmode key).

A normal Appointment is input by typing A on the terminal. CALENDAR will now ask for the date of the appointment, the number of days between reminders you want to see before the appointment, the total number of reminders you want to see, and finally some arbitrary length text about the appointment or deadline. It should be noted that appointments are considered by CALENDAR to be simply tasks which have some reminder criteria. All operations which can be done on tasks can be done to appointments.

Another type of appointment may be input via the Appointment command. This is a so called "forward" appointment for which you would like to see a reminder every specified number of days (up to a particular count) starting from a given date and going forward in time. This is specified by terminating the date field of an appointment command by a left arrow, carriage return (or Z or escape) sequence.

Forward reminders are distinguished in task typeouts (see List command) by an "_" preface. When the number of times a forward reminder has been given equals the requested count, the preface is changed to an "=" and the reminder is listed every day until it is explicitly marked as finished, cancelled, or deleted in the normal fashion.

The reminder typeouts are very diligent about reminding you when you asked. For example, if you asked for 4 reminders 4 days apart and then didn't use CALENDAR until 10 days before the deadline, you will get reminded this time as well as the very next time you use CALENDAR so that it can "catch up". This is accomplished by keeping a count of the actual number of reminders given. A reminder is unconditionally forced on the work day before an appointment or deadline. CALENDAR knows about weekends but not holidays...

If the input for the total number of reminders you want to see is null (just carriage return typed in), CALENDAR will put in a count sufficient for enough reminders to cover the entire period from the current day to the deadline at the frequency you specified.

Updating the Calendar Data

The Enter and Appointment commands will not have permanent effects on the data base until the Update command is given. This is true of all commands which modify the data base. The output by CALENDAR of a reminder modifies the data base and an Update should be done to have a permanent effect. Update requires a confirming carriage return before it is executed.

Relative Dates

When CALENDAR asks for a date, you can type in a date relative to the current date by typing a small number (1-9). If you want this to be a relative number of workdays (Saturday and Sunday will be skipped over), preface the small number with colon (:). For convenience, : alone is equivalent to :1.

The character "." is used to mean the most recently typed-in explicit date (i.e. 5/15/73 and null are explicit date type-ins, :3 is an implicit date meaning plus three work days. Explicit date typeins re-establish the meaning of "." Implicit date typeins have no effect on the meaning of "."). It is possible to use signed, small numbers as arguments to the ":" relative date operator. Signed small numbers also work as arguments to "." the sequence :.-2 is interpreted to mean two work days before the last explicit date typed in. Most combinations of the date operators which make sense are accepted and do what you might expect.

Listing the Calendar Data

Selected portions of the data base are listed by the List, Total list, and Individual list commands. The normal command used is List. List asks for a date, and normally a null input should be typed which means to use the current date and to invoke the standard reminder option. This will result in the current date and time being output followed by a list of reminders (if any) followed by a list of tasks (if any).

Each reminder or task is prefaced by the date of the task and its task number; then the text associated with the task is output. The preface date field is omitted if the task is for the current date. Reminders and tasks are always output in chronological then increasing task number order. Tasks should be marked as either Finished, Cancelled, Rescheduled, or Deleted to get them to disappear from the List printout. If you don't do one of these, the List printout of the task is repeated in the task list ad nauseum. The List output can be directed to a file by using the Output command. Output is initially set to the terminal. The standard reminder option causes the reminder counts to be updated. Recall this is invoked by inputting a null date. If List is given a date input, the reminder counts are not updated; instead, the CALENDAR program outputs reminders and task lists as though the current date were the date that was input. Note that if this is a future date, all unfinished tasks through that date are output. Appointments and deadlines are both treated as tasks (except that they are prefaced by an "!" for normal appointments or "_" or "=" for "forward" reminders), and this becomes obvious as they jump from the reminder to the task list as the List date is the same or greater than the appointment date.

There are times when you want to see only the tasks for a given date with no reminders of old, unfinished tasks. This is done by using the Individual list command which is otherwise exactly like List. There are also times when you want to see everything including Finished tasks and Cancelled tasks. This is accomplished by using the Total list command. With this command a single character indicator prefaces finished tasks (F) and cancelled tasks (C). Appointments are prefaced by three numbers of the form (l,m,n) where l is the number of days between reminders, m is the number of reminders, and n is the current count of reminders issued.

Format Control

CALENDAR normally outputs multi-line tasks in a "balanced format" which tests to see if words will fit on lines and inserts a carriage return,line feed if the word would over run the line. In the normal mode all explicit carriage returns are translated to spaces. This mode may be turned off by use of the "Balancing Format Switch ON (Y or N?: " command - answer that question N to turn the mode off. Within each task the user has some degree of multi-line format control. The first V (control V) encountered in a task turns balance formatting off. Each successive V complements the state of balancing format from off to on, on to off... Remember that V is the "quote next character" control so to input one, V must be itself input twice. Also, when balanced formatting is off, V is ignored.

The Print Command

The "Print" command is available for printing sections of the calendar data - such as for a month at a time. It prints a week at a time per page (which may over run a page for a busy week). On hard copy controlling terminals, it will await the input of any character (such as a space) between pages to allow the user to tear off output. Print expects the user to input a beginning and ending date for the period. If the first date is null (default for the current date), a standard "List" command for that first date will be executed including reminders and any incomplete tasks before the current date. If an explicit date is typed for the first date, a listing will be made for the first date only for that date with no reminders. Successive dates are typed with only the data for that date with no reminders.

Modifying the Calendar Data

Tasks or appointments are marked as finished with the Finished command which takes a task address of the form task number, date. They are marked as cancelled by the Cancelled command which takes a task address of the same form. The Delete command takes a task address and marks the space to be reclaimed during the update process. The Gain command is used to reclaim space in a similar fashion for all tasks with a date as old or older than the date supplied. Deleted and gained tasks literally disappear from the data base. Gain will optionally gain space only for completed or finished entries.

There is a provision for rescheduling tasks with the Reschedule command which marks the original entry as cancelled (it's still there) and makes a copy of the entry with the newly specified date.

An individual task may be itself modified with the "Modify" command. This expects a task number and date. It then invokes Teco which is used to edit the task. When you are finished, type ";H$(alt-mode)" to Teco to get back to CALENDAR.

Returning to the EXEC

The Quit command (if confirmed by a carriage return) will get you back to the EXEC. Quit closes any opened output file and switches output back to the terminal for subsequent CONTINUE's. CONTINUE will get you back into CALENDAR. Do not use re-enter; for one thing there is no re-enter address and TENEX won't let you.

Encrypting the Data

The Key command enables a change in the optional Key for the encrypting of the data base in a way that is believed to be relatively crack proof. The encrypting algorithm uses this key to form the basis or seed of a double word pseudo-random number generator sequence. Successive high order parts from the generator are exclusive-ored with the data thus encrypting the data. A checksum is included with the data to be encrypted. This is used on input to determine if the user typed in the correct Key.

If you are down on keys, passwords, and privacy..., then don't specify a key. When you first use CALENDAR it asks you whether or not you want the data encrypted. Of course you can always encrypt or change the key with the Key command. Don't forget the key! It is very likely the data will be impossible to decrypt without it! If you want to go from encrypted to non-encrypted data, type a null key (just CR) to the Key command.

With encrypted data, CALENDAR will ask you the key whenever you use the CALENDAR subsystem. If you mistype the key, you are returned to the TENEX EXEC.

Managing other CALENDARS

The Yank file command enables the user to specify another file for CALENDAR to work with. This can be a file in the connected directory or any other directory to which the user has access. Simply input a standard TENEX file name after the Yank file command is given. This command is most useful for a secretary to manage her supervisor's calendar. The dafault fields are set to "<connected-directory>CALENDAR.DATA;1".

Miscellaneous Commands

There is an Xor command which allows the undoing of deletes,finishes, cancels, etc. This is accomplished by specifying a task and date then typing D for delete,F for finished,C for cancelled in any combination to accomplish an exclusive OR of that operation with the present state of that operation for the specified task; thus, you can also mark a task, for example, as finished with this command.

Interrupting CALENDAR

The rubout key is enabled in CALENDAR to abort operations and output whenever it is safe to do so. Control C followed by CONTINUE is safe as with all subsystems. Reenter would not be safe and is not allowed.

The Data Base

The data base is stored in the users file CALENDAR.DATA;1. This is an ordinary TENEX file, and if it is deleted, it will go away in the ordinary way. (This is by contrast with MESSAGE.TXT;1). The file is read into the address space of CALENDAR when it starts up. This makes data base references very fast but limits the maximum useable size of the file to 246K data words. This is effectively infinite, but if it gets nearly full, the user will be so warned and expected to make use of the Delete and/or Gain commands. Of course it can be Total listed prior to doing this. Users should be aware that many files of this size tax disc space enormously. Since CALENDAR.DATA;1 is an ordinary file, accounting for it's space is done in the standard way.

The update operation was written to immunize the data base from crashes. Namely the file CALENDAR.NEW;1 is used to write the updated version then this is renamed to CALENDAR.DATA;1. Of course like any file, it may be wise to back it up and/or list it periodically.

```
Calendar 6-11-73     THU 12/27/73 15:26:11
Do you want to encrypt your data (Y or N)?: Y
Key is: COMPACT
If you forget this key, your data is likely to be irretrievably  lost!  It
is printed here again, enclosed in square brackets.
[COMPACT]
#Appointment or deadline for date: January 7, 1974
Number of days between reminders: 1
Number of reminders: 3
Task(1): 1400 PLANNING MEETING
#Enter task for date: 12/31/73
Task(1) TURN IN TIME SHEET$
#Enter task for date: .
Task(2): FINISH INPUT FOR ARPA QPR$
#Enter task for date: .+1
Task(1): HAPPY NEW YEAR, OUT ALL DAY TODAY$
#List tasks date: 12/28/73
THURSDAY, DECEMBER 27, 1973 15:29:49-EST for FRIDAY, DECEMBER 28, 1973
#List tasks date: 12/31/73
THURSDAY, DECEMBER 27, 1973 15:29:59-EST for MONDAY, DECEMBER 31, 1973
     1 TURN IN TIME SHEET
     2 FINISH INPUT FOR ARPA QPR
#List tasks date: 1/1/74
THURSDAY, DECEMBER 27, 1973 15:30:12-EST for TUESDAY, JANUARY 1, 1974
12/31/73 1 TURN IN TIME SHEET
12/31/73 2 FINISH INPUT FOR ARPA QPR
         1 HAPPY NEW YEAR, OUT ALL DAY TODAY
#Individual Listing date: Jan 3, 1974
Want to see F and C entries (Y or N)?: N
Want to see reminders (Y or N)?: Y
THURSDAY, DECEMBER 27, 1973 15:30:42-EST for THURSDAY, JANUARY 3, 1974
#Individual Listing Date: Jan 4, 1974
Want to see F and C entries (Y or N)?: N
Want to see reminders (Y or N)?: Y
THURSDAY, DECEMBER 27, 1973 15:30:53-EST for FRIDAY, JANUARY 4, 1974
**Reminders**
! 1/07/74 1 1400 PLANNING MEETING
#Update [Confirm]
#Quit [Confirm]
@;NOW CALENDAR IS CALLED AT A LATER SESSION
@CALENDAR
 Calendar 6-11-73     THU 12/27/73 15:31:35
Key is:
#Cancel Task Number: 1 Date: 1/7/74
#Reschedule task number: 2 Date: 12/31/73
to new date: 1/2/74$
#Update [Confirm]
#List tasks date: 1/2/74
```

```
THURSDAY, DECEMBER 27, 1973 15:33:15-EST for WEDNESDAY, JANUARY 2, 1974
12/31/73 1 TURN IN TIME SHEET
 1/Ø1/74 1 HAPPY NEW YEAR, OUT ALL DAY TODAY
         1 FINISH INPUT FOR ARPA QPR
#Quit [Confirm]


@
```

```
#          1 FINISH INPUT FOR ARPA OPR
# #Quit [Confirm]
# @
```

COBOL

COBOL (COmmon Business Oriented Language) is a large DEC-supplied compiler and operating system. Full documentation on COBOL will be found in the DECsystem10 COBOL manuals.

The COBOL user can create his programs and data on-line using one of the system editors -- TECO or LINED. He can transfer his program and data files from different media by means of PIP or FILEX. After the COBOL compiler translates the source programs into relocatable binary code, the linking loader of the loads the compiled programs and assigns addresses to the relocatable code.

The COBOL system offers a group of utility programs to aid the COBOL user in doing some of his COBOL-oriented tasks. These programs are:

LIBARY	-- to prepare and maintain source libraries
SORT	-- to sort user data in stand-alone mode
RERUN	-- to restart a program from a user-specified checkpoint
ISAM	-- to build and maintain indexed sequential files
COBDDT	-- to enable the COBOL programmer to debug COBOL programs at source level (both interactively and in batch mode)

COBOL Command Strings

```
@COBOL
*RELFIL,LSTFIL=SRC1,SRC2,.../SWITCH/SWITCH
```

1. Each file descriptor has the form:
   device:filname.ext[project,programmer]/switch/switch...

2. RELFIL receives the machine code generated by the compiler. If no code is desired, replace "RELFIL" by "-".

3. LSTFIL receives the program listing produced by the compiler. If no listing is desired, replace "LSTFIL" by "-".

4. SRC1, SRC2, ... are the COBOL source files required to produce one input program.

5. If "DEVICE:" is omitted for RELFIL or LSTFIL, "DSK:" is assumed. If "DEVICE:" is omitted for any source file, either the preceding device name is used, or "DSK:" is used if there was no preceding device name.

6. If the filename for RELFIL or LSTFIL is omitted, the filename of the first source file is used.

7. If ".EXT" is omitted from the RELFIL descriptor, ".REL is used. If ".EXT" is omitted from the LSTFIL descriptor, ".LST" is used. If ".EXT" is omitted from any source file descriptor, ".CBL" is assumed.

8. Switches:

A	List the machine code generated in the LSTFIL.
C	Produce a cross-reference table of all user-defined symbols.
E	Check program for errors, but do not generate code.
H	Type description of COBOL command strings and switches.
I	Suppress output of start address. (Program is to be used only by CALL´s.)
J	Force output of start address in spite of the presence of subprogram SYNTAX.
L	Use the preceding source file as a library file whenever a COPY verb is encountered. (If the first source file is not a /L file, LIBARY.LIB is used as the library file until the first /L file is encountered. (The default extension for library files is ".LIB".)
M	Include a map of the user defined items in the LSTFIL.
N	Do not type compilation errors on the user´s TTY.
P	Production mode. Omit debugging features from RELFIL.
R	Produce a two-segment object program. The high segment will contain the Procedure Division; the low segment all else. When the object program is loaded with the linking loader, LIBOL.REL will be added to the high segment.
S	The source file is in "conventional" format (with sequence numbers in cols. 1-6 and comments starting in col. 73).
W	Rewind the device before reading or writing. (Magtape only.)

Z	Zero the directory of the device before writing. (DECtape only.)

LIBARY

LIBARY -- Editor for creating, maintaining,
and listing COBOL library files

Command String Format:

```
@ LIBARY
*outputfil,listfil=inputfil
```

Notes:

If listfil is not specified, no listing is produced.

If inputfil is omitted, it is assumed that there is no input library. In this case only insertions can be done.

If the device-name is omitted from any field, "DSK:" is assumed.

If the extension is omitted from the output file or the input file specification, ".LIB" is assumed.

If the extension of the listing file is omitted, ".LST" is assumed.

If the input file and the output file have the same filename and extension, and both are on disk, the extension of the input file is changed to ".BAK" at the end of the run.

Switches:

/Z	Clear output device directory (for DECtape only)
/W	Rewind (magtape only)

SORT

SORT Command Strings:

```
@SORT n
*outfil/sw1/sw2...=infil/sw3/sw4...
```

```
# Notes:
#
# If the core argument is specified, nK core is used for the sort;
# otherwise SORT uses half of available user core for the
# sort buffer.
#
# If the device name is omitted from either file specification, it
# is assumed to be DSK:.
#
# For multireel magtape input or output, specify multiple devices,
# e.g., *MTA1:MTA1:MTA1:=MTA2:MTA2:MTA2:.
#
# The command string can be continued on another line by placing a
# hyphen at the end of the current line.
# *@dev:cmdfil.ext    causes commands to be obtained from the specified
#                     command file. (Default extension: ".CCL")
#
# Switches:
# /A         The associated file is recorded in ASCII.
#
# /Bn        A logical block of the associated file contains n
#            records (n is a decimal number). If the /B switch
#            is omitted for any file, that file is assumed to
#            be unblocked.
#
# /Kabcm.n     Defines a sort key as follows:
#
#            a             If the field is not numeric, this
#                          parameter is ignored.
#            a = S      The field is signed.
#            a = U      The field is unsigned; its magnitude only is
#  used.
#            a omitted  If the field is numeric, a = S is assumed.
#
#            b = X      The field is alphanumeric.
#            b = N      The field is numeric display.
#            b = C      The field is COMPUTATIONAL.
#            b = F      The field is COMP-1 (floating point)
#            b omitted  If parameter a also omitted, b = X is assumed.
#                         If parameter a is given, b = N is assumed.
#
#            c = A      The field is to be sorted in ascending order.
#            c = D      The field is to be sorted in descending order.
#            c omitted  c = A is assumed.
#
#            m             Relative position within the record of the
#                          first byte of the key.
#
#            n             Size of the field in bytes (digits if numeric).
#
#            More than one key can be entered with a single /K by
```

	separating the key descriptors with commas (e.g., /Kabcm.n,abcm.n...). The keys are sorted in the order that they are entered in the command string.

/Lam	Specifies the labeling convention for any file as follows:
	
	a = S Labels are standard.
	a = O Labels are omitted.
	a = N Labels are non-standard.
	
	m Specifies the size of a non-standard label in bytes.
	
	If the /L switch is omitted for any file on a directory device, standard labels are assumed. If it is omitted for any other file, labels omitted is assumed.

/Rm	Indicates the size of the record, where m is the number of bytes.
/S	The associated file is recorded in SIXBIT.
/Tdev	Indicates that the specified device is to be used as a scratch device for the sort. As many as 6 devices can be specified (e.g., /Tdev1,dev2,dev3,dev4).

If neither the /A nor the /S switch is specified for the input file, /S is assumed if there are any COMP or COMP-1 keys; otherwise /A is assumed.

If neither /A nor /S is specified with the output file, the mode of the input file is assumed.

The /K, /R, and /T switches can appear anywhere in the command string. The other switches must follow the files to which they apply.

RERUN

1) Insure that the state of the system is what it was at the dump time; i.e., all disk files used by the program are present, and all mag-tapes that were being used are mounted.

2) Start RERUN by typing, to the MONITOR: @RERUN

3) RERUN will type: TYPE CHECKPOINT FILENAME
user responds with FILE.CKP where ´FILE´ is the name of COBOL program that created the CHECKPOINT file.

4) RERUN may type: ASSIGN DSK LOGNAM

```
                         TYPE CONTINUE WHEN DONE
    user responds by assigning the logical name, then types CONTINUE.
```

5) After reloading the COBOL program, RERUN turns control over to that program.

ISAM

ISAM - Indexed Sequential File Maintenance Program

```
          /B Mode
                    Build Indexed File from Sequential Access File
```

```
@ISAM
*indexfil,isamdatafil=sequfil/B
```

This mode is assumed by default if no mode switch is supplied.

If the device name is not specified for any file, "DSK:" is assumed.

The default extensions are, respectively, ".IDX", ".IDA", and ".SEQ".

If the ISAM data file name is omitted, the name of the index file is used. If both the index file name and the ISAM data file name are omitted, the name of the input file is used for them.

Answers to questions:

```
MODE OF INPUT FILE: S(IXBIT) or A(SCII)

MODE OF (ISAM) DATA FILE: S(IXBIT) or A(SCII) (may differ from input)

MAXIMUM RECORD SIZE: (size of largest record of input file
                           in bytes)

KEY DESCRIPTOR: sxm.n
                where    s = S indicates the key is signed
                         s = U indicates the key is unsigned
                         x = X indicates the key is alphanumeric
                         x = N indicates the key is numeric display
                         x = C indicates the key is COMPUTATIONAL
                         x = F indicates the key is COMP-1
                                    (floating point)
                         m = the number of the byte in the record
                                     where the key begins
```

n = the size of the key in bytes or digits

RECORDS PER INPUT BLOCK: number of records per logical block of the input file (Ø if unblocked)

SIZE OF LARGEST INPUT BLOCK: number of characters per block (asked only if input file is unblocked and on magtape)

TOTAL RECORDS PER DATA BLOCK: number of records per logical block of the ISAM data file

EMPTY RECORDS PER DATA BLOCK: number of records to initially leave empty in each data block (to facilitate later random insertions)

TOTAL ENTRIES PER INDEX BLOCK: number of index entries to be contained in each logical block of the index file

EMPTY ENTRIES PER INDEX BLOCK: number of entries to initially leave empty in each index block

PERCENTAGE OF DATA FILE TO LEAVE EMPTY: essentially, this specifies number of additional empty blocks to be initially added to the file (in order to speed up later growth)

PERCENTAGE OF INDEX FILE TO LEAVE EMPTY: similar to above

MAXIMUM NUMBER OF RECORDS FILE CAN BECOME: a number in excess of what the file is ever likely to grow to

/M Mode
Maintain Existing Indexed File

```
@ISAM
*outputindexfil,outputdatafil=inputindexfil/M
```

Default devices are all "DSK:". Default extensions are, respectively, ".IDX", ".IDA", and ".IDX". Default filenames are as with the /B mode.

Answers to questions are the same as for /B, except that only the last 5 questions are asked, and the existing values for these parameters are typed in parentheses. Any of these parameters may be left unchanged by typing just carriage-return.

/P Mode
Pack Indexed File Back into a Sequential File

```
# @ISAM
#
# *sequfil=indexfil/P
#
# Default devices are "DSK:". Default extensions are
# ".SEQ" and ".IDX", respectively.
# If the sequential output file name is omitted, the name of the
# index file is used.
#
# Answers to questions:
#
# MODE OF OUTPUT FILE: S(IXBIT) OR A(SCII)
#
# RECORDS PER OUTPUT BLOCK: blocking factor of output file
#
# SIZE OF LARGEST OUTPUT BLOCK: number of characters per block
#           of the output file
#           (asked only if output is on magtape and unblocked)
#
# Indirect commands:
#
# @ISAM
# *@commandfil.ext
#
#
#                                 COBDDT
#
# Load COBDDT as follows:
#
#
# @ LINK/Ø
# *userprogram/DEBUG:COBOL/G  or
#
# @ LOADER
# */Suserprogram,SYS:COBDDT$
#
# When program is executed, it will first type "STARTING COBOL DDT".
# The user may then set breakpoints, examine locations, etc., before
# proceeding.
#
# COBDDT Commands:
#
# ACCEPT data-name                 (accept data typed on next line
#                                        as new value for specified
#                                        data item)
# ACCEPT                                (assumes data-name of last
#                                        DISPLAY or ACCEPT)
# BREAK paragraph or section-name       (set breakpoint)
# CLEAR paragraph or section name       (clear breakpoint)
# CLEAR                                 (clear all breakpoints)
# DISPLAY date-name
# DISPLAY                          (assumes data-name of last
#                                    DISPLAY or ACCEPT)
# MODULE program-name          (use symbol table of named program)
# PROCEED                          (proceed from breakpoint)
# PROCEED n                       (proceed to nth occurrence of
#  breakpoint)
# STOP                             (stop run)
# TRACE ON                        (type each paragraph or section
#                                   name as it is encountered)
# TRACE OFF
# WHERE                            (list all breakpoints)
```

All command verbs and arguments may be truncated, so long as the
part type is enough to identify that item.
#
#
#
COBRG
#
COBRG -- COBOL Report Program Generator
#
Since the implementation of the Report Writer module in
DECsystem-10 COBOL, COBRG has been rendered unnecessary.
However, for a time COBRG will continue to be supported so that
old user programs written in this language can be maintained. It
is recommended that new report programming be done with the
Report Writer module.
#
The output from COBRG consists of one or more COBOL source files
and a listing file. The name of each COBOL source file is that
given in the NAME specifications contained in the input file,
with ".CBL" as extension. The listing file contains a list of
input specifications for each file generated, plus any error
diagnostics. All output is always to DSK:. Also the input file
must be on DSK:. Command String Format:
#

```
# @ COBRG
# *listfilinputfil
```

#
Defaults:
#
If the name of the listing is omitted, the name of the input file
is used. If the listing-file extension is omitted, ".LST" is
assumed. The input-file name and extension must be specified.

COPYM

COPYM (COPY Multiple) is a program designed to facilitate copying groups or lists of files from place to place. It will accept a file containing a list of files, or information may be entered from the controlling terminal. It will copy each input file to a similarly named destination file, e.g. <JONES>FOO.MAC to <SMITH>FOO.MAC.

It has particular features for copying to DECtape; it keeps track of the amount of free space remaining on the DECtape and will ask for another DECtape to be mounted if the next file to be copied won´t fit.

The copy command of the EXEC may eventually be modified so as to dominate all of the functions available with COPYM, but until then, COPYM should be useful.

Operating Procedure

COPYM first requests the name of a file from which to obtain the source file descriptors. TTY: may be used for this.

The source file names, whether from a file or a terminal, may contain *´s in any fields except the device name, e.g. DTA1:*.MAC. If the source device is a multiple directory device (DSK:), a directory name may be entered and will be used for all source files until a subsequent directory name is entered.

COPYM next requests the output designator (see Limitation 1). Use of *´s here is permitted and should be appropriate to the form of the input designator. For example, if the input is to be *.MAC;*, then *.FOO;* and *.*;* are both appropriate. A * here means the same string for the field as in the source file, therefore, if a * is used in a field of the source designator, one should also be used in the corresponding field of the output designator.

If the output device is DECtape, COPYM will next ask whether or not the directory is to be cleared before beginning the copying (2).

Answer Y or N.

COPYM will then begin processing input file descriptors. If input file names are being entered on the terminal, a ready character will be typed each time COPYM is ready for another descriptor. Each of the files identified by the source descriptor will be copied to a file on the output device having a name constructed by replacing the *´s of the output descriptor with the strings from the corresponding fields of the actual source file being copied. The name of each actual source and destination file is reported as copying proceeds.

If the output device is DECtape and the tape becomes full, COPYM will again ask for an output designator, and when one is supplied, copying will be resumed.

Current Limitations

The following limitations exist in the current version, but should be removed in a future version.

1. The output designator may consist only of a device name, e.g. DTA1:, DSK:. The program behaves as if all other fields were specified as *.

2. If the directory of a DECtape is not cleared, COPYM will not know how full it is, and so the facility which detects the condition of insufficient space for next file will not operate correctly.

3. SSAVE files created by TENEX before version 1.29 could not be sequentially copied, and hence could not be placed on DECtape. Therefore, if the source file extension is .SAV, COPYM does a GET of the source file into an inferior fork. Then, if the destination device is DTA:, it does a SAVE of the entire core image onto the output file. If the destination device is DSK:, it does an SSAVE of the entire core image. This works correctly for all but a few specially constructed types of SSAVE files, e.g. LISP
# SYSOUT's.

CREF

CREF produces a sequence-numbered assembly listing followed by one to three tables, one showing cross references for all operand-type symbols (labels, assignments, etc.), another showing cross references for all user-defined operators (macro calls, OPDEFs, etc.), and another (if the proper switch is specified) showing the cross references for all op codes and pseudo-op codes (MOVE,XALL, etc.) A number sign (#) appears on the definition line of all symbols. The input to CREF is a modified assembly listing file created during a MACRO-1Ø assembly or FORTRAN IV compilation when the /C switch is specified in the command string.

Detailed information on CREF is contained in the DECsystem1Ø Assembly Language Handbook in the Utilities Section.

WARNING

The following changes have been made to the TENEX version of CREF:

1. After CREF has processed a source file, it is deleted from the user's directory.

2. The output listing from CREF is generated for wide line printer paper (132 characters per line). Unless wide paper is in the line printer, output to a disk file and request that the operator list the file on wide paper. (Narrow paper is standard for the TENEX line printer. Wide paper must be explicitly requested.)

3. If an output file name is specified without an output device, the default output device will be DSK. If neither an output device nor an output file name is specified, the default output device will be LPT.

FORMAT

Two input formats are acceptable to CREF. The first is produced by early versions of MACRO (prior to version 3Ø), the PALX assembler for the PDP-8, and early versions of FORTRAN (prior to version Ø6). The second input format for CREF is produced by current versions of MACRO (version 3Ø and later), version Ø6 and later of FORTRAN, FORTRAN-1Ø, and the Stanford FAIL assembler.

EARLY INPUT FORMAT

The codes listed below are produced by early versions of MACRO and FORTRAN as input to CREF. They are ignored by CREF if the control characters produced by the current versions of MACRO and FORTRAN are present.

ASCII Code	Meaning
33	Indicates that the code type is an op code, a pseudo-op code, or an op code defined by the user by OPDEF.
34	Indicates that the code type is a macro name.
35	Indicates the end-of-line.
36	Indicates a normal symbol; i.e., a symbol defined with an equal sign (=) or a colon (:).
37	Indicates a program break (between FORTRAN subroutines).

This input format to CREF should not be used when new programs are being developed.

CURRENT INPUT FORMAT

The control characters described below are placed on the listing produced by current versions of MACRO ,FORTRAN, and FORTRAN-1Ø as input to CREF.

Normally, each line of the listing contains CREF input data followed by the line of the listing. The CREF input data on each line is preceded by RUBOUT B and terminated by RUBOUT C. Each symbol or instruction type in the listing is defined in the CREF input by a control character (described below). The number of characters in each symbol or instruction is also defined by a control character. The set of control characters for defining symbols and instructions is identical to the set of control characters for defining the number of characters in the symbol or instruction. The position of a control character in the CREF input data determines the use of the control character. For example, in the input CREF data B^C^CENDC, the B indicates the beginning of the data, the first ^C defines the instruction END as a pseudo-op code, the second ^C defines the number of characters in the instruction END as 3, and the C terminates the

CREF data.

Optionally, CREF information may be terminated by either RUBOUT A or RUBOUT D. If RUBOUT D is used, more than one block of CREF information may be placed on a single line.

The control characters and their meanings are described below.

Beginning and Ending Control Characters

The control characters that begin and end the CREF input data are:

RUBOUT B (prints as B)	Signals the beginning of the CREF data on each line
RUBOUT C (prints as C)	Terminates the CREF data on each line and inserts the line number to the listing
RUBOUT A (prints as A)	Terminates the CREF data on each line and inserts the line number and a horizontal tab to the listing.
RUBOUT D (prints as D)	Terminates a block of CREF data in a line without inserting any additional information in the listing
RUBOUT E (prints as E)	Indicates program break (between FORTRAN subroutines)

Symbol-Definition Control Characters

The control characters that define symbols, instruction types, and macros are:

Character	ASCII Code	Meaning
CONTROL-A (^A)	ØØ1	Precedes each symbol that is defined with an equal sign (=) or a colon (:) each time the symbol appears in the listing; e.g., FOO:

CONTROL-B (^B)	ØØ2	Immediately follows the defining occurrence of the symbol defined by equal sign or colon.
CONTROL-C (^C)	ØØ3	Precedes the use of an op code (either hardware-defined or defined by OPDEF) or a pseudo-op code.
CONTROL-D (^D)	ØØ4	Precedes the defining occurrence of an op code defined by OPDEF.
CONTROL-E (^E)	ØØ5	Precedes a macro call.
CONTROL-F (^F)	ØØ6	Precedes the definition of a macro.
CONTROL-G (^G)	ØØ7	Precedes the definition of a line number
	Ø15	Signals the beginning of a FAIL symbol block.
	Ø16	Signals the end of a FAIL symbol block.

Although CREF recognizes and accepts all of the above control characters, current versions of MACRO do not produce all of these characters. As shown above, CREF recognizes a symbol defined by OPDEF as an op code because it is preceded by ^D when it is defined, and by ^C when it is used. MACRO treats a symbol defined by OPDEF as a macro and thus precedes it by ^F when it is defined, and by ^E when it is used. CREF also recognizes these symbols as macros because of the control characters produced by MACRO. The fact that symbols defined by OPDEF are treated as macros has no effect on the cross- reference listing from CREF because OPDEF's and macros are grouped into the same table.

Character-Count-Definition Control Characters

The octal value of the control characters described below is used by CREF to determine the number of characters in a symbol or instruction. The same set of control characters define the symbol as well as the number of characters in the symbol. The position of the control character in the input data determines the use. The character-count control character immediately precedes the symbol with no intervening spaces or characters (e.g., ^CEND). The control characters and their meanings are as follows:

Character	ASCII Code	Meaning
CONTROL-A (^A)	001	The symbol contains 1 character
CONTROL-B (^B)	002	The symbol contains 2 characters
CONTROL-C (^C)	003	The symbol contains 3 characters
CONTROL-D (^D)	004	The symbol contains 4 characters
CONTROL-E (^E)	005	The symbol contains 5 characters
CONTROL-F (^F)	006	The symbol contains 6 characters

No symbol or instruction can contain more than six characters.

Example of the Current Input Format

The example below shows a small MACRO program and the listing produced by MACRO to be input to CREF.

```
        ;CREF SPECIAL CHARACTER DEMONSTRATION
 .MAIN   MACRO 44.0 09:30 10-DEC-70 PAGE 1

        ;CREF SPECIAL CHARACTER DEMONSTRATION
        M=6                     ;1 CHAR SYMBOL DEFINITION
        MOVEI M                 ;5 CHARACTER OPCODE
                                ;1 CHAR SYMBOL USE
FOO:    SIXBIT /123/            ;3 CHAR SYMBOL DEFINITION
                                ;6 CHAR PSEUDO INSTRUCTION
        MOVEI 6+FOO             ;MORE OF THE ABOVE
OPDEF TTYCAL [51B11]            ;OPCODE DEFINITION
DEFINE TEST (X) <TLNE X>        ;MACRO DEFINITION
        TTYCAL                  ;OPCODE USE
        TEST M                  ;MACRO CALL & SYMBOL USE
        END                     ;PSEUDO INSTRUCTION OCCURRENCE

 .MAIN   MACRO 44.0 09:30 10-DEC-70 PAGE 1
TEST  .MAC
BC                                  ;CREF SPECIAL CHARACTER DEMO
B^A^AM^BC                           000006 M=6
```

```
#                                        **  ;1 CHAR SYMBOL DEFINITION
# B^C^EMOVEI^A^AMC         000000' 201000  000006 MOVEI M
#                                        **  ;5 CHARACTER OPCODE
# BC                                       ;1 CHAR SYMBOL USE
# B^A^CFOO^B^C^FSIXBITC 000001' 212223  000000  FOO;  SIXBIT /123/
#                                        **  ;3 CHAR SYMBOL DEFINITION
# BC                                       ;6 CHAR PSEUDO INSTRUCTION
# B^C^EMOVEI^A^CFOOC  000002'  201000  000007'   MOVEI 6+FOO
#                                          ;MORE OF THE ABOVE
# B^C^EOPDEF^F^FTTYCALC                  OPDEF TTYCAL [51B11]
#                                        **  ;OPCODE DEFINITION
# B^C^FDEFINE^F^DTESTC                   DEFINE TEST (X) <TLNE X)
#                                        **  ;MACRO DEFINITION
# B^E^FTTYCALC     000003'  005100  000000 TTYCAL
#                                        **  ;OPCODE USE
# B^E^STESTC                              TEST M
#                                        **  ;MACRO CALL _SYMBOL USE
# B^C^DTLNE^A^AMC     000004'   603000 0000006 TLNE M
# B^C^CENDC                                END
#                                        **  ;PSEUDO INSTRUCTION OCCURRENCE
# NO ERRORS DETECTED
# PROGRAM BREAK IS 000005
# 2K CORE USED
#  .MAIN   MACRO  44.0 09:30  10-DEC-70 PAGE 2
# TEST   .MAC     SYMBOL TABLE
# FOO                               000001'
# M               000006
# TTYCAL 005100   000000
```

DDT

DDT is the debugger for most of the TENEX/DEC language processors. It is described in detail in the DECsystem1Ø Assembly Language Handbook. The differences between TENEX DDT and DEC DDT are specified here.

1. The "undefined symbol" assembler has several improvements. "LOADER" leaves a table of undefined symbols pointed to by the contents of a ".JBUSY" (117). DDT now uses this same table (rather than its own separate table) for assembling undefined symbols. The result is that DDT may be used to define and automatically patch locations for symbols that LOADER said were "undefined externals". This will work correctly for "linked" references or for "additive requests" in either the right or left half of a location.

2. Using symbol# on a symbol that is already defined will give the ubiquitous ? message.

3. DDT will assemble undefined symbols only when:

    1. There exists a reasonable .JBUSY pointer.

    2. A register is open and being modified, ("symbol#=" is not valid.)

    3. Only the arithmetic operations plus or minus may be used on the symbol. Multiply or divide or parentheses may not be used.

4. The problem of user defined tags being confused with machine operation codes finally has a reasonable solution. An input symbol is considered to be a machine operation code (and searched for first in DDT´s OP code table, then in the user´s symbol table) if:

    1. It is the first symbol or number input as part of an expression, and

    2. It is terminated by a space.

    The input symbol is searched for first in the user´s symbol table (then in DDT´s OP code table) if the above conditions are not both true.

EXAMPLES: Suppose MOVE is defined to be 6 (6<MOVE:), then:

```
MOVE=6
MOVE =200000,,0
MOVE MOVE=200000,,6
MOVE MOVE =200000,,6
MOVE+MOVE=14
```

Note that "space" and "+" are not equivalent. As a general rule, use spaces and pluses the same way that they are used in MACRO. Also, to force a symbol to be interpreted as a machine operation code, type it first and terminate it with a space.

5. The form "symbol?" will list all the program names where "symbol" is defined. The program name will be followed by "G" if "symbol" is a global symbol.

6. "FOO 1,,FOO 1" will now give the same results as "FOO+1,,FOO+1". Note again that space and plus are not normally equivalent, but have been made to be so in this special case. People insist on the "FOO 1,,FOO 1" form even though the DDT manual doesn't allow it. Remember, MACRO won't allow it either.

7. The PC word flags are now saved and restored correctly for all cases in the breakpoint logic.

8. For the instructions JRST, JFCL, and XCT, the accumulator field is always typed out in numeric mode, not symbolic.

In addition to the above, the "GO" command ($G) has been improved. In general, commands with two altmodes such as FOO$$G clear the interrupt system before going to FOO. With a single altmode, the interrupt system is not affected.

If there is a small number between the altmode(s) and the G, and no argument supplied, the command will start the program at the specified entry vector location. Thus, $$0G is the equivalent of the EXEC command @START, while $$1G is the same as @REENTER, $3G starts the program at the third entry vector location.

If the fork has a DEC 10/50 style entry vector ("length" = 254000), only 0 or 1 is legal between the altmode(s) and the G. In such cases, the contents of location 120 is used for $0G and contents of location 124 for $1G for consistency with DEC conventions.

There are two special cases: $$G is an abbreviation for $$0G and will start the program at its normal start address. $G means the same as $I which is where the user's flags,,PC are stored while in DDT.

When DDT is entered, it will attempt to automatically set the program name (i.e., MAIN.$:) by finding which program contains the start address.

The following commands were added earlier in the history of TENEX:

$$Q	has the value of the last quantity typed with halves swapped.
$V	has the value of the left half of the last quantity typed.
$$V	has the value of the left half of the last quantity typed with sign extended.

thus

```
FOO/ -4,,3  $Q_ -4,,3 $$Q_ 3,,-4        $$Q_ -4,,3

FOO/ -4,,3  $$V_-4

FOO/ -4,,3  $V=        777774
```

For completeness, the following is a list of differences between BBN DDT and DEC DDT.

DEC DDT $G with no argument is done with $$G in TENEX DDT.

BBN DDT does not have paper tape commands: $J, ^R and $L.

BBN DDT uses the Stanford block structured symbol table code which permits debugging FAIL-assembled programs. The command FOO$& will make the symbols in block named FOO current. This is relevant only for FAIL-assembled programs.

CONTROL-A in DDT causes its argument to be taken as RADIX-5Ø SQUOZE code.

DOUBLE-QUOTE (") sets up to accept a string in the same way that the ASCII pseudo-instruction does in MACRO. That is, "/STORED TXT/" will exactly fill two PDP-1Ø words with ten 7-bit characters.

$$".FOO MUNG. This is the sixbit text storage command. The point (.) is a delimiter, as was the slash in the above example. This example will fill one and one-half words (9

characters).

"Q$= will type the octal value of ASCII Q.

"/Q/= will type a 36 bit number containing ASCII Q in the left seven bits.

Note that if a register is not open, DDT will not permit inputting more than one word´s worth of characters.

The BBN hardware instructions have been added to the OPCODE table.

In floating point type out mode ($F) unnormalized numbers are printed as ordinary decimal numbers with a decimal point.

DELVER

DELVER is a program for assisting in the management of file versions. It provides the ability to delete excess* versions of files according to the most frequently needed algorithms. DELVER will delete excess versions of files as specified by a standard TENEX file group designator. For each group of files, two options are provided. The oldest (lowest version number) version may be optionally deleted and the version numbered one less than the most recent (highest numbered) version may be optionally deleted. This provides the ability to save a version which may be closest to the most recent in case that version gets lost, and the ability to save a version which is the most likely base for a series of changes for source comparisons etc. If any version number is greater than one of most recent, second most recent, or oldest, (time-wise) it must satisfy all tests before it will be deleted.

To use DELVER, type DELVER<cr> to the TENEX EXEC. Then answer the two questions about whether or not to delete the second most recent and oldest versions with either Y or N depending on which options are desired. DELVER will then ask for the file group designator over which deletion is to occur. Default group designator is *.*;*, i.e. everything in the connected directory. Each file deleted is printed on the TTY. If DELVER deletes any files you do not wish to have deleted, the EXEC command UNDELETE may be used to correct the error. Note that if DELVER is being overzealous, the best way to stop it is to type a single control-C (ETX). This will allow printout of all files actually deleted to appear on the terminal. Using multiple control-C (ETX) will not stop the program any faster, but will clear the output buffer of the terminal and thus lose any record you might have of what has actually happened.

----

*(i.e. all but the most recent, second most recent, and oldest, with the optional exceptions described below)

DO

DO is a new subsystem for passing a parameterized text string from a specified file to the EXEC for execution. Parameters are indicated in the text file by a % character, followed either by

1. a digit, or

   string not containing the character, followed by the second instance of the character.

For each parameter, the user is asked for a text word (arbitrary text delimited by space, tab, or cr.) to substitute for each instance of the parameter in the input text string. He is prompted for this word by the digit, if a parameter of type 1, or by the text string, if a parameter of type 2. Each uniquely named parameter is prompted for once, even if it is used several times in the input text string. See the attached example.

The DO program simply dumps the expanded text string into the input buffer, then HALTF's. It behaves just as if the user had typed the expanded text string ahead to the EXEC. Therefore, Control-C can be used to clear the input buffer and return to the EXEC. Before stuffing the input buffer, the program waits, checks for other input characters from user type-ahead, and, if so, rings bells, waits again, buffers the characters, and appends them to the expanded text string. Thus, the user may type ahead, even while DO is running, and is warned to stop while the program is stuffing the input buffer.

NOTES:

1. A % character which is followed by another % character expands to a single % character (this is an escape convention for % characters in the expanded text string).

2. Beware: the TTY input buffer is currently limited in size; if the expanded text string is too long (>119 characters), the extra characters will get lost, just as if you had typed too far ahead. A change to TENEX to allocate more TTY input buffer space in such situations is planned.

```
# ;  <XBCPL>DO.EXAMPLE;3    WED 10-APR-74 3:49 PM              PAGE 1
#
# @teco.SAV;12908
#
# *;Y$
#
# INPUT FILE: EXAMPLE.;1 [confirm]
# 47 CHARS
#
# *zt$
# LOADER
# sys:bcplib
# dsk:%#Program#$ssa$$$%#Program#.sav
#
#
# *;u$
#
# OUTPUT FILE: EXAMPLE.;2 [New version]
#
# *;h$
# @do
#
# Input file: exAMPLE.;2 [Old version]
# Program: t
# @LOADER
# *sys:bcplib
# *dsk:t$
#
# FIRST 4K CORE, 461 WORDS FREE
# LOADER USED 7+4K CORE
#
# EXIT.
# ^C
# @ssaVE (PAGES FROM) 0 (TO) 777 (ON) t.sav [New version]
#                             DTACPY
#
#
#
# DTACPY copies full DECtapes to full DECtapes.  There is an option
# to  write  a  copy  of DTBOOT onto the front of a DECtape without
# having to copy it from another tape.  Also it  can  relocate  the
# bootstrap for any size of core.
#
# Functions by reading a REL FILE of DTBOOT, produced by assembling
# DTBOOT  with  REL==1,  and keeping it in core.  This file is then
# used and relocated when needed.
#
# DECtape.
#
# EXAMPLE:
#
# COPY TAPE OR NEW BOOTSTRAP (C OR N)?
#
#
# (which does the old COPY functions if you say "C" and does the
# new BOOTSTRAP function if you say "N".  If the BOOTSTRAP has
# not been put into DTBOOT, it will ask for it.  The version
# currently on SUBSYS has the BOOTSTRAP in it.
#
#
# COPY DTA1: (TO) DTA2: [CONFIRM],
# COPY BOOTSTRAP? Y
# COPY REST OF TAPE ? Y
# VERIFY? Y
# ^C
# @ST
#
#
# COPY DTA1: (TO) DTA2: [CONFIRM],
# COPY BOOTSTRAP? Y
# COPY REST OF TAPE? N
# VERIFY? N
# ^C
# @ST
#
#
# COPY DTA1: (TO) DTA2: [CONFIRM],
# ^C
# @
#
#
```

DFTP User's Guide

Overview

The Datacomputer is a shared large-scale data base utility offering data storage and data management services to other computers on the Arpanet. The system is intended to be used as a centralized facility for archiving data, for sharing data among various network hosts, and for providing inexpensive on-line storage for sites needing to supplement their local capability. The Datacomputer is implemented on dedicated hardware, and comprises a separate computing system specialized for data management. Logically, the system can be viewed as a closed box shared by multiple external processors and accessed in a standard notation called Datalanguage.

The Datacomputer File Transfer Program (DFTP) is a user-invoked program that stores and retrieves local files on the Datacomputer. DFTP translates simple user commands into Datalanguage, sends the Datalanguage and data to the Datacomputer, processes the messages and data returned from the Datacomputer, and notifies the user of the results. DFTP also manages local file input/output and secondary network connections to and from the Datacomputer.

The Directory

The DFTP Datacomputer directory is a tree, with site nodes anchored to a common root node, user nodes subordinate to site nodes, optional subdirectories of arbitrary depth and breadth beneath user nodes, and user files stored in special leaf nodes (called '<FILES>' nodes). Pictorially,

```
                       <ROOT>
                       /    \
                   SITE      SITE    ...
                  /    \
              USER      USER    ...
             /
       <FILES>     SUBDIRECTORY    ...
                  /            \
            <FILES>             SUBDIRECTORY    ...
                                            \
                                            <FILES>    ...
```

The <FILES> nodes are the repositories of all data. The user files they contain are not known individually to the Datacomputer (unlike nodes), but are separate entities only to DFTP. There can be only one <FILES> node directly under any given user or subdirectory node. DFTP users do not reference a <FILES> node directly; a reference to a file under a specific user or subdirectory node is expanded into a reference to that file in the <FILES> node under the specified node. (If a node is not specified a default is supplied).

There are two basic types of commands -- those that reference only nodes and those that reference user files. Node level commands operate at the global level of sites, users, and subdirectories. File level commands operate at the local level (inside a <FILES> node) storing, retrieving, and modifying data within that node. The argument to a file level command can consist only of a file name, or of a file name preceded by a node argument (such as the node level commands take).

Referencing Nodes

The mechanism for referencing a node, called a node path, consists of a context and a node list. A context is an anchoring point for node name references (indicated by one, two, or three left-angle-brackets); if none is specified, DFTP supplies a default. A node list is a sequence of node names, starting from the anchor, defining the desired branch of the directory tree.

There are three contexts, TOP, ATTACH, and CONNECT.

1. The top context ('<<<') anchors the node path at the root node and is used primarily for referencing other site and user nodes.

2. The attach context ('<<') is a node path, set by the ATTACH command (and by DFTP automatically at the beginning of a session), and usually indicates a user node. It is used mainly as a reference point for name space division beneath the user node.

3. The connect context ('<') is a node path (initially the same as the attach context), set by the CONNECT command, and conventionally indicates a user node or subdirectory.

A node list consists of a sequence of node names (consecutive levels in the tree) separated by right angle brackets. A password may be necessary in acquiring access privileges at a particular node, in which case the node name is followed by a colon and the password. Sets of nodes can be referenced -- all nodes at a particular level are indicated by '*', and all inferior nodes are designated by '**' (which can occur only at the end of the node list).

For example,

```
<<<CCA>HACKER:>WALDO
```

Starting at the top context, the node path references the subdirectory WALDO under user HACKER at site CCA (with a password supplied to gain access to HACKER).

```
<<WALDO>**
```

Starting at the attach context, the node path references the subdirectory WALDO and all inferior nodes (note that WALDO is included -- the REMOVE command, for example, would delete the node WALDO as well as its inferiors).

Referencing Files

User file names have the same form as TENEX file names: a file designation, an optional extension, and an optional version number. The file designation is separated from the extension by a period, and the extension from the version number by a semicolon. File sets may be indicated by an asterisk in any or all of the file designation, extension, and version number fields.

Version numbers allow unambiguous reference of files with the same file designation and extension. Each file has a version number assigned to it by DFTP (which is unrelated to its TENEX version number) -- version numbers cannot be set by the user. Later versions of a file with the same file designation and extension receive higher version numbers. A version number may be explicitly supplied in referencing an existing file, otherwise a default is provided.

All commands that accept as input a file name will also accept a file path, which consists of a node path followed by a file name, with the two parts separated by a right angle bracket (unless the node path is only a context, in which case the right angle bracket is omitted). If a node path is given, the file name is used in the <FILES> node under the node referenced. If a node path is not given, the file name is used in the <FILES> node under the connect context -- the default context for a file reference is the connect context.

For example:

```
MAIL.TXT
```

The file name references the file MAIL.TXT in the <FILES> node under the connect context.

```
<*.SAV;*
```

The file name references all versions of all files with the extension SAV in the <FILES> node under the connect context.

```
<<MACROS>COMMON>SYSMAC.MAC
```

Starting at the attached context (presumably a user node), the file path references the file SYSMAC.MAC in the <FILES> node under the COMMON subdirectory of the MACROS subdirectory.

Command Summary

DFTP command and argument input is similar to TENEX, with command recognition and TENEX editing controls. In particular,

<control-A> deletes a character,

<control-R> retypes the line,

<control-X> and <rubout> delete the line,

<escape> and <space> are separators, and

<carriage return>, <line feed> and <eol> are terminators.

The DFTP commands and their arguments are:

ATTACH <node path>

CONNECT <node path>

DELETE <file path>

DIRECTORY <file path>

TERSE

VERBOSE

EXPUNGE <node path>

GET <file path> [local synonym]
RETRIEVE is exactly equivalent to GET.

LIST <node path>

NO-DATALANGUAGE

PUT <file path> [remote synonym]
STORE is exactly equivalent to PUT.

QUIT

REMOVE <node path>

SHOW-DATALANGUAGE

TIME-TRANSFERS

UNDELETE <file path>

UNTIME-TRANSFERS

Items in angle brackets are required arguments; items in square brackets are optional ones.

The connect context is the default context for all commands except ATTACH and CONNECT, which have as their respective defaults the top context and the attach context.

Many commands have default arguments and trailers which are invoked by giving a space or escape as the argument or argument terminator. The default argument is '<<' for the CONNECT command, '<' for EXPUNGE, and '**' for LIST and REMOVE, which is also the default trailer. For DIRECTORY, GET, PUT, DELETE, and UNDELETE, the default argument and trailer (after a '>') is '*.*;*'.

Node Oriented Commands

The ATTACH command sets the attach context and initiates Datacomputer accounting functions.

The CONNECT command sets the connect context (and creates new subdirectories).

The EXPUNGE command removes files marked as deleted from the <FILES> node under the node given as the command argument. If the <FILES> node contains no files (deleted or undeleted) it is deleted from the Datacomputer directory.

The LIST command lists Datacomputer nodes (sites, users, subdirectories, and <FILES>) and information about them. The information displayed by the VERBOSE option comes directly from the Datacomputer.

The REMOVE command removes nodes from the Datacomputer directory; they must either have no inferior nodes, or be part of a node set specified using "**". In the latter case, data stored under the nodes will also be deleted.

File Oriented Commands

File deletion operates as with TENEX. The DELETE command marks files as deleted, but does not eliminate them. They can be listed via the VERBOSE option of the DIRECTORY command, and their deleted status can be changed by the UNDELETE command. The removal of deleted files is deferred until an EXPUNGE is performed on the <FILES> node containing them. The default version number is the lowest undeleted, unless a file set is indicated, in which case all versions in the set are deleted.

The DIRECTORY command lists files and information about them. The VERBOSE option lists deleted and undeleted files (with deleted ones indicated by a D after the name), the date and time created (for TOPS-10 sites), the date and time last written (for TENEX sites), the date and time stored, and the size. For files stored from TENEX sites the size information is in the form <number of bytes>(<byte size>). For files stored from TOPS-10 sites the information is in the form <number of 36 bit words>(-<data mode)). The TERSE option lists undeleted files and their sizes (as with the VERBOSE option). The default version number is the highest undeleted, unless a file set is indicated, in which case information for all versions in the set is listed.

The GET and PUT commands retrieve and store local disk files on the Datacomputer. Files of any type (text or binary image, for example) can be stored. If a synonym is not supplied, the Datacomputer file name is used as the local file name. If the first argument to either command is completed with an escape or a space, the synonym option is invoked and the commands then operate in the form

```
        GET Datacomputer file [AS] local file, and
        PUT local file [AS] Datacomputer file.
```

For the GET command the default version number is the highest undeleted, unless a file set is indicated, in which case all versions of the set are retrieved. The PUT command sets the version number of the file being stored to be one greater than the highest version of existing files with the same file designation and extension (note that a file set indicated in any file name field is treated as if all existing files had the same field -- storing *.* results in the stored files receiving version numbers one greater than the highest version number found in any existing file).

The UNDELETE command rescinds a file's deleted status. The default version number is the highest deleted, unless a file set is indicated, in which case all versions in the set are undeleted.

Miscellaneous Commands

ENABLE causes DFTP to recognize an expanded set of commands, including the privileged commands discussed below. Its action is marked by a change in the prompt character, from "*" to"!".

The SHOW-DATALANGUAGE and NO-DATALANGUAGE commands respectively allow and inhibit the output to the user's terminal of the messages sent to and received from the Datacomputer.

Data transfer rates are calculated and given to the user when the TIME-TRANSFERS command has been invoked. The calculations are avoided with the UNTIME-TRANSFERS command.

QUIT exits gracefully from DFTP, closing network connections.

Site Dependent Features

For the TOPS-10 version, the LOCAL-DIRECTORY command lists the user's local file directory.

For the TENEX version, the EXEC command provides the user with an inferior exec, which is flushed when the user returns to DFTP. Where a local file name is possible (in the GET and PUT commands) an initial space or escape invokes TENEX name recognition, indicated by a right angle bracket prompt. A control-O can be used to halt the output from the LIST and DIRECTORY commands.

Responses

There are three types of messages that DFTP gives the user. Comments surrounded by square brackets are primarily informational messages, and are never errors. Parentheses enclose non-fatal errors and informational Datacomputer messages, such as '(LEBAR2: ERROR: NO SUCH FILE)', resulting from an attempt to DELETE a nonexistent file, and '(SXPF9: STAGING DATA FOR FILE = DFTP.CCA.DFTP.<FILES>)', indicating that data is being moved from tertiary mass memory to secondary buffer memory.

These messages come directly from the Datacomputer, indicated by the name and colon at the head of the message. Fatal error messages are surrounded by question marks, and of course never occur.

Access Control

Access control in DFTP uses a subset of the full Datacomputer facilities. The full discussion of Datacomputer privilege facilities is in the current Datacomputer User manual; however, the following summary should suffice for most DFTP users. Access privileges are specified in "privilege blocks" attached to nodes in the directory. A node may have any number of privilege blocks attached to it; each specifies a particular set of access privileges, and a class of users to whom that set applies. DFTP provides two classes of access:

- CONTROL allows users to create and allocate nodes under the node at which it is granted, change privileges at that node and below, and read, write, and delete data stored below that node.

- READ allows users to attach to a node and read data stored under it, but not to perform any of the other functions granted by CONTROL.

All other users are prevented from any access to the data. Users may be may be identified in DFTP by their network identity (defined by the host and socket from which they access the Datacomputer), and by passwords. (Some systems enforce assignment of socket numbers according to the user's identity on that system, thus providing a convenient automatic identification to DFTP.)

When a user attempts to ATTACH to a node, or to read or write data stored under it, the Datacomputer checks to see if the user is in any of the user-sets identified in privilege blocks on that node, and if so, assigns the corresponding class of privileges. If no set matches the user, then no privileges are allowed. The scan is done in the order of creation of privilege blocks, and if a user matches more than one, the first one takes effect. Access controls are set by the CHANGE and CREATE commands, described in the following section.

Privileged Commands

Certain administrative functions are performed by a set of restricted commands, known as "privileged commands." These are not normally available to the user; they are recognized by DFTP only after execution of the ENABLE command.

- The ALLOCATE command is used to set the maximum number of megabits a user may consume (it can also be used to set subdirectory limits). Allocations are made and reported in Megabits (actually 1,ØØ13,76Ø bits), which is 55 512-word pages, or 22Ø 128-word blocks.
- The CHANGE command resets the access control information for a given node. It first deletes all existing access control specifications (privilege blocks); then it builds new privilege blocks interactively with the user. The PROTECTION subcommand of the LIST command can be used to examine the privilege blocks of nodes.
- The CREATE command is used to create a node for a new user; after the node has been created, it falls into the same access control specification as the CHANGE command.
- The DISABLE command returns the user from Privileged Command mode to normal use; this is signalled by a return to an asterisk as a prompt character.

- The LINK command allows the user to send Datalanguage to the Datacomputer directly. In this mode, prompted by a left angle bracket, each line typed by the user is sent directly to the Datacomputer. It is terminated by submission of an empty line, whereupon DFTP returns to ENABLE mode, as signalled by an exclamation-point prompt.

DFTP Command Summary
Paths

```
<node path> ::=   <context>
                | <node list>
                | **
                | <context> **
                | <context> <node list>
                | <context> <node list> > **
                | <node list> > **
<context> ::=  <               (connect context)
                | <            (attach context)
                | <<<          (top context)
<node list> ::=  <node>
                | <node> > <node list>
<node> ::=       <name>
                | <name> : <password>
                | *

<file path> ::=  <file name>
                | <context> <file name>
                | <node list> > <file name>
                | <context> <node list> > <file name>
<file name> ::=  <file>
                | <file> .
                | <file> . <extension>
                | <file> ; <version>
                | <file> . ; <version>
                | <file> . <extension> ; <version>
<file> ::=       <name> | *
<extension> ::=  <name> | *
<version> ::=    <number> | *
```

Notes:
    Underscored angle brackets should be included literally.
    Any printing ASCII characters except <, >, ., :, *, ?, ', and " may be used in a <name>.
    Any printing ASCII characters (plus space) except >, ., ', and " may be used in a <password>.

Commands

```
ATTACH <node path>

CONNECT <node path> (1)

DELETE <file path> (4) (5)

DIRECTORY <file path> (4) (6)
        TERSE
        VERBOSE

ENABLE

EXPUNGE <node path> (2)

GET (RETRIEVE) <file path> [local file name synonym] (4) (6)

LIST <node path> (3)
        TERSE
        VERBOSE

NO-DATALANGUAGE

PUT (STORE) <file path> [remote file path synonym] (4)

QUIT

REMOVE <node path> (3)

SHOW-DATALANGUAGE

TIME-TRANSFERS

UNDELETE <file path> (4) (7)

UNTIME-TRANSFERS
```

Notes:

Required input is indicated by angle brackets.
Optional input is indicated by square brackets.
The connect context is the default context for all commands except ATTACH and CONNECT, which have as their respective defaults the top context and the attach context.

(1) The default argument is <<.
(2) The default argument is <.
(3) The default argument (and trailer) is **.
(4) The default argument (and trailer) is *.*;*.
(5) The default version is the lowest undeleted.
(6) The default version is the highest undeleted.

(7) The default version is the highest deleted.

Privileged Commands and Their Subarguments

```
ALLOCATE <node path>
    Megabits: [decimal integer]

CHANGE <node path>
    Add a new privilege? [Y(es)] or [N(o)]
        Allow write? [Y(es)] or [N(o)]
        Restrict via network? [Y(es)] or [N(o)]
            Restrict via local host? [Y(es)] or [N(o)]
                Host number (octal): [octal integer]
                  (if host not local)
            Restrict via user? [Y(es)] or [N(o)]
                Socket number (octal): [octal integer]
                  (if user restricted and host not local)
                User: [user name]
                  (if user restricted and host local)
                  (a directory name if TENEX)
                  (a programmer number if TOPS-10)
        Restrict via password? [Y(es) or N(o)]
            Password: [string]

CREATE <node path>
    (see CHANGE)

LIST
    PROTECTION

LINK
    (A null input line returns the user to command mode.)
```

Examples Using Privileged Commands

```
;Attach to a node.
; (gain control at site CCA by supplying the proper password)
*ATTACH <<<CCA:

*ENABLE

;Create a user and privileges.
; (the first privilege allows creation and deletion for
;   local user HACKER upon supplying the password "ETAOIN")
; (the second privilege allows creation and deletion for
; anyone from Harvard (host 11) upon supplying the password "SHRDLU")
; (the third privilege allows anyone read)
!CREATE HACKER
 [OK]
 Add a new privilege? Yes
  Allow control? Yes
  Restrict via network? Yes
   Restrict via local host? Yes
   Restrict via user? Yes
    User: HACKER
  Restrict via password? Yes
   Password: ETAOIN
  [OK]
 Add a new privilege? Yes
  Allow control? Yes
  Restrict via network? Yes
   Restrict via local host? No
    Site: 11
   Restrict via user? No
  Restrict via password? Yes
   Password: SHRDLU
  [OK]
 Add a new privilege? Yes
  Allow control? No
  Restrict via network? No
  Restrict via password? No
  [OK]
 Add a new privilege? No

;List the privileges.
; (in Datacomputer format (passwords are never listed))
!LIST HACKER
!!PROTECTION
    CCA
        HACKER
            ] (1),U=**,H=31,S=12582928,G=CLWRA
            ] (2),U=**,H=9,S=ANY,G=CLWRA
            ] (3),U=**,H=ANY,S=ANY,G=LR

;Replace the privileges.
; (the first privilege allows creation and deletion for
;   local user HACKER)
; (the second privilege allows anyone read
;   upon supplying the password "WALDO")
; (the "[OK]" indicates that the previous privileges have been delete)
!CHANGE HACKER
 [OK]
 Add a new privilege? Yes
  Allow control? Yes
  Restrict via network? Yes
   Restrict via local host? Yes
   Restrict via user? Yes
    User: HACKER
  Restrict via password? No
  [OK]
 Add a new privilege? Yes
  Allow control? No
  Restrict via network? No
  Restrict via password? Yes
   Password: WALDO
  [OK]
 Add a new privilege? No

;List the privileges.
!LIST HACKER
!!PROTECTION
    CCA
        HACKER
            ] (1),U=**,H=31,S=12582928,G=CLWRA
            ] (2),U=**,H=ANY,S=ANY,G=LR

;List all information.
; ("MX-U" indicates the maximum allocation in megabits)
!LIST HACKER
!!VERBOSE
    CCA
        HACKER
            ] MX-U=10.00 CHRG=0.00
            ] IN-N=0 IN-F=0
            ] CREA=761101052805

;Change the allocation.
; (decrease the allocation from 10 megabits to 2 megabits)
!ALLOCATE HACKER
 [Megabits:2]


;List all information.
!LIST HACKER
!!VERBOSE
    CCA
        HACKER
            ] MX-U=2.00 CHRG=0.00
            ] IN-N=0 IN-F=0
            ] CREA=761101052805
```

ECAP

ECAP is an Electronic Circuit Analysis Program.

This is an integrated system of programs which can be used for design and analysis of electronic circuits. The system of programs can produce DC, AC, and/or transient analyses of electrical networks from a description of the connections of the network (the circuit topology), a list of corresponding circuit element values, a selection of the type of analysis desired, a description of the circuit excitation, and a list of the output desired.

The user requires neither a knowledge of the internal construction of the system of programs nor computer programming techniques to use ECAP effectively.

This subsystem was originally distributed as DECUS No. 10-34. It is documented in The IBM 1620 Electronic Circuit Analysis User´s Manual, #GH20-0170-2.

ECAP accepts input from the file DSK:INPUT.DAT and writes its output on DSK:OUTPUT.DAT.

It is started by typing ECAP to the EXEC.

FILCOM

FILCOM compares two files in either ASCII mode or binary depending upon switches of file name extensions. All standard binary extensions are recognized as binary by default.

Switches are:

/A	compare in ASCII mode
/B	allow compare of Blank lines
/C	ignore Comments and spacing
/S	ignore Spacing
/H	type this Help text
/#L	Lower limit for partial compare or number of Lines to be matched (# represents an octal number)
/#U	Upper limit for partial compare
/Q	quick compare only, give error message if files differ
/U	compare in ASCII Update mode
/W	compare in Word mode but don´t expand files
/X	expand files before word mode compare

FASBOL II

FASBOL II is a SNOBOL Compiler for the PDP-1Ø written by Paul Joseph Santos, Jr.

ABSTRACT

The FASBOL II compiler system represents a new approach to the processing and execution of programs written in the SNOBOL4 language. In contrast to the existing interpretive and semi-interpretive sytems, the FASBOL compiler produces independent, assembly-language programs. These programs, when assembled, and using a small run-time library, execute much faster than under other SNOBOL4 systems.

While being almost totally compatible with SNOBOL4, Version 3, FASBOL offers the same advantages as other compiler systems, such as:

1. Up to two orders of magnitude decrease in execution times over interpretive processing for most problems.

2. Much smaller storage requirements at execution time than in-core systems, permitting either small partitions or large programs.

3. Capability of independent compilation of different program segments, simplifying program structure and debugging.

4. Capability of interfacing with FORTRAN and MACRO programs, providing any division of labor required by the nature of a problem.

5. Measurement and runtime parameter facilities to aid in optimizing execution time and/or storage utilization.

CHAPTER 1

1. Introduction

The first FASBOL [1] was a similar system designed and written for the UNIVAC 11Ø8 under the EXEC II operating system, and operational as of October 1971. FASBOL II, the PDP-1Ø system , is an enhanced version which is in addition compatible with Version 3 of SNOBOL4. It is presumed that the reader is familiar with SNOBOL4, Version 3, as described by the second edition (1971) of the Prentice-Hall publication [2]. Using [2] as a base description of SNOBOL4, the following chapters explain any differences and additions present in FASBOL II, as well as describe how to use it to compile and run programs.

The FASBOL II compiler is itself written in FASBOL and is like FORTRAN and MACRO in that it accepts specifications for source, listing, and object files (the object is a MACRO program which must be assembled). The reason for writing the compiler in FASBOL was for speed of implementation, automatic checkout of the run-time library, and ease of modification. If after some use the compiler should prove to be unsatisfactory in terms of core utilization or execution speed, the MACRO stage can be hand-tailored, using the measurement techniques available in FASBOL, into a more efficient program. A further enhancement would be the direct production of relocatable code by a one-pass compiler written in either FASBOL or MACRO.

The FASBOL II run-time library is written in MACRO, since its efficiency is paramount, and is searched in library mode, after loading all FASBOL programs, in order to satisfy program references to predefined primitives and system routines. Sections of the FORTRAN library may also be loaded, provided they do not compete with FASBOL for UUO's and traps.

Internal documentation of the operation of the run-time system is available as a separate document.

Use of the male gender in third-person references in this manual in no way implies that FASBOL is not useful for female persons; the author is simply not aware of any easy way to write in neuter.

CHAPTER 2

2. Language Description

The syntax for FASBOL II is given in Appendix 1. In addition to the detailed changes mentioned below, this syntax differs from that given in [2] only in that it is more restrictive of compile-time syntax. For example, since FASBOL II does not permit redefinition of operators, the expression

```
( A . B ) + ( C . D )
```

is flagged as a compilation syntax error, whereas the interpreter (i.e. the system described in [2]) would accept it and then produce an "illegal type" error message during execution. Most SNOBOL4 programs should run "as is" under FASBOL II; Sections 2.1.1 and 2.1.2 describe exactly all features that may cause incompatibility, and the remaining sections deal with enhancements.

2.1 General Language features

The following three sections discuss, respectively, features of SNOBOL4 not implemented in FASBOL II, features of SNOBOL4 implemented differently in FASBOL II, and additional features available in FASBOL II and described more completely in sections 2.2, 2.3, 2.4, and 2.5.

2.1.1 SNOBOL4 features not implemented

The predefining primitives EVAL and CODE, the datatypes CODE and EXPRESSION, and direct gotos (as used with CODE) are not implemented. They imply a run-time compilation capability which is not available in the FASBOL library at this time.

The redefinition of operators via OPSYN, or the redefinition of predefined primitive pattern variables (e.g. ARB) or functions (e.g. SPAN) is not permitted in FASBOL, which considers all these items as a structural part of the language essential to generating efficient code. For this reason, the keywords &ARB, &BAL, &FAIL, &FENCE, &REM and &SUCCEED are not needed and therefore not implemented. Also, &CODE has no meaning for the PDP-1Ø, and is not available either.

The SNOBOL4 tracing capability is not implemented in FASBOL at this time. However, the &STNTRACE keyword (see section 2.5) provides some tracing capability.

Although QUICKSCAN mode for pattern matching is implemented in FASBOL, two features of this mode, available in SNOBOL4, are not implemented. They are a)continual comparison of the number of characters remaining in the subject string against the number of characters required by non-string-valued patterns, and b) assumption that unevaluated expressions must match at least one character. This implies that some matches may last a little longer and perhaps have a few more side-effects (e.g. via $), and that left-recursive pattern definitions will loop indefinitely (see Section 3.2.3).

2.1.2 SNOBOL4 features implemented differently

The FASBOL I/O structure is time-sharing oriented and does not use FORTRAN I/O, so that it differs somewhat from the SNOBOL4 I/O. Both input and output can be either line or character mode. Line mode is similar to SNOBOL4 I/O, with input records being terminated just prior to a carriage return, line feed sequence, and with this sequence being added to output records. Trailing blanks seldom occur on input, and so the &TRIM keyword is not implemented (but the TRIM function is). Character mode gets one character (including a carriage return or linefeed) on input, and outputs a string without appending a carriage return, linefeed sequence to it. INPUT, INPUTC, OUTPUT and OUTPUTC have predefined associations (see 2.4.2, I/O primitives) corresponding to line and character mode teletype input and output, respectively. PUNCH does not have a predefined association. There are additional predefined primitives for device and file selection, etc., discussed in Section 2.4.2.

Changes in program syntax are as follows:

a) Compiler generated statement numbers are always on the left.

b) Source lines (not statements) are truncated after 132 characters.

c) The character codes and extended syntax are like the S/360 version, except the character ! (exclamation point) replaces | (vertical stroke) and \ (back slash) replaces \ (not sign).

d) Binary $ and . (immediate and conditional pattern assignment) have lower precedence than the binary arithmetic operators, but higher precedence than concatenation. Thus, the expression

```
        X   A + B $ C
```

is taken to mean

```
        X   ((A + B) $ C)
```

In SNOBOL4, $ and . have the highest precedence of all binary operators, and would give the meaning

```
        X   (A + (B $ C))
```

to the above expression.

e) The number of arguments in a function call, formal arguments in a function definition, and fields in a datatype definition are limited to a maximum of 15.

f) The object of the unary * (unevaluated expression) operator cannot be an explicit pattern structure (e. g. use LEN(*n) instead of *LEN(n) and (*pat1 ! *pat2) instead of *(pat1 ! pat2).

Changes in program semantics and operation are as follows:

a) The binary . (name) operator always returns a value of type NAME (SNOBOL4 sometimes returns a STRING). Names of TABLE entries are permitted.

b) Some predefined primitive functions operate differently than in SNOBOL4 (see Section 2.4.2).

c) &MAXLNGTH is initially set to 262143 (in SNOBOL4, the value is 5000). This value is also the absolute upper limit on string size.

d) If &ABEND is nonzero at program termination, an abnormal (EXIT 1,) exit to the system is taken.

e) Primitive functions may be called with either too few or too many arguments, even via OPSYN or APPLY.

2.1.3 Additions to SNOBOL4

Declarations are provided in FASBOL for the enhancement of programs. No declarations are ever required, but if they are used they must all precede the first executable statement in a program. Declarations are described in Section 2.2.

Additional control cards and compilation features are available, discussed in Section 2.3, and additional predefined primitive functions and keywords are described respectively in Section 2.4 and 2.5.

Additions to program syntax are as follows:

a) Quoted strings may be continued onto a new line, with the continuation character removed from the literal.

b) Single and double quotes may be included in a literal that is bracketed by the same, by use of the construction '' to stand for ' and "" to stand for " inside of literals bracketed by ' and ", respectively.

c) Comment and control lines (i.e. starting with * or -) may start inside a line image (i.e. after a ;), and consume the remainder of the line image.

d) The run-time syntax for DEFINE and DATA prototypes has been loosened to conform with the rest of FASBOL syntax by permitting blanks and tabs after ((open parenthesis), around , (comma), and before) (close parenthesis).

2.2 Declarations

FASBOL declarations have two primary purposes. One purpose is to optimize a program in space and/or time. The second purpose is to allow inter-program linkage and communication. The general form of a declaration is a call on the pseudofunction DECLARE, with two or three arguments, the first of which is always a string literal identifying the type of declaration, and the remaining arguments specifying the parameters or program symbols upon which the declaration has effect. As has been noted, all declarations must precede the first executable statement; this also implies no declaration line may contain a label. A FASBOL program with declarations can be made otherwise compatible with a SNOBOL4 interpreter by inserting the statement

```
          DEFINE('DECLARE()', 'RETURN')
```

at the beginning of the program.

2.2.1 PURGE and UNPURGE

Normally, a FASBOL application will involve a main program and several independently compiled subroutines. During execution, the run-time system maintains a run-time symbol table for each separately compiled program, as well as a global symbol table. In the absence of declarations to the contrary, all explicitly mentioned variables, labels, and functions are put into the local symbol table for that program. Thus, program X and program Y may both have labels LAB to which they perform indirect gotos. The global symbol table contains such global symbols as OUTPUT and RETURN, and any new symbols that arise during execution of any of the programs. A symbol lookup in program X first searches the local symbol table for program X, then the global symbol table, and then, if still not found, creates a new entry in the global symbol table. Thus a local symbol table never grows beyond the size determined for it at compilation time.

The purpose of the PURGE.VARIABLE, PURGE.LABEL, and PURGE.FUNCTION declarations is to eliminate symbols from the local symbol table and thus conserve space. This can be safely done for labels provided that the label is never referenced indirectly ($ goto), or explicitly and/or implicitly in a DEFINE call. A similar criterion applies to safely eliminating variables, only the number of cases to watch for is greater; any situation that requires an association between the string representing the variable name, and the actual location assigned to that variable, is such a case. For example, the statement

```
          INPUT('VARB',0,60)
```

implies that the variable VARB, if it is mentioned explicitly in the program and thus assigned a location, must be in the run-time symbol table. An explicit reference to VARB would be, for example,

```
          TTYLIN  =  VARB
```

On the other hand, a variable that is never referenced explicitly need not be in the local symbol table, but the first symbol lookup for it will create an entry for it in the global symbol table. In the case of functions, the only symbols that can be safely purged are the ones corresponding to predefined primitives, since all others are needed to be able to define the user functions, via DEFINE or otherwise.

When there appear to be more symbols of a given type to be purged than left in the symbol table, the second argument to the declaration can be the pseudovariable ALL; then, the UNPURGE.VARIABLE, UNPURGE.LABEL or UNPURGE.FUNCTION declarations can be used to place specific symbols into the symbol table.

2.2.2 GLOBAL. ENTRY. and EXTERNAL

These declarations permit interprogram communication on an indirect (i.e. symbol lookup) and/or direct (i.e. loader linking) basis. The GLOBAL.LABEL, GLOBAL.VARIABLE, and GLOBAL.FUNCTION declarations override PURGE/UNPURGE and cause the specified symbols to be placed in the global symbol table instead of the local one. Only one subprogram may globalize a particular symbol, since the implication is that the variable, label, or function belongs to that program. Any other program that does not have a similar symbol in its local symbol table will then be able to reference the global symbol.

While GLOBAL provides for interprogram communication via the symbol table, the ENTRY/EXTERNAL declarations provide for more direct interprogram communication by using the linking loader to connect external references. The ENTRY.VARIABLE, ENTRY.LABEL, and ENTRY.FUNCTION declarations make the specified local entities accessible to external programs. The second and third arguments to the ENTRY.FUNCTION declaration are like the arguments to DEFINE, and the function is automatically DEFINED the first time it is called, so no extra DEFINE is necessary. The ENTRY.FORTRAN.FUNCTION declaration is similar to ENTRY.FUNCTION except that the compiler assumes the entry will be called by a FORTRAN program. Any combination of FASBOL, FORTRAN, and MACRO programs is permitted, provided the main program is FASBOL, and certain restrictions on FORTRAN code (see Section 3.2.5) are observed.

The EXTERNAL.VARIABLE, EXTERNAL.LABEL, EXTERNAL.FUNCTION, and EXTERNAL.FORTRAN.FUNCTION declarations are the converse of the ENTRY

literal) to FORTRAN is to use a variable declared to are STRING, which is the only real use for that declaration; a fixed amount of storage is allocated for the variable based on the max character count given in parenthesis after each name in the parameter list (second argument). The only restrictions on these variables, referred to here as <u>dedicated mode variables</u>, is that they may not have I/O associations. All keywords (except for &RTNTYPE and &ALPHABET) are treated as dedicated integers.

2.2.4 Other declarations

The OPTION declaration serves to specify various compilation options. The HASHSIZE=n declaration, ignored in all but the main program, is used to cause a larger or smaller than normal hash bucket table to be allocated for use by the run-time symbol table. The number n should be a prime and represents the number of buckets in a linked hash table; the standard value is 127. This bucket table is at the center of all symbol lookups in the runtime system, including TABLE references, so that there is a distinct tradeoff between the sparsity of the table and the time required for a lookup. The NO.STNO option causes the compiler to eliminate the normal bookkeeping on &STNO, &STCOUNT, etc. that occurs each time a statement is entered, and is helpful to speed up slightly the execution of debugged programs. The TIMER option, which is incompatible with NO.STNO in the same program, adds to the normal bookkeeping a valuable statement timing feature (see Section 3.2.4). The timing statistics on each program being timed are printed out at the end of execution, and intermediate timing statistics can be printed out during execution by using the primitive EXTIME (see Section 2.4.2).

The SNOBOL.MAIN and SNOBOL.SUBPROGRAM declarations indicate whether the program is a main program or a subprogram, and give it a name (i.e. TITLE in MACRO). In the absence of either declaration,

```
          DECLARE('SNOBOL.MAIN','.MAIN.')
```

is assumed.

The RENAME declaration is used primarily to rename predefined symbols (see Appendix 2) that would otherwise conflict with a given user's. For example, if a user wished to have a variable called ARB, or his own IDENT function while retaining the primitive also, he should rename them some other names. On the other hand, if a user wants to re-define IDENT, for example, no RENAME should be used, and IDENT will become redefined when his own DEFINE is executed.

Although usually the order in which declarations occur is not important, all ('PURGE.x',ALL) declarations should precede others which also refer to entities desired to be purged. For example, the sequence

```
         DECLARE('ENTRY.VARIABLE','A,B,C')

         DECLARE('PURGE.VARIABLE',ALL)
```

will cause the variables A, B and C to be missed and included in the symbol table, since the purge flag only has effect on new symbols.

It should also be noted that whereas the syntax of variable and function name lists uses a comma as a separator, label lists are separated by blanks. The reason for this is that the synatx for labels includes commas, but a blank is a valid label terminator. Also, all quoted strings in declarations are delimited by single quotes. A single quote may be entered inside such a string (for example, in a label) by using the '' convention mentioned in Section 2.1.3.

2.3 Control

In FASBOL there is an expanded repertoire of control cards for controlling listing, cross-referencing, and failure protection. In the following list, the first of a pair controlling a switch is the initial mode.

LIST, UNLIST	turns program listing on,off.
NOCODE, CODE	turns object listing off, on (the generation of object code can be inhibited by not specifying an object output file).
EJECT	causes a page eject (form feed).
SPACE n	spaces n lines (or 1 line if n is absent).
NOCROSS, CROSREF	turns symbol cross-referencing off, on. This can be done for a whole program or selectively for parts of it.
FAIL, NOFAIL* (1)	turns off a compiler feature that traps unexpected statement failures. When the feature is on (NOFAIL), any statement within its scope that does not have a conditional GOTO, and which fails, will cause an error exit. An unconditional GOTO is equivalent to none at all, and will be trapped if the statement fails.

2.4 Predefined Primitives

In the following sections, only those primitives which differ from SNOBOL4 or are new in FASBOL will be discussed. Appendix 2 has a complete list of primitives available in FASBOL.

(1)
*Credit for this idea goes to the authors of SPITBOL [3], who also inspired the inclusion of DUPL, LPAD, RPAD, and REVERS.

2.4.1 Pattern Primitives

Three new primitives in the SPAN/BREAK class have been added; these are structural, like the other pattern primitives, and cannot be redefined.

NSPAN(class) is like SPAN, but may match the null string.

BREAKQ(class) is like BREAK, but does not look for break characters inside of substrings delimited by single (') or double (") quotes.

BREAKX(class) is like BREAK, but has alternatives that extend the match up to each succeeding break character. Operates like

```
BREAK(class)   ARBNO(LEN(1)  BREAK(class)).
```

2.4.2 Expression Primitives

A number of SNOBOL4 primitives work somewhat differently in FASBOL, and new primitives have been added for I/O, string manipulation, and communication with the run-time system.

COLLECT(n) forces a garbage collection, returns the total number of words collected, and fails if no block of size n or larger is available.

CONVERT(table,'ARRAY') and CONVERT(array,'TABLE') are implemented differently, by removing the above facilities from CONVERT and putting them in ARRAY and TABLE. See below.

ARRAY(table) converts a TABLE datatype to an ARRAY as described in [2], pp.122. An empty TABLE causes ARRAY to fail.

TABLE(array) converts certain types of ARRAY datatypes to a TABLE as described in [2], pg. 122. The TABLE datatype is different from all others in that, once it has been created, it exists independently from its use in the program. Thus, to reclaim the storage, it must be explicitly deleted by TABLE(table). Once a table has been deleted, further references to it are illegal.

APPLY(fun,args) will accept either more or fewer arguments than required by the function; it will reject extra ones or fill in missing arguments with null values.

OPEN(device,chan) opens an I/O device on a software channel, assigns buffers and returns the channel number.

If chan < Ø or > 15, illegal I/O unit error.
If chan = Ø, an unused channel is assigned; if channel table is full (> 15 channels), error.
If chan is already in use, illegal I/O unit error.
If device is not a string of the form:
devnam [([outbuf] [,[inbuf]])]
it is a bad prototype or illegal arg error.
If devnam is not recognized, or is not a file structure and is already assigned to a channel, illegal I/O unit.
If ([outbuf] [,[inbuf]]) is missing, (2,2) is assumed.
If either outbuf and/or inbuf is missing, Ø is assumed for the missing value.
If the device allows only input (or output), the other buffer parameter is ignored.

Examples:

```
OPEN('DTA3(,4)',5)
OUTPUT('DUMP',OPEN('MTAØ'),1ØØØ)
```

RELEASE(chan) releases the software channel and all associations to it, returns all buffers to free storage, and returns a null value.

If chan < Ø or > 15, illegal I/O unit error.
If chan = Ø, release all channels in use.
If channel not in use, ignore and return.

LOOKUP(file,chan) opens file for input (reading) on software channel, returns channel. Fails if file is not found.

ENTER(file,chan) opens file for output (writing) on software channel, returns channel. Illegal I/O error if file is not found.

If chan < Ø or > 15, illegal I/O unit error.
If chan = Ø, a preliminary OPEN('DSK') is performed, the new channel returned.
If file is not of the form
filnam [. ext] [[proj , prog]]
, a bad prototype error.
If channel is not open for operation, illegal I/O unit error.
If input (LOOKUP) or output (ENTER) side of channel already selects a file, the old file is closed.

CLOSE(chan,inhib,outhib) closes the input and/or output side of the software channel, returns null.

If outhib is non-null, the output side is not closed.
If inhib is non-null, the input side is not closed.
If chan < 1 or > 15, illegal I/O unit error.
If channel is not in use, ignore and return.

INPUT(var,chan,len)
OUTPUT(var,chan,len) create an input (output) association between the variable var and software channel chan, with line/character mode and association length specified by len, and return null.

If len > Ø, line mode.
If len = Ø, line mode with default length (72).
If len < Ø or not integer, character mode.
If chan > 15, illegal I/O unit.
If chan > Ø use channel table to determine I/O device.
If chan = Ø use TTY I/O.
If chan < Ø or not INTEGER, disconnect association but do not DETACH it.
If the input (output) side of the channel has not been opened, illegal I/O unit error.
If var is not a string, illegal arg.
If an association for var already exists, it is changed.
If variable is dedicated, illegal arg.

Examples:

```
INPUT('SOURCE',LOOKUP('SRCELT.SNO'),8Ø)

OUTPUT('TYPEOUT.CHARS',Ø,-1)
```

The initial I/O configuration is equivalent to:

```
INPUT('INPUT')

INPUT('INPUTC',Ø,-1)

OUTPUT('OUTPUT')

OUTPUT('OUTPUTC',Ø,-1)
```

During execution, all system messages are output via the variables OUTPUT and OUTPUTC (which should always both be associated to the same channel). In order to switch system output to the printer, for example,

```
LPT = OPEN('LPT')

OUTPUT('OUTPUT',LPT,132)

OUTPUT('OUTPUTC',LPT,-1)
```

Channel Ø is never assigned, but when used in INPUT and OUTPUT associations implies the user TTY and TTCALL operation. On input, line mode reads up to (but not including) the next carriage return, line feed (CR,LF) sequence, and then these are discarded. Character mode reads only one character (including CR or LF). Line mode discards any characters beyond the association length. An EOF causes failure in either mode, but cannot occur on some devices (such as the user TTY).

On output, line mode writes out the string value with a CR, LF appended, whereas character mode does not append the CR, LF. In line mode, if the string length is greater than the association length, extra CR, LF characters are inserted every association length substring.

DETACH(var) disconnects input and output associations for the variable and detaches it from I/O processing, returns null. If the variable had no association, ignore and return.

SUBSTR(string,len,pos) returns the substring of string starting at pos of length len, and fails if len < Ø, pos < Ø, or pos + len > SIZE(string). The position convention is the same as that for patterns, and the operation is similar (but faster and less space-consuming) to:

```
string   TAB(pos)  LEN(len) . SUBSTR
```

INSERT(substring,string,len,pos) returns the new string formed by substituting substring for the one specified by the last 3 arguments into string, failing under the same conditions as:

```
string   TAB(pos) . PART1  LEN(len)  REM . PART2

INSERT = PART1 substring PART2
```

LPAD(string,len,padchr) returns the string formed by padding string on the left with padchr characters to a length of len. If string is already too long, it is returned unchanged; if padchr has more than one character, only the first is used. If the third argument is null, blanks are used.

RPAD(string,len,padchr) is like LPAD, but pads to the right.

REVERS(string) returns the string formed by reversing the order of the characters of its argument.

EXTIME(progname) causes the runtime system to output current timing statistics for the program progname and returns null, or fails if the program is not being timed.

REAL(x) is like INTEGER for reals.

EJECT() causes a page eject (form feed) to be assigned to OUTPUTC.

DAYTIM() returns an 11-character string representing the time of day (since midnight), as

```
HH:MM:SS.HH
```

meaning hours, minutes, and seconds to the nearest hundreth.

2.4.3 FORTRAN Primitives

These are predefined EXTERNAL.FORTRAN functions that, except for FREEZE, merely perform some simple arithmetic task and have integer values.

FREEZE() can be called to freeze the state of the FASBOL execution for resumption at some future date. When FREEZE is called, it exits to the monitor; the job may be SAVED, and when run again, it will start off by returning from the call to FREEZE. This is particularly useful for some applications that perform a considerable amount of initialization and wish to be able to start after that point on a repeated basis. No I/O devices (other than the console TTY) may be open at the time of the FREEZE and the call should be made from function level Ø if any timing is active.

ILT(int,int) or ILT(real,real)
[also ILE, IEQ, INE, IGE, IGT] Like LT, etc. except more efficient for dedicated variables or expressions.

AND(int,int)
[also OR, XOR, RSHIFT, LSHIFT, REMDR] perform the specified arithmetic or logical operation on their integer arguments and return the value (logical AND; inclusive OR; exclusive OR; logical right and left shift of first by second argument; remainder of integer division of first by second argument).

NOT(int) returns one's complement of its argument.

2.4.4 Library functions

These are additional library functions that add new features without changing the list of predefined primitives in the compiler. They are accessible via the EXTERNAL.FUNCTION declaration.

MEMBER(table,key) can be used to directly replace any occurrence of a table reference (i.e. table<key>), except that if the key is not already in the table, the reference fails (i.e. FRETURNs) and a new table entry for the key is not created. A normal table reference always suceeds and always creates a new entry if one does not already exist. Notice that, like a normal table reference, MEMBER() may appear on either the left or right side of an assignment.

2.5 Keywords

Three new keywords, all unprotected, have been added in FASBOL.

&STNTRACE is initially Ø, but if assigned a nonzero value it causes a trace output for each statement, giving statement number, program name, and time. This slows down execution considerably, so it is best to turn it on as close to the suspected bug as possible. Programs compiled under the NO.STNO option will ignore the value of &STNTRACE, however, so another approach is to run with all but the suspect program under NO.STNO, with &STNTRACE on all the time.

&DENSITY is initially 75, and represents the desired density of free storage immediately following a garbage collection. For example, &DENSITY = 75 means that the free storage system will try to maintain at least a 1:4 ratio between available and total storage immediately following a garbage collection, and will expand total storage as far as necessary or possible in order to try to maintain this ratio. See Section 3.2.4.

&SLOWFRAG is initially Ø, but if assigned a nonzero value it serves to switch in a heuristic in the free storage mechanism that slows down the rate of fragmentation of blocks at the expense of some wasted storage. See Section 3.2.4.

CHAPTER 3

3. FASBOL II Programming

Using FASBOL involves two separate stages, as in FORTRAN: compilation and execution. The first requires the compiler, an absolute program named FASBOL.SAV (or FASBOL.DMP, depending on the operating system). Execution of compiled (and then assembled) programs requires a library search, during loading, of the FASBOL library; this is a collection of relocatable programs named FASLIB.REL. The relative accesibility of these programs will depend on the installation. The compiler requires a minimum of 35K to run, and requires more core in proportion to the program being compiled. The core requirements for execution of user programs depends on the size of the compiled programs plus at most 5K (if every single facility in the library is used) for the library.

3.1 Using the compiler and runtime library

To compile a FASBOL program, type

```
 .RUN FASBOL n
```

where the CORE argument (n) is optional. It is best to give the compiler an amount of core commensurate with the size of program being compiled: this will increase compilation speed by minimizing garbage collections, since the compiler will expand core on its own only when it absolutely has to.

The compiler will respond with

, to which the user is expected to respond with a set of file specifications of the form

```
*macfil,lstfil_srcfil
```

Each file specification is of the standard form, as would be given to MACRO, for instance. The MACRO output file, macfil, is given a default extension of MAC if not specified. Lstfil is the listing file, and both macfil and lstfil are optional. The source file, srcfil, is given a default extension of SNO if not specified. Only one source file is permitted.

Examples:

```
*DTA3:SAMPLE,LPT:_SAMPLE
*,TAPE,LST_MTA0:
*NEW.NEW,_TEST.NEW
```

Once the compiler produces the MACRO output file, it must be assembled, using the Q flag to supress anxiety messages from MACRO:

```
 .COMPILE macfil(Q)
```

The MACRO file can be deleted after assembly, as it will be of little interest to most users; it is mainly a shortcut for the compiler to avoid having to generate relocatable code. On the other hand, those individuals who understand the workings of the run-time system may wish to hand-tailor these intermediate programs to suit their own needs; Caveat Emptor.

To prepare any collection of FASBOL and other programs for execution, the command list should be terminated with a library search of FASLIB, for example:

```
 .LOAD fil1,fil2, . . . ,filn,FASLIB/LIB
```

It is important that FASLIB be searched only once, after all FASBOL programs have been loaded, since it is very carefully sequenced to provide dummy versions of elements that are somehow referenced, but not really needed. The automatic search of the FORTRAN library should take place after searching FASLIB, since FASLIB may require some FORTRAN routines.

While FASBOL may call or may be called by FORTRAN or user MACRO programs, the main program must be a FASBOL program. Furthermore, the FASBOL runtime system enables traps and uses user UUO's 1 through 10, so it is incompatible with the FORTRAN runtime system. What this means is that FORTRAN programs used within a FASBOL execution must not do any I/O or otherwise cause FORSE. to be loaded. FASBOL does provide an infinite stack (all FASBOL stacks are infinite, up to user core limits) in register 17, however, so a broad class of FORTRAN user programs and library routines are permissible.

Unless changed by the user's program, all system output during execution is sent to the user's console; upon either error or normal termination of execution, the appropriate messages and statistics will be printed out, and control returned to the monitor. The error numbers are described in Appendix 3.

3.2 Programming techniques

Because of the basic differences between interpretive and compiler systems, and the additional features available in FASBOL, some programming techniques besides those discussed in [2], Ch. 11, are described here. An interested user may wish to get a listing of the compiler itself to see examples of some of these techniques.

3.2.1 Dedicated expressions

Dedicated expressions in FASBOL are those that are known, because of some component, to have a numerical value of a predetermined type. At one extreme is the totally dedicated statement that involves nothing but declared dedicated variables, constants, and perhaps FORTRAN calls. For example, if I were declared INTEGER, the statement

```
      I = 2 * I + 10
```

would be totally dedicated, and compile into

```
MOVE 1,I
IMULI 1,2
ADDI 1,1Ø
MOVEM 1,I
```

Even if an expression is mixed, with both dedicated and descriptor-mode subexpressions, in-line arithmetic code is compiled for as much of the expression as is possible to commit to a specific type of value (i.e. INTERGER or REAL) at compile time. It is therefore to the user's advantage to declare as many variables as he perceives will be dedicated in use to be of that dedicated type. Not only will the program run faster, it may even use less core. In a situation where all entities are descriptor mode, even arithmetic operators have to check the type of, and possibly convert each argument.

In this connection it should also be noted that the predefined FORTRAN primitives ILT, ILE, IEQ, INE, IGE, IGT have been provided in order to do a much more efficient job than LT, ..., GT when the arguments are dedicated. For example, if R and S are REAL, the test

```
ILT(R,S)
```

takes up several fewer words and runs about 1ØØ times faster than the test

```
LT(R,S)
```

FASBOL permits mixed mode (INTEGER and REAL) arithmetic, the general rule being that the result of an operation is INTEGER only if both sides are integer; furthermore, an arithmetic operation involving dedicated and descriptor mode values always has a dedicated result. A value being combined with a stronger mode is first converted, and then the operation is performed in that mode; for example, if I is INTEGER but D has not been declared dedicated,

```
I + D
```

implies the value of D will first be converted to an integer, and then added to I. The only exception to this rule is the ** (exponentiation) operator which permits a REAL raised to an INTEGER power.

Finally it should be noted that whereas the range of values of dedicated variables is the same as in FORTRAN, descriptor mode integers have a range two powers of 2 less in magnitude, and descriptor mode reals have two fewer bits of precision in the mantissa. The reason for this is that the two bits are needed for the descriptor type.

3.2.2 Use of the Unary? and . operators

Unary ? (interrogation) is useful to indicate to the compiler that an expression is evaluated for its effect, rather than value. For example, a frequent occurence in SNOBOL programs is the concatenation of null-valued functions for their sequential effects and/or succeed/fail potential. If the compiler knows that an element has a null value, it does not generate code to include it in the concatenation. Therefore it is efficient to precede predicates and other null-valued elements in a concatenation with the ? operator. This technique is especially valuable when combining predicates and dedicated arithmetic, as in

```
       I = ?IDENT(A,B) ?IGT(I,25) I + 1
```

, since concatenation is avoided entirely and the dedicated arithmetic is performed after the execution of the predicates without any need for conversion between dedicated and descriptor values.

Another frequent occurence in SNOBOL programs is the repetitive access of the same indirect variable, array element, or field of a programmer-defined datatype. Each of these accesses, whether to retrieve or store a value, involve some overhead which is repeated for each access. For example, in the statements

```
       $X = $X + 1
```

, the variable represented by the string value of X is looked up in the symbol table twice, the first time to retrieve its value, the second time to store into it. The unary . (name) operator can be used to save the result of one lookup by creating a NAME datatype, and then the NAME can be used in an indirect reference wherever the original expression was used. Instead of the above statement a more efficient sequence would be

```
       Z  = .$X
       $Z = $Z + 1
```

, where Z contains the NAME of the variable pointed to by X. The same considerations apply to array references and field references as apply to indirection; it is efficient to save the NAME of the variable referenced if it will be used more than once in close succession. For example, the statements

```
       A<25>      = F(A<25>)
       NEXT(LIST) = NODE(VAL, NEXT(LIST))
```

would be more efficient if coded as

```
       Z  = .A<25>
       $Z = F($Z)
       Z  = .NEXT(LIST)
```

```
        $Z = NODE(VAL, $Z)
```

It should be noted that TABLE references, and the symbol lookup involved makes it even more efficient to save the NAME. The NAME of an array or TABLE element, or of a field of a programmer-defined datatype, is only valid as long as that array table or datatype exist; attempts to retrieve or store using the NAME afterwards will have unpredictable results. Also, the NAME of a variable evaluated before that variable acquires an I/O association, does not reflect that association.

3.2.3 Pattern matching

Frequently a programmer wishes to write a degenerate-type statement consisting of a concatenation of elements executed for their effect, as in

```
        F(A) F(B) F(C) F(D)
```

This syntax, however, is parsed as a pattern match, and, though having the same effect as intended (providing the match is successful), is less efficient in both space and time. The original intent can best be achieved by enclosing the concatenation in parentheses, and in this case, using the ? operator

```
        (?F(A) ?F(B) ?F(C) ?F(D))
```

,which will suppress string concatenation.

One particularly unique feature of FASBOL is that explicit pattern expressions, i.e., those involving the pattern operators and/or primitives, are compiled as re-entrant subroutines, rather than constructed at run-time into intermediate-language structures. The significance of this to the programmer, aside from the increase in execution speed, is the there is less of a need to pre-assign subpatterns that will appear in pattern matches later on; in fact, an unnecessary pre-assignment will be slightly less efficient because the pattern match will have to recurse one level deeper than otherwise during execution. The way to determine the need for pre-assignment is to note how much evaluation is actually required in a subpattern; if little or none is required, it can just as efficiently be included in the body of the match. Of course, if a subpattern is large and/or used in several matches, the programmer may wish to pre-assign it anyway for convenience sake. Pattern evaluation involves only the elements of the pattern, not the structure itself. Literals and other constant values do not require evaluation, so the pattern

```
        TAB(7)  (SPAN('XYZ') ! BREAK(';')) $ SYM ';??'
```

requires no evaluation at all. A generally applicable rule for all FASBOL programming is that it is more efficient to use, wherever possible, literals instead of variables with constant value. Simple variables appearing in a pattern require little evaluation (only a determination if they have a string or pattern value), and even character class primitives (i.e. SPAN, BREAK, etc.) require little evaluation, if their argument is non-literal, provided the argument is a variable with a constant value. Examples of pattern elements requiring more extensive evaluation are (non-pattern primitive) function calls, non-pattern expressions requiring considerable evaluation in their own right, and character-class primitives whose arguments are other than literals or simple variables. An example of the latter case would be

```
        ANY('XYZ' OTHER)
```

; even if the value of OTHER remains constant, the concatenation produces a new string each time, which prevents ANY from immediately using the break table it has generated on the last execution of that call. It has been assumed in all this discussion of pattern evaluation that the value of the pattern element would not change value between the time of assignment of the subpattern and its use in a match. Should this not be the case, of course, the alternative of including the subpattern in the match does not exist.

Even when integer constants cannot be used, it is still helpful to use dedicated integer variables or expressions in patterns, if possible. Dedicated integer expressions are ideally suitable as arguments to the positional pattern primitives (i.e. POS, LEN, etc.), and integer variables are ideally suitable as objects of the cursor assignment operator (@). For example, suppose one wishes to take a string composed of sentences separated by semicolons (and terminated by a semicolon) and output the sentences on separate lines. A single pattern match to do this would be

(P is INTEGER):

```
        STRING   @P  SUCCEED  TAB(*P)  BREAK(';') $ OUTPUT  LEN(1)
+       @P  RPOS(Ø)
```

Note that the pattern requires no evaluation.

Since FASBOL does not employ the QUICKSCAN heuristic of assuming at least one character for unevaluated expressions, left-recursive patterns will loop indefinitely at execution time, as they would in SNOBOL4 under FULLSCAN. Usually a set of patterns involving left recursion can be re-written to eliminate it. To take a simple example, the pattern

```
        P = *P  'Z' ! 'Y'
```

, which matches strings of the form 'Y', 'YZ', 'YZZ', 'YZZZ', etc., could be re-written as the pair of patterns

```
        P1 = 'Z'  *P1 ! ''
        P = 'Y'  P1
```

3.2.4 Timing and storage management

The TIMER option permits the programmer to monitor the operation of any (or all) separately compiled programs, and provide feedback on where the time is being spent. Initial programming of some problem can be done rapidly with not much attention being paid to optimization. It is usually the case that some small sections of a program account for a large percentage of the execution time; these are identified using the TIMER option. The programmer's time is then spent most efficiently optimizing the critical areas and ignoring the rest. Of course, after a series of optimizations, a new bottleneck will develop; the process can then be iterated until the law of diminishing returns takes hold. Finally, the TIMER declarations can be removed and the programs run in production mode.

The programmer has a large degree of control over storage management in FASBOL, which in turn means control over the space/time tradeoff that exists due to the dynamic storage allocation system (free storage). To begin with, requests from the free storage system prior to the first garbage collection (regeneration of dynamic storage) have very little overhead compared to ones subsequent to the first garbage collection. Unless there are good reasons for the contrary, the user should capitalize on this by starting his execution with approximately the amount of core he expects will eventually be required - past experience with the program is the best guide. Thus the number of garbage collections will be reduced to a minimum, and initial execution speeded up. In the absence of a core specification, the program will begin with the minimum required for loading, and will expand core as it becomes necessary, but undergoing more garbage collections.

The &DENSITY keyword is also useful in controlling the space/time tradeoff. &DENSITY may be set dynamically to any value between 1 and 100; immediately following a garbage collection, the dynamic storage allocation mechanism attempts to satisfy this value, interpreted as the percentage of total storage allocated that is in use at that time. Nothing is done unless the actual ratio is greater than the desired one, in which case core is expanded to satisfy the desired ratio, or until user core limits are reached. For example a user who sets &DENSITY to 99 is saying he wishes to keep his core size to a minimum, and is willing to pay a (rather large) premium in repeated garbage collections. On the other hand, a user who sets &DENSITY to 1 is asking for all the core he can get, in order that his program execute as rapidly as possible. It is also perfectly feasible to use a strategy where &DENSITY is set to different values at different times during execution. The initial value of &DENSITY is 75, which represents a general-purpose compromise.

If a user's application will occasionally require large contiguous blocks of storage, he may give himself 100% insurance by reserving dummy arrays of the appropriate size at the very beginning of his program. An alternative is to turn on the keyword &SLOWFRAG, which activates a heuristic which tends to slow down the fragmentation of large blocks at the expense of some wasted storage. While not 100% guaranteed, it will give the desired effect in most cases, minimizing the situation where a large block is called for, and though enough total storage is available, no contiguous area is large enough to satisfy the request.

Finally, the COLLECT primitive may be invoked at appropriate times, both to force a regeneration and also measure the amount of storage that is available.

3.2.5 FORTRAN interface

External FORTRAN subroutines, whether user-written or from the FORTRAN library, must be declared, including the number of arguments expected. A function call to the external subroutine may have any expression (except patterns) as arguments, but all but a few recognized expressions are assumed to evaluate to integers and will cause an error exit if not. Dedicated integer and real variables are passed directly to the subroutine, as they would in a call from a FORTRAN program. Also, dedicated integer and real expressions are evaluated, the value is saved in a temporary location, and this location passed to the FORTRAN routine. Finally, dedicated string variables and literals are passed to FORTRAN as a vector, the first word of which is pointed to by the calling sequence, and the FORTRAN routine may interpret it as one-dimensional array of ASCII characters packed five to a word. In addition to returning an integer or real value, the function may modify the value of any dedicated integer, real, or string variable that is passed to it. In the last case, not only may the characters be modified, but the character count may be changed by storing it in the right half of the word immediately preceding the first word of the string (array(0)). For example, suppose a FASBOL program contains the declarations

```
DECLARE('INTEGER','I')
DECLARE('REAL','R')
DECLARE('STRING','S(15)')
DECLARE('EXTERNAL.FORTRAN.FUNCTION','GETDAT(3)')
```

and the FORTRAN function GETDAT is defined as

```
FUNCTION GETDAT(INDEX,ISTR,IDAT)
COMMON IDATA(1000,4)
EQUIVALENCE (RDATA,IDATA)
DIMENSION ISTR(3),RDATA(1000,4)
ISTR(1)= IDATA(INDEX,1)
ISTR(2)= IDATA(INDEX,2)
ISTR(0)= (ISTR(0)/2**18)*2**18+10
```

```
        IDAT= IDATA(INDEX,3)
        GETDAT= RDATA(INDEX,4)
        RETURN
        END
```

, then a typical use of GETDAT within the FASBOL program might be

```
        R = GETDAT(2,S,I)
```

,which would have the effect of setting I to some integer value, R to some real value, and S to a 10-character string.

Entries that are expected from FORTRAN must be declared with the ENTRY.FORTRAN.FUNCTION declaration. This works like ENTRY.FUNCTION in that an automatic DEFINE is performed on the first call. Valid actual arguments in the FORTRAN call to FASBOL can be integers, reals, and Hollerith arrays (as described above, denoted by the codes 0,2, and 5 in the calling sequence. Upon entering FASBOL, the actual arguments are copied, and converted if necessary, into the formal arguments, which are dedicated or descriptor mode FASBOL variables. The right half of the word immediately preceding the first word of a Hollerith array is considered to be the character count, and may be modified by FASBOL if the string argument is modified. Upon return to FORTRAN (via RETURN), the function value is determined by dedicated or descriptor value in the variable corresponding to the function name, but must be integer or real. The formal argument values are copied back (and re-converted, if necessary) into the actual arguments in the FORTRAN calling sequence, thus providing a means of passing back additional values, besides the function value, to FORTRAN.

It should be noted that FORTRAN is not recursive, and therefore any recursive combination of FASBOL and FORTRAN calls will not work. Even when a series of calls is not recursive, care must be taken not to re-enter a FASBOL routine which has a FORTRAN call pending, because the FASBOL routine uses the same temporary storage locations for all FORTRAN calls, including the predefined primitives IGT, etc.

In writing FORTRAN programs to be used with FASBOL programs, care should be taken not to perform any I/O, or use any other FORTRAN facility that requires the FORTRAN runtime system (FORSE.) to be loaded.

APPENDIX 1

Syntax for FASBOL II

Explanation of syntax notation

1. All terminal symbols are underlined, the remainder of the syntax consisting of non-terminals and syntax punctuation.

2. The ::= operator indicates equivalence.

3. The | operator indicates a series of alternatives.

4. The blanks between consecutive elements indicate concatenation.

5. The \ operator indicates the specific ruling out of the immediately following element as a precondition for further concatenation.

6. The ... operator indicates the indefinite repetition of the immediately preceding element.

7. The < > brackets serve to group expressions into a single element.

8. The [ ] brackets indicate the optional occurence of the expression contained within brackets, and also serve to group the expression into a single element.

9. The order of precedence for the operators, from highest to lowest, is: \ ... (blanks) | ::=.

Syntax

```
program::= [declaration | comment]... [execute.body] end.statement
declaration::=bl DECLARE( [bl] declaration.type [bl] ) eos
comment::= * [char]... eol | - [bl] control.type [bl] eol
execute.body::= statement [statement]...
end.statement::= END [bl label] eos
bl:= <blank | tab> [bl] | eol <+ | .> [bl]
eos::= [bl] <; | eol>
eol::= carriage.return linefeed [formfeed]...
statement::= comment | [label.field] [statement.body] [goto.field] eos
label.field::= \<END bl> label
goto.field::= bl : [bl] <goto | S goto [bl] [F goto] | F goto [bl]
        [S goto]>
goto::= ( [bl] <identifier | $ string.primary> [bl] )
statement.body::= degenerate | assignment | match | replacement
degenerate::= bl string.primary
assignment::= bl variable equals [bl expression]
```

```
match::= bl string.primary bl pattern.expression
replacement::= bl variable bl pattern.expression equals [bl
        string.expression]
equals::= bl <= | >
variable::= \pattern.identifier identifier | & unprotected.keyword
        | string.variable
expression::= string.expression | \<string.primary
        [bl string.primary]... > pattern.expression
pattern.expression::= conjunction [bl ! bl conjunction]...
conjunction::= pattern.term [bl pattern.term]...
pattern.term::= pattern.primary [<bl . bl | bl $ bl>
        pattern.variable]...
pattern.primary::= pattern.identifier | pattern.primitive | @
        pattern.variable | [*] string.primary | sum | ( [bl]
        pattern.expression [bl] )
pattern.variable::= [*] variable
string.expression::= sum [bl sum]...
sum::= term [<bl + bl | bl - bl> term]...
term::= factor [<bl * bl | bl / bl> factor]...
factor::= string.primary [bl <** | > bl string.primary]...
string.primary::= \pattern.identifier identifier | literal | &
      < unprotected.keyword | protected.keyword> | string.variable |
      <? | \ | - | +> string.primary | . variable | (
      [bl] string.expression [bl] )
literal::= integer.literal | real.literal | string.literal
string.variable::= $ string.primary | array.element |
        procedure.call
procedure.call::= \pattern.primitive < identifier ( [bl]
        [parameter.list] [bl] )>
array.element::= \pattern.identifier identifier << | [> [bl]
        [parameter.list] [bl] <> | ]>
parameter.list::= expression [pc expression]...
pc::= [bl] , [bl]
identifier::= letter [letter | digit | . | -]...
label::= \<blank | tab | . | + | - | *> char [\<blank
        | tab | ;> char]...
integer.literal::=  digit [digit]...
real.literal::= digit [digit]... . [digit]...
string.literal::= ' [\<' \'> <'' | cont.char>]... ' | " [\<" \">
        <"" | cont.char>]... "
cont.char::= char [eol <+ | .>]...
letter::= A | B | C | D | E | F | G | H | I | J |
        K | L | M | N | O | P | Q | R | S | T | U |
        V | W | X | Y | Z | a | b | c | d | e | f |
        g | h | i | j | k | l | m | n | o | p | q |
        r | s | t | u | v | w | x | y | z
digit::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
char::=  any printing.character
protected.keyword::= STECOUNT | LASTNO | STNO |
      FHCLEVEL | STCOUNT | ERRTYPE | RTHTYPE | ALPHABET
unprotected.keyword::= ABEND | ANCHOR | FULLSCAN |
      STNTRACE | MAXLNGTH | STLIMIT | ERRLIMIT |
       DENSITY | INPUT | OUTPUT | DUMP | SLOWFRAG
pattern.identifier::= FAIL | FENCE | ABORT | ARB |
        BAL | SUCCEED | REM
pattern.primitive::= <LEN | TAB | RTAB | POS | RPOS |
        SPAN | NSPAN | BREAK | BREAKX | ANY |
        NOTANY> ( [bl] <string.expression | * string.primary>
        [bl] ) | ARBNO( [bl] pattern.expression [bl] )
control.type::= LIST | UNLIST | NOCODE | CODE |
        EJECT | SPACE [bl integer.literal] | NOCROSS |
        CROSREF | FAIL | NOFAIL
declaration.type::=
        'OPTION' pc <'NO.STNO' | 'TIMER' | 'HASHSIZE=
                integer.literal '> |
        'SNOBOL.MAIN' pc ' identifier ' |
        'SNOBOL.SUBPROGRAM' pc ' identifier ' |
        'PURGE.VARIABLE' pc <ALL | ' identifier.list '> |
        'UNPURGE.VARIABLE' pc ' identifier.list ' |
        'PURGE.LABEL' pc <ALL | ' label.list '> |
        'UNPURGE.LABEL' pc ' label.list ' |
        'PURGE.FUNCTION' pc <ALL | ' identifier.list '> |
        'UNPURGE.FUNCTION' pc ' identifier.list ' |
        'STRING' pc ' string.specifier.list ' |
        'INTEGER' pc ' identifier.list ' |
        'REAL' pc ' identifier list ' |
        'RENAME' pc ' identifier ' pc ' identifier ' |
        'GLOBAL.VARIABLE' pc ' identifier.list ' |
        'GLOBAL.LABEL' pc ' label.list ' |
        'GLOBAL.FUNCTION' pc ' identifier.list ' |
        'EXTERNAL.VARIABLE' pc ' restricted.identifier.list ' |
        'ENTRY.VARIABLE' pc ' restricted.identifier.list ' |
        'EXTERNAL.LABEL' pc ' restricted.label.list ' |
        'ENTRY.LABEL' pc ' restricted.label.list ' |
        'EXTERNAL.FUNCTION' pc ' restricted.identifier.list ' |
        'ENTRY.FUNCTION' pc ' restricted.identifier ( [bl]
                [identifier.list [bl] ) [[bl] identifier.list] ' [pc
                ' label '] |
        'EXTERNAL.FORTRAN.FUNCTION' pc ' fortran.identifier.list ' |
        'ENTRY.FORTRAN.FUNCTION' pc ' restricted.identifier ( [bl]
                [identifier.list] [bl] )' [pc ' label ']
 identifier.list::= identifier [pc identifier]...
 label.list::= label [bl label]...
 string.specifier.list::= string.specifier [pc string.specifier]...
 string.specifier::= identifier ( integer.literal )
 restricted.identifier.list::= restricted.identifier
        [pc restricted.identifier]...
 restricted.label.list::= restricted.identifier
        [bl restricted.identifier]...
 fortran.identifier.list::= fortran.identifier
        [pc fortran.identifier]...
 fortran.identifier::= identifier [= <INTEGER | REAL>]
        ( integer.literal )
 restricted.identifier::= letter [lnd [lnd [lnd [lnd [lnd ]]]]]
```

lnd::= letter | digit | .

APPENDIX 2

Predefined symbols

1. GLOBAL and EXTERNAL variables
INPUT INPUTC OUTPUT OUTPUTC

2. GLOBAL and EXTERNAL labels
END FRETURN NRETURN RETURN

3. EXTERNAL.FORTRAN functions (all integer valued) AND(2) FREEZE(Ø) IEQ(2) IGE(2) IGT(2) ILE(2) ILT(2) INE(2) LSHIFT(2) NOT(1) OR(2) REMDR(2) RSHIFT(2) XOR(2)

4. Primitive pattern variables
ABORT ARB BAL FAIL FENCE REM SUCCEED

5. Primitive pattern functions
ANY ARBNO BREAK BREAKQ BREAKX LEN NOTANY NSPAN POS RPOS RTAB SPAN TAB

6. Predefined primitive functions
APPLY ARRAY CLOSE COLLECT CONVERT COPY DATA DATATYPE DATE DAYTIM DEFINE DETACH DIFFER DUPL EJECT ENTER EQ EXTIME GE GT IDENT INPUT INSERT INTEGER ITEM LE LGT LOOKUP LPAD LT NE OPEN OPSYN OUTPUT PROTOTYPE REAL RELEASE REPLACE REVERS RPAD SIZE SUBSTR TABLE TIME TRIM

7. Predefined library functions
MEMBER

APPENDIX 3

Runtime Errors

Conditionally Fatal

1. Illegal Data Type
2. Error in Arithmetic Operation
3. Erroneous Array or Table Reference
4. Null String in Illegal Context
5. Undefined Function or Operation
6. Erroneous Prototype
7. Dedicated String Overflow
8. Variable Not Present Where Required
9. Real to String Conversion Overflow
1Ø. Illegal Argument to Primitive Function

11. Reading Error
12. Illegal I/O Unit
13. Limit on Defined Datatypes or Tables Exceeded
14. Negative Number in Illegal Context
15. String Overflow

Unconditionally Fatal

17. Error in FASBOL System
18. Return from Zero Level
19. Failure During Goto Evaluation
20. Insufficient Storage to Continue
21. Illegal Memory Reference
22. Limit on Statement Execution Exceeded
23. Object Exceeds Size Limit
24. Undefined or Erroneous Goto
25. [unused]
26. [unused]
27. Writing Error
28. Execution of Statement with Compilation Error
29. Failure Under NOFAIL
30. Divide Check
31. Arithmetic Overflow

FILEX

The FILEX program is a general file transfer program intended to convert between various core image formats, and to read and write various directory formats. It is primarily useful for DECtapes.

A writeup on FILEX can be found in the DECSYSTEM1Ø Users Handbook, second edition, page 557.

Following are the only three command strings which will work for 11 format tape:

1) to put files A.A,B.B,C.C on 11 format tape:
```
*DTA1:(ZQV)_DSK:A.A,B.B,C.C
```

2) to read the whole 11 DTA onto DISK:
```
*DSK:_DTA1:*.*(QV)
```

3) to list a directory from an 11 format tape:
```
*_DTA1:(VL)
```

WARNING:

Since FILEX was written for the DECsystem1Ø Monitor, not all forms of the command string will work under TENEX.

FIOCNV

Program to convert standard TENEX text files (ASCII) to punched paper tape for either flexowriter (FIO-DEC code) or Dura typewriter. FIO stands for Flexowriter Input Output.

@FIOCNV

Asks all necessary questions including name of source file and format of output. Prints message for any character not translatable into object code set.

```
@FIOCNV

TO OR FROM PAPER TAPE? (T-F) T
INPUT FILE: WIN.MOVES;1$

FLEXO OR DURA? (F OR D) D
CAN´T CONVERT ^^ (36)

INPUT FILE:
```

FLIST

FORTRAN LISTING

FLIST copies FORTRAN-generated ASCII disk files to the line printer or another disk file. The first character (column 1) of each ASCII record is not printed, but is interpreted as a carriage control character according to the DECsystem-10 FORTRAN Standards. Characters other than those listed have the same effect as a space in column 1.

Character		Effect
	space	skip to next line with a FORM FEED after every 60 lines
0	zero	skip a line
1	one	form feed-go to top of next page
+	plus	suppress skipping - will overprint line
*	asterisk	skip to next line with no FORM FEEDS
-	minus	skip 2 lines
2	two	skip to next 1/2 of page
3	three	skip to next 1/3 of page
/	slash	skip to next 1/6 of page
.	period	skip to next 1/20 of page
,	comma	skip to next 1/30 page

FLIST also has the option of replacing all line feed characters with a DC3 character, which has the properties of a line feed character but inhibits the automatic FORM FEED after every 60 lines.

FLIST asks for the name of the file to be listed as shown below. TENEX file name recogniton is in effect. A confirming carriage return must follow the file name. FLIST then asks if the output is to go to the lineprinter (LPT:) or a file (filename). A confirming carriage return must follow either the LPT: or the file name. FLIST then asks if the DC3 option is to be activated.

EXAMPLE:

```
@FLIST%

INPUT FILE: TEST.DAT% [CONFIRM]%

OUTPUT FILE (LPT: OR FILENAME):TST.LST [CONFIRM]%

CONVERT LINEFEED TO DC3 (Y OR N)?Y
INPUT FILE:  TEST2.DAT;1 [CONFIRM]%


OUTPUT FILE: LPT: [OK]%

CONVERT LINEFEED TO DC3 (Y OR N)?N
INPUT FILE: ^C
@
```

NOTE: Output disk files which have used the DC3 option should be listed on the lineprinter with the COPY command instead of the LIST command.

FLOW

FLOW is an automatic flowcharting program which produces a flowchart from a FORTRAN source file.

This subsystem was originally distributed by Digital Equipment Corporation as DECUS 10-38 and has been modified by BBN.

FLOW requests an input file name. This should be answered with the name of a FORTRAN source file on the DSK. (A maximum of 5 characters is allowed.) It then requests the input file extension and an extension of up to 3 characters should be supplied. An output file name is requested and a file name of up to 5 characters should be supplied for the DSK output file. An output file extension is then requested and an extension of up to 3 characters should be supplied.

During processing, a binary scratch file FOR20.DAT will be created on the DSK for each main program and subroutine. FLOW will delete FOR20.DAT after it has been processed. The ASCII output file will consist of a program listing of the main program and each subroutine/function with each associated flow chart followed by a list of all statement numbers used by the associated main program or subroutine/function.

```
@FLOW

OUTPUT BINARY SCRATCH FILE ON DSK:FOR20.DAT
BEGIN EXECUTION



INPUT FILE NAME (5 CHARS)=FLOW%
INPUT FILE EXTENSION (3 CHARS)=F4%
OUTPUT FILE NAME (5 CHARS)=FLOW%
OUTPUT FILE EXTENSION (3 CHARS)=LST%
FLOW-CHARTING PROGRAM MAIN
FLOW-CHARTING PROGRAM PRNT
FLOW-CHARTING PROGRAM GORT
FLOW-CHARTING PROGRAM ASSIGN
FLOW-CHARTING PROGRAM MOVE
FLOW-CHARTING PROGRAM REPT
FLOW-CHARTING PROGRAM NUM
FLOW-CHARTING PROGRAM NUMAL
```

```
FLOW-CHARTING PROGRAM BLOCK DATA
FLOW-CHARTING PROGRAM ERROR
FLOW-CHARTING PROGRAM I REFIN
FLOW-CHARTING PROGRAM SORT
FLOW-CHARTING PROGRAM XTRACT
FLOW-CHARTING PROGRAM TABFIX
FLOW-CHARTING PROGRAM TABPAK



     END OF JOB

CPU TIME: 4:0.96     ELAPSED TIME:  17:10.00
NO EXECUTION ERRORS DETECTED

EXIT.
^C
```

Note: Before doing FLIST of output file put WIDE paper in the Line Printer.

```
@FLIST%
INPUT FILE: FLOW.LST%[CONFIRM]%

OUTPUT FILE (LPT: OR FILENAME):LPT:%[OK]%

CONVERT LINEFEED TO DC3 (Y OR N)?Y
INPUT FILE: ^C%
@

EXECUTION TIME:           9.88 SEC.
TOTAL ELAPSED TIME:       2 MIN. 18.00 SEC.
NO EXECUTION ERRORS DETECTED
EXIT.
^C
```

FORTRAN System

The DEC FORTRAN-IV Compiler, for further information see DECsystem10 FORTRAN-IV Manual.

To obtain the FORTRAN compiler call F40.

COMPILING AND RUNNING A FORTRAN PROGRAM

It is assumed that the user has a FORTRAN program on a file in a card image format (one FORTRAN statement per line with the statement body starting in column 7 or greater as usual. Lines may be up to 80 characters terminating with a carriage return line feed). In the examples below we will call the FORTRAN program file X.

The FORTRAN compiler, F40, takes only one command of the form

```
object-file, listing-file _ input-file1,input-file2,...
```

All file names may include a device, but not a version number. Furthermore, the name part is limited to 6 characters and the extension (if any) to 3 characters.

The object-file is the relocatable binaries output by the compiler, and will assume a default extension of .REL if no extension is given.

The listing file is for the compiler listing (error messages will also appear on the user´s terminal as compilation proceeds), and will assume a default extension of .LST if no extension is given.

As shown, there may be several input files separated by commas. The compiler will first look for an extension of .F4 (not .F40), and failing to find that, look for a file with no extension.

EXAMPLES:

To get both relocatable binaries and listing:

```
@F40
*X,X_X
```

The first X will be X.REL, the second X.LST
The last X could have no extension or an extension of .F4.

To get just relocatable binaries with no listing:

```
@F40
*X_X     The first X will be X.REL
```

THE LOADER

After the FORTRAN programs are compiled, the relocatable binaries output by the compiler must be loaded along with required routines from the FORTRAN library. Assuming the output from F4Ø is on X.REL, this is done by:

```
     @LOADER
     */SX$        ($ represents the escape character - it causes
                   the load process to be completed with a search of
                   the FORTRAN library and a return to the EXEC)
```

If several relocatable binary files representing the separate compilations of the main program and subroutines of a system are to be loaded together (say X.REL, Y.REL, and Z.REL), then do:

```
     @LOADER
     */SX,Y,Z$
```

The ´/S´ is not required, but is desirable if the program is to be debugged via DDT. It causes the Loader to leave the user´s FORTRAN symbols (variables and statement numbers) in core for use by DDT. If no debugging is to be done, the /S may be left off, but its use is encouraged.

SAVING THE ABSOLUTE CORE IMAGE

The Loader has no "output file". Instead, everything is loaded into core and left there. Thus after the escape ($) completes the load and causes the exit to the EXEC, it is necessary to save the assembled core image (if repeated executions are desired - if they are not, the program may be immediately started, but then the load process must be repeated for another execution). This is done by:

```
     @SAV$$$X$ ret    (the $ is escape so there are four escapes
                           here altogether)
```

This will echo as

```
     @SAVE (CORE FROM) 2Ø (TO) 777777 (ON) X.SAV;1 [NEW FILE]
```

and commands that the entire core image be saved on file X.SAV (which is here assumed to be a new file - a new version would be created if an earlier version of X.SAV already existed).

The program may then be started by

```
@START ret
```

and may be repeatedly executed by simply typing

```
@X ret
```

NOTES ON DEBUGGING FORTRAN Programs via DDT

These notes are not a complete explanation of DDT. it is assumed that the reader has at least glanced at the DDT manual and that he has a copy convenient for reference. it is also assumed that the reader knows how to compile, load, and run a FORTRAN program.

We use the following conventions below:

$	represents the altmode or escape character.
^x	represents control-x, e.g., ^C is control-C.
lf	represents line-feed.
ret	represents carriage return.

No convention is established for distinguishing user type-ins from the system´s response. Most DDT commands are short and the system will respond as soon as the command is typed (that is, no carriage return is necessary with DDT commands). Thus, the novice user should be able to get started by typing slowly and waiting for the system´s response.

PREPARATION FOR DEBUGGING - THE LOADER

Although it is possible to get an assembly language listing from the FORTRAN compiler and a load map from the Loader, this is not usually necessary in debugging a FORTRAN program. It is however necessary to instruct the Loader to save the symbol table that is automatically output by the compiler, so that the user´s symbols will be in core and available for reference after the loading process. This is done by using the /S switch during loading. Example: assume the user intends to debug program PROG. The Loader command would then be

```
        @LOADER
        */SPROG$            (the $ is the altmode)
```

After the load is complete, the usual TENEX SAVE command is executed, and whenever the program is loaded into core to be run, the user´s FORTRAN symbols will also be brought in for debugging.

USER AND OTHER SYMBOL NAMES

In addition to the user´s symbols (that is, his variable and subprogram names), there are several types of symbols created by the compiler which the user will need to reference:

1. The user´s FORTRAN statement numbers will all be suffixed by a ´P´. Example: if the following statement appears in the program,

   ```
   11Ø A = B
   ```

   then the machine instruction to load B will be labelled with the symbol 11ØP.

2. The main program is always labelled MAIN. (the . is included as part of the name), and all FORTRAN programs start at location MAIN. .

3. When the compiler needs to generate a label for a jump instruction, it uses the form nM where n is an integer starting at 1 for the first label generated. Examples: 1M, 3M, 17M, etc. In particular, the first executable statement of any FORTRAN program (not containing arithmetic statement functions) will be labelled with ´1M´ (if not labelled by the user).

GETTING IDDT AND THE RUNNING FORTRAN PROGRAM TOGETHER

There are two approaches - IDDT and DDT. The best is to use IDDT. IDDT contains many useful features including the ability to interrupt a running program and then restart it (as opposed to break-pointing it as required by the old DDT). Documentation for IDDT is available in the TENEX Users´ Guide.

To use IDDT, the user may do one of two things: he may first run his program and when it terminates invoke IDDT, or he may start IDDT, use it to load the program via the ;Y (for Yank) command, and then start it with $$G (two altmodes, G), possibly after setting breakpoints.

Example of the first method (assume program named PROG):

```
@RUN PROG
```

(assume program fails and the system reports errors on the TTY and returns to the EXEC; or alternatively, the program stopped for TTY input and the user interrupted it with ^C.)

```
        @IDDT
```

The user is now in IDDT in the context of the program just run, and he may proceed with the commands described below. If the program was interrupted with ^C, it may be continued with $P (altmode, P for Proceed).

Example of the second method:

```
        @IDDT

        ;Yank file: PROG.SAV$ ret


        MAIN.$:      11ØP$B      $$G
```

(If the program requests TTY input before reaching the breakpoint, it may be typed in normally - if a user´s program is waiting for input there is no confusion over whether the input goes to the program or to IDDT.)

```
        $1B>>11ØP
```

(IDDT signals the break as shown and waits for commands. The user may now examine variables, insert or remove breakpoints, and continue the program from the break or from some other location, etc., as described below.)

EXAMINING FORTRAN AND OTHER VARIABLES

Indicating the Routine to be Debugged

We must first consider the establishment of the proper context when referring to variable (or any other kind of) names. Consider a program that consists of a main routine and a subroutine called SUB. The compiler will automatically give the name MAIN. to the main routine (including the period). The command for establishing a routines symbols as the current context is of the form ´name$:´ (routine name, altmode, colon). Thus, if we wish to debug the main routine we first type

```
        MAIN.$:
```

To then debug the subroutine we type

```
        SUB.$:
```

Thus we may refer to the symbols of several different routines by first establishing the context as that routine with the colon command.

The colon command is not always absolutely necessary. If variable X exists in the main routine but not in routine SUB, and if we have issued SUB$:, and we then type ´X/´, the system will respond with

```
        X/[_MAIN.$:X]    ...
```

This does not change the context, but shows us that there is no X in the current context. Note that if several different X´s exist outside of the current routine, then the colon command may be necessary to refer to the desired one.

Typing Out a Location and the Current Type-out Mode

In general, typing any location name followed by a slash (/) will cause IDDT to type out the contents of the location in the current type-out mode. This mode is initially symbolic. Thus, typing ´110P/´ will cause IDDT to type out the instruction which starts statement 110. For variables, however, we would like the type-out to be numeric. The mode may be set permanently to floating point by ´$$F´ (two altmodes followed by ´F´), or temporarily by ´$F´. (The floating point mode will also type integers correctly.)

Examples:

```
        110P/   MOVE 2,A   ret
        X/   MOVEI 11,@146315(3)    $F; 1.2000   lf
        X+1/   1.3000

        $$F       X/   1.2000   lf
        X+1/   1.3000   ret
        110P/   17196647924.   $S; MOVE 2,A
```

In the first three lines, the user first opens location 110P which types out symbolically. Then X is opened which also types out symbolically, and then the altmode-F-; is used to show the corresponding numeric interpretation (the ´; ´ - semicolon, space - command causes IDDT to retype the last value typed, presumably in a new mode). Finally a line feed causes the cell after X to be opened (assume X is an array), and typed out in the temporary mode. This temporary mode will continue until a carriage return is typed, after which the mode will revert to the current permanent mode. Thus, multiple line feeds will continue in the temporary mode, or additional cells may be opened by name in the temporary mode:

```
        $FX/   1.2000      Y/   16.
```

In the second example above, the user set the mode permanently to floating point. then, when 110P was opened, its floating point representation was converted back to symbolic with the ´$S´ command. as with F, ´$$S´ would be used to set the type out mode permanently to symbolic.

Examining Related Locations

Once a word has been typed out, related words may be examined by:

lf	line feed examines the next word.
^	up arrow (shift N) examines the previous word.
/	examines the word specified by the address of the current word, but does not move the location pointer thus, lf and ^ are still relative to the first original word).
tab	(control-I) changes the location pointer to the word specified by address of the current word, and types out the contents of this new current word (especially useful for examining array parameters - see below).

Examining Arrays Passed as Parameters to Subprograms

Assume we have a main program with an array X, and that the array is passed as parameter Y to subprogram SUB. That is, we have

```
        REAL X(100)
            .
            .
        CALL SUB(X)
            .
            .
        SUBROUTINE SUB(Y)
        REAL Y(100)
            .
            .
```

The following shows how the user may examine the contents of X in floating point format while in the subroutine:

```
SUB$:   Y/   JUMP MAIN.X  tab
MAIN.X/   MOVEI 11,@146315(3)    $F; 1.2   lf
MAIN.X+1[   1.34   lf
MAIN.X+2[   6.21
   .
   .
   .
```

As may be seen, when the user opened Y within the subroutine, he got not the contents of array X, but the address of array X (for simple scalar variables passed as parameters, examining the parameter gets the contents of the actual parameter as it is used within the subprogram). These array parameter addresses are always of the form ´JUMP ADDRESS´, and in fact, seeing a parameter of that form is a reliable way of telling that a routine has been passed an array as opposed to a scalar variable or expression. To return to the example, the user then moved the location pointer to the array X by using the tab command (which moves the pointer to the address specified by the word last typed, in this case the MAIN.X). He then changed the type-out mode to floating point and continued to examine successive words of the array with successive line feeds.

Value of LOGICAL Variables

Logical variables set to the values .TRUE. and .FALSE. have the values -1 and 0 respectively in core.

BREAKPOINTS, PROCEEDING, AND OTHER CONTROL

We now present a nonsensical example of a FORTRAN program in both the FORTRAN form and as typed out by IDDT. The example will be used throughout the discussion of breakpoints and other control commands.

```
@COPY TST.;3 (TO) TTY: [OK]

      REAL A(10)
      READ(-1,60100) A,B
60100 FORMAT(11F10.0)
C
  110 IF(A(1).EQ.B) GO TO 120
      B=1.2
  120 CALL SUB(A,B)
      END
      SUBROUTINE SUB(X,Y)
      REAL X(10)
      END

@LOADER
*/STST$
LOADER 2K CORE
5+4K MAX 522 WORDS FREE
EXIT.
^C
@SAVE (CORE FROM) 20 (TO) 777777 (ON) TST.SAV [New version]
@IDDT

MAIN.$:  1M/  MOVEI 1,60100P         lf
1M+1/   16040,,-1          lf
1M+2/   25100,,A           lf
1M+3/   JUMP 12            lf
1M+4/   20100,,B           lf
1M+5/   21000,,0           lf
60100P/   JRST 2M          lf
```

```
        60100P+1/   ROT 10,@143142(6)   $T; (11F1      lf
        60100P+2/   0.0)                ret
        lf
        2M/   MOVE 2,B                  lf
        2M+1/   CAMN 2,A                lf
        2M+2/   JRST 120P               lf
        2M+3/   MOVE 2,CONST.           lf
        2M+4/   MOVEM 2,B               lf
        120P/   JSA 16,SUB              lf
        120P+1/   JUMP 2,A              lf
        120P+2/   JUMP 2,B              lf
        120P+3/   JSA 16,EXIT           lf
        MAIN./   15000,,0               lf
        MAIN.+1/   JRST 1M              lf
        CONST./   MOVEI 11,@146314(3)   $F; 1.1999999     ret
        lf


        B/   0              lf
        A/   0              lf
        A+1/   0            lf
        A+2/   0            lf
        A+3/   0            lf
           .
           .
           .
```

We first note several things about the IDDT output:

1. The "instructions" from 1M+1 to 1M+5 are system calls to perform the READ statement. In general, the user can expect to find similar code wherever I/O is done.

2. FORMAT statements and any other text in the user's program, when typed out in instruction mode will appear as strange looking instructions with unusual address as shown at 60100P+1. Note the jump instruction around the FORMAT (JRST 2M), and also note the use of the '$T; ' command to temporarily change the type-out mode to text at 60100P+1. This mode was continued for one word with a line feed, after which a carriage return returned us to the original mode.

3. The instruction labelled 2M is in fact the first instruction of the statement numbered 110, and if the user typed '110P', IDDT would have opened the same cell. This illustrates that there may be several labels which all refer to the same location (the user's plus additional labels generated by the compiler). This also occurs with variables at times, where the compiler may assign a name like '%TEMP.' to a variable which the user has called 'B', for example. If that had been

the case here, then the instruction at 2M+4 might have typed out as ´MOVEM 2,%TEMP.´ instead of ´MOVEM 2,B´. ´%TEMP.´ and ´B´ would then represent the same location, and typing out either would show the same contents.

4. Subroutine calls are of the form ´JSA AC,ADR´ as shown at 120P The JSA subroutine jump instruction saves the AC (accumulator) at the ADDRess, and jumps to the ADR+1 to start the subroutine. thus, the first executable instruction of out subroutine is not at SUB but is at SUB+1. The first few instructions of the subroutine do initialization and are not usually of interest to the user. After the initialization there will be a jump to a label 1M which usually labels the first executable FORTRAN statement of the subroutine.

   Following the JSA instruction is the parameter list where each parameter´s address is part of a JUMP instruction. These JUMP instructions are never executed because the subroutine will return to the instruction following the last JUMP. This shows why variable length calling sequences to subprograms may not be used with DEC FORTRAN since the return point is always based on the number of parameters in the SUBROUTINE or FUNCTION statement, not on the number in the call.

5. The first instruction of the main program is at label MAIN. This is a system call for initialization, followed by a jump instruction to the first executable FORTRAN statement (the JRST 1M instruction). This first statement will always have the label 1M for all programs (main, subroutine and function), unless the program starts with arithmetic statement functions. In that case, the first executable statement may be at 2M, or 3M, etc. It can be found either by typing out instructions starting from MAIN. (or from the subprogram name) looking for the JRST, or by typing out instructions starting at label 1M until the first executable statement is recognized.

6. Following the code is space for constants and variables. Note the use of the ´$F; ´ command to examine the constant at CONST.

We now give some of the breakpoint and other control commands, and then describe with reference to the example how they might be used. A breakpoint is a way of stopping a program at a particular location. When the execution reaches the breakpointed location, IDDT regains control (before the instruction at the breakpoint is executed), and types something of the form

```
        $nB>>adr
```

where ´n´ is a number from 1 to 8 and ´adr´ is the address of the breakpoint. The user may have up to 8 breakpoints at any one time.

```
# Commands:
#
#    adr$b       to set the next  unused  breakpoint  (from  1  to  8)  at
#                location ´adr´.
#
#    x,,adr$B    to  set   the   next   unused   breakpoint   which   will
#                automatically  type  out  the contents of location x when
#                the breakpoint is reached.
#
#    Ø$nB        to remove specific breakpoint n.
#
#    $B          to remove all breakpoints.
#
#    $P          to proceed from a breakpoint - that is, after stopping at
#                a breakpoint (and possibly examining cells, setting other
#                breakpoints , removing breakpoints, etc.), $P causes IDDT
#                to  restart  the  program  with  the instruction that was
#                broken.
#
#    n$P         to proceed from the current breakpoint, and not  stop  at
#                that  breakpoint the next ´n´ times it is reached, but to
#                then break on the n+1´st time the breakpoint is  reached.
#                Note  that unless the number n contains an 8 or a 9 digit
#                or a decimal point that it will be interpreted as  octal.
#                Thus,  2Ø$P  means to proceed past the current breakpoint
#                2Ø octal  =  16  decimal  times,  while  2Ø.$P  means  2Ø
#                decimal.
#
#    adr$G       to start the program at location adr.  after a breakpoint
#                it  is usually safe to restart the program at a different
#                place (instead of proceeding) if  the  new  statement  is
#                labelled  with  a  FORTRAN  statement  number.   Starting
#                within a statement will seldom be desirable and  starting
#                at  an  unnumbered  statement  may occasionally fail if a
#                previous  statement  is   needed   to   initialize   some
#                accumulator  or  temporary.  An example of starting at a
#                new numbered statement is 11ØP$G.
#
# Examples of Use of the Commands with Reference to the  Sample  Program
# Above:
#
#       1.        @IDDT
#
#                 ;Yank file: PROG.SAV$ ret
#
#                 MAIN.$:      11ØP$B      $$G
#
#             The user GETs the program, enters IDDT, establishes the main
#             program  as  the  context  for  names,  sets a breakpoint at
#             FORTRAN statement 11Ø (equivalent to label 2M  in  the  IDDT
#             output), and starts the program at the beginning.
```

The program will first request the input of the A array and B, and after the user has typed in the data, will proceed to the breakpoint and stop by typing:

```
$1B>>2M
```

2. At this point the user may want to insert a breakpoint at the statement 'B=1.2', but this statement does not have a FORTRAN statement number so the location of its first instruction is not known. The easiest course of action is to start typing out instructions starting at 110P (the nearest preceding statement with a number) until the start of the 'B=1.2' statement is recognized. The type-out is done by first typing '110P/' and then continuing with line feeds (alternatively, the user could start at 120P and go backwards with successive up arrows = shift-N). Even though the user may not know PDP-10 assembly language, it is fairly easy to recognize the statement boundaries because the instructions contains references to the user's FORTRAN variables and statement numbers (but only if they were requested during loading with the /S switch as described above in the section Preparation for Debugging). In this case, the user may recognize the 'JRST 120P' as the 'GO TO 120' at the end of the previous statement, or the 'MOVEM 2,B' as the storage of B in the assignment statement to be broken. We see that the instruction to be broken is at 2M+3 and so the IDDT command 2M+3$B will insert the breakpoint as desired.

3. It is frequently useful to stop the program as soon as it has entered a subroutine. This may be conveniently done by:

```
SUB$:       1M$B
```

The first command establishes the context for names as that of the subprogram (here named SUB), and the second command inserts the breakpoint at the first executable statement of the subprogram at label 1M (but see note 5 above for exceptions to the 1M label). This will be after the parameters have been set up during subprogram initialization, and so when the breakpoint is reached, the subprogram's parameters may be examined.

4. Although it is possible to set and remove specific breakpoints, the novice may find it easy to start by simply setting the next unused breakpoint (with the adr$B command), and when all eight breakpoints have been used (IDDT will give a message when the user attempts to set a ninth breakpoint), to remove them all with the $B command and then reset any that are still desirable.

FORTRAN-IV Library Subroutines.

MODIFICATIONS TO THE FORTRAN LIBRARY

1. IFILE and OFILE include the addition of an optional extension. The formats for these subroutines now are:

```
#       CALL IFILE(device,filename,extension)
#       CALL OFILE(device,filename,extension)
```

where device	=	file device number (required)
filename	=	ASCII filename of up to 5 ASCII characters (required)
extension	=	ASCII extension of up to 3 ASCII characters. If set to Ø for IFILE, system will search for DAT extension first and then blank extension. If set to Ø for OFILE, system will set extension to DAT. (optional only if a user-code is not to be specified.)

2. The format for DEFINE has been modified from

```
CALL DEFINE FILE(U,S,V,F,PJ,PG,CODE)
```

to

```
CALL DEFINE FILE(U,S,V,F,USER,CODE)
```

# where USER is Ø and only required if it is to be followed by
# CODE which is an optional protection code field.

3. The CalComp Plotter routines have been modified for operation on the TENEX CalComp Plotter, Model 665 which uses 12" wide paper and has 2 step sizes.

CalComp subroutines enable plotting of lines, curves, text, and graphs by calls from FORTRAN programs. The subroutines also control pen position, labeling of plots, scaling, and data identification. Detailed subroutine discussions follow standard FORTRAN notation conventions so that variables beginning with I,J,K.L,M, or N, are fixed point while all others are floating point.

OPEN PLT: causes about 6" of paper to be slewed. The system routines maintain a "fence" that prevents a user from going off the edge of the paper. The pen is assumed to be centered in the Y direction when PLT: is opened, and users should leave the pen in a similar place when PLT: is closed.

The plotter accepts 5-bit bytes to direct its motion as shown below:

```
                 +Y
 7      33      Ø       32      1               PEN UP=11
 31     27      2Ø      21      3Ø              PEN DOWN=12
 6      26      .       22      2       +X      OTHERS ARE NOP´S
 35     25      24      23      34
 5      37      4       36      3
```

Moving in the +X direction slews paper on to the floor. The +Y direction is arranged so that the coordinate system is right handed.

PLOTTING CONTROL

PLOTS

CALL PLOTS(I)	OPENS the "Hardware" file PLT: If successful, I is set to Ø (a logical TRUE). Otherwise, I is set to -1. Output goes directly to the plotter.
CALL PLOTS(I, ´FILE NAME´)	Uses the named file instead of PLT: Any legal TENEX file name will do. I is returned as described above.
CALL PLOTS(I,Ø)	As above but the file name is requested from the Teletype similar to what TECO does with ;U$.
CALL PLOTS (´FILE NAME´)	No value returned

In addition each of the above has an integer function counterpart which returns the success/fail value to IPLOTS as well as the argument I.

NOTE: Call PLOTS before any other plotting subroutine and use it once and only once in a plotting program. It prepares the system to do plotting, setting its current position as the origin (Ø,Ø) and raising the pen from the paper.

PLOT

CALL PLOT (X,Y,IPEN)

X,Y	moves the pen in a straight line from its current position to the position specified by the floating point coordinates X,Y, in inches.
IPEN	specifies pen position during motion: 1 - unchanged from previous action 2 - lowers pen before motion so a line is drawn 3 - raises pen before motion -1,-2,-3 - same action as for positive values, except that after motion is completed, the pen position is taken as a new origin.

Call PLOT after completing all plotting in order to move the pen past the last plot.

WHERE

CALL WHERE (X,Y)	X,Y set to the current position of the pen, in inches, and returns these values to the program.

PLTEND

CALL PLTEND	Call PLTEND after completing all plotting to close the plotter output file.

LABELING

SYMBOL

CALL SYMBOL (X,Y,HEIGHT,CHAR,THETA,NS)

X,Y	coordinates of the lower lefthand corner of the first alphanumeric character.
HEIGHT	height of the character(s) in floating point inches
CHAR	an array containing a Hollerith string of letters, numbers of special characters.
THETA	direction, in degrees, of the base-line on which the text is plotted; normal orientation, X direction, is Ø.Ø.
NS	length of the string CHAR in characters.

NUMBER

CALL NUMBER (X,Y,HEIGHT,FPN,THETA,NN)

FPN	floating point number to be plotted; assumes the ASCII character set using the digits Ø through 9, "-" and ".".
NN	the number of decimal digits to be plotted to the right of the decimal point; a negative value suppresses the decimal point, printing only the integer part of the number.

SCALING

SCALE

CALL SCALE (X,N,AL,XMIN,DX)

X	a real vector containing the data to be plotted.
N	length or number of points in

	the array
AL	axis length in floating point inches
XMIN	set to the minimum value found in X, truncated to be multiple of DX
DX	set to the X-increment, a "reasonable" interval allowing X to be plotted in S inches and still be read, equal to A*1Ø**B, where A and B are integers and A is either 1,2,4,5 or 8.

Scans the data to be plotted for the maximum and minimum values, adjusting these to optimize the plot and determine scale values that are easy to interpolate between divisions.

DATA IDENTIFICATION

AXIS

```
CALL AXIS (XO,YO,TITLE,NC,AL,THETA,XMIN,DX)
```

XO,YO	the coordinates of the starting point of the axis, relative to the current origin.
TITLE	name of an alphabetic array for the axis title, may be of the form NH THIS IS AN N CHARACTER TITLE
NC	number of characters in the title; standard position is the counterclock-wise side of the axis, a negative NC places the title on the clockwise side

```
NORMAL      NEGATIVE NC
I           I
I           I
I           I
I           I
I TITLE     I----------
I--------     TITLE
```

AL	length of the axis in inches

THETA	angle, in degrees, of the axis; standard X-axis is 0, standard Y-axis is 90
XMIN	minimum value on the axis, determined by SCALE
DX	increment size, determined by SCALE

Draws a line with tic marks and scale values at one-inch intervals labeling the axis.

LINE OR CURVE GENERATION

LINE

CALL LINE (X,Y,N,K)

X	array of data for standard X direction
Y	array of data for standard Y direction
N	number of points to be plotted
K	skip factor, usually set to 1; for K>1, every Kth point is plotted.

Draws a line (or curve) through the specified points.

ADDITIONS TO THE FORTRAN LIBRARY

1. ACPT -- accept an arbitrary number of characters from the controlling teletype, float each ASCII code character and dump into an equal number of variables. The calling sequence is:

   CALL ACPT(CH1,CH2,...,CHn) where CH1,CH2,...,CHn are the variables into which the results are to be dumped.

2. ATN -- Arctangent Function which may be called with one or two floating point arguments. The subroutine decides whether it has one or two arguments and then calls the appropriate routine (ATAN or ATAN2). The format is:

   ATN(X) or ATN(X,Y)

3. COT -- Cotangent Function which computes COT(X) by the simple expedient of COT(X)=COS(X)/SIN(X). The format is:

   COT(X)

4. CPUTIM -- Subroutine to return the job CPU time in seconds. The calling sequence is:

   CALL CPUTIM(X)

5. CRAND -- Complex Random Number Generator which generates a complex random number between two given complex limits. The format is:

   CRAND((A,B),(C,D)) where (A,B) is the lower limit and (C,D) is the upper limit

6. DRAND -- Double Precision Random Number Generator which generates a double precision random number between two given double precision limits. The format is:

   DRAND(A,B) where A is the lower limit and B is the upper limit.

7. DTE -- Subroutine to return the real result of the current date in the form MMDDYY. The calling sequence is:

   CALL DTE(X)

8. FILES -- A collection of file handling utility subroutines which include DELETE, FILOOK, PROTEC, RENAM, UNPROT.

   a. DELETE -- perform a deletion on a specified filename.

      The calling sequence is:

      CALL DELETE(NAME,EXT,IANS)

      where:

      NAME -- file name of up to 5 characters
      EXT -- file extension of up to 3 characters
      IANS -- indicates the result of the deletion
          =0 -- Deletion successful
          =1 -- No such file
          =2 -- File open for output
          =3 -- Error in filename
          =4 -- Error during LOOKUP
          =5 -- Error in deletion

   b. FILOOK -- perform a LOOKUP on a specified filename.

The calling sequence is:

```
CALL FILOOK(NAME,EXT,IANS,IDCODE)
```

where:

```
NAME -- filename of up to 5 characters
EXT -- file extension of up to 3 characters
IANS -- indicates the  result of the LOOKUP
    =Ø -- LOOKUP successful
    =1 -- No such file
    =2 -- File read-protected or open for output
    =3 -- Error in filename
    =4 -- Error during LOOKUP
IDCODE -- Optional 6-character ASCII IDcode
```

c. PROTEC -- set the protection code on a specified filename so that only the owner can access, modify or delete the file. The calling sequence is:

```
CALL PROTEC(NAME,EXT,IANS)
```

where:

```
NAME -- filename of up to 5 characters
EXT -- file extension of up to 3 characters
IANS -- indicates the result of setting the protection
    =Ø -- Protection set successfully
    =1 -- No such file
    =2 -- File open for output
    =3 -- Error in filename
    =4 -- Error during LOOKUP
    =5 -- Error in setting protection
```

d. RENAM -- performs a RENAME operation. The calling sequence

```
CALL RENAM(NEWNAM,NEWEXT,OLDNAM,OLDEXT,IANS)
```

where:

```
NEWNAM,NEWEXT -- filename and extension of resultant
file
OLDNAM,OLDEXT -- filename and extension of original
 file
IANS -- indicates the result of the RENAME
    =Ø -- RENAME successful
    =1 -- No such old file
    =2 -- Old file read-protected or open for
          output
    =3 -- Error in filename (old or new)
```

=4 -- New name in use
=5 -- Error during RENAME

e. UNPROT -- set the protection code on a specified filename so that anyone can access the file, but only the owner can modify or delete it. The calling sequence is:

```
CALL UNPROT(NAME,EXT,IANS)
```

where:

```
NAME -- filename of up to 5 characters
EXT -- file extension of up to 3 characters
IANS -- indicates the result of unprotecting the
        file
    =Ø -- File unprotected successfully
    =1 -- No such file
    =2 -- File open for output
    =3 -- Error in filename
    =4 -- Error during LOOKUP
    =5 -- Error in unprotecting the file
```

9. FPLOT -- a collection of FORTRAN Subroutines for the Calcomp Plotter. All the subroutines use the FORTRAN PLOT Routine.

a. CIRCLE -- draws a circle, arc or spiral. The calling sequence is:

```
CALL CIRCL(X,Y,ANS,ANF,RS,RF,DI)
```

where:

```
X -- X-coordinate of the starting point
Y -- Y-coordinate of the starting point
ANS -- Starting angle of the radius vector
       relative to the X-axis
ANF -- Finishing angle of the radius vector
       relative to the X-axis
RS -- Length of the radius at ANS
RF -- Length of the radius at ANF
DI -- Either a Ø or 1;  where Ø means a solid line
      and 1 a dashed line
```

NOTE: The user can cause many types of curves to be drawn by changing various parameters.

RS=RF draws a circle
RS<RF draws a spiral

If RS and RF span zero, a curlycue shaped figure is drawn.

b. DASHL -- draw dashed lines connecting a series of data points. The calling sequence is:

```
CALL DASHL(X,Y,N,INC)
```

where:

X -- Name of the array containing X-coordinates of the points
Y -- Name of the array containing Y-coordinates of the points
N -- Total number of points in the X and Y arrays
INC -- Skip factor, i.e. set to 1 for all points, set to 2 for every second point.

NOTE: If using INC greater than 1 be sure to start at correct point. That is the arguments X and Y should be X(INC), Y(INC) respectively.

c. DASHP -- draw a dashed line to a specified point. The calling sequence is:

```
CALL DASHP(X,Y,DASH)
```

where:

X -- X-coordinate of point
Y -- Y-coordinate of point
DASH -- Length of dash (also space length).
If distance is less than dash length, DASH is automatically reset to 1/2 the distance.

d. ELIPS -- draw an ellipse or elliptical arc. The calling sequence is:

```
CALL ELIPS(X,Y,A,B,ANG,ANS,ANF,IPEN)
```

where:

X -- X-coordinate of the starting point
Y -- Y-coordinate of the starting point
A -- Length of semi-major axis
B -- Length of semi-minor axis
ANG -- Angle of semi-major axis with respect to the X-axis
ANS -- Starting angle of the radius vector relative to the X-axis
ANF -- Finishing angle of the radius vector

relative to the X-axis
IPEN -- Pen position for first move

3=Pen up
2=Pen down

e. GRID -- draw a linear grid. The calling sequence is:

```
CALL GRID(X,Y,DX,DY,NXS,NYS)
```

where:

X -- X-coordinate of lower left hand corner (origin)
Y -- Y-coordinate of lower left hand corner
DX -- Distance between lines parallel to Y-axis
DY -- Distance between lines parallel to X-axis
NXS -- Number of lines parallel to Y-axis
NYS -- Number of lines parallel to X-axis

NOTE: Two calls to GRID can be used to emphasize major division lines. Example: First call with spacing of .1 inches, second call with spacing every inch with (X,Y) offset slightly from that of first call.

f. POLY -- draw an equilateral polygon. The calling sequence is:

```
CALL POLY(X,Y,SL,SN,ANG)
```
where:

X -- X-coordinate of lower right hand corner of polygon
Y -- Y-coordinate of lower right hand corner of polygon
SL -- Side length
SN -- Number of sides
ANG -- Orientation angle of base with respect to X-axis and point (X,Y)

g. RECT -- draw a rectangle. The calling sequence is:

```
CALL RECT(X,Y,XW,YH,ANGLE,IPEN)
```

where:

X -- X-coordinate of the starting point
Y -- Y-coordinate of the starting point
XW -- Width of the rectangle
YH -- Height of the rectangle
ANGLE -- Angle
IPEN -- Pen position of first move

3=Pen up
2=Pen down

10. GETRAN -- get both halves of random number routine which gets the origin of the random number sequence and is used in conjunction with ZETRAN to allow a "random" number sequence to be repeated. The first random number generated after a call to GETRAN will be the same as the arguments. The calling sequence is:

    CALL GETRAN(X,Y) where X is the location to store the high part of the random number and Y is where to put the low part of the random number.

11. GRAND -- Gaussian Random Number Generator which uses the Gaussian method to generate a random number. The format is:

    GRAND(X,Y) where X is the mean and Y is the variance.

12. IRAND -- Integer Random Number Generator which gets a random integer uniformly distributed between two limits. The format is:

    IRAND(I,J) where I is the lower limit and J is the upper limit.

13. KJOB -- FORTRAN callable subroutine to logout the current job. The calling sequence is:

    CALL KJOB

14. RAND -- Random Bits Random Number Generator which generates a 36-bit integer random number. The result is returned in ACØ. The calling sequence is:

    CALL RAND

15. RANDOM -- Random Real Number Generator which generates a random real number uniform between two limits. The format is:

    RANDOM(A,B) where A is the lower limit and B is the upper limit.

16. REENTR -- FORTRAN callable subroutine which sets a reentry point within a FORTRAN program to which control should be returned by the monitor command REENTER after an interruption by a Control-C or an error. The calling sequence is:

    CALL REENTR

which sets the reentry point at the next sequential FORTRAN statement.

17. RN -- Random Real Number Generator which generates a random real number between 1.0 and 131071.0. The calling sequence is:

```
CALL RN(X)
```

# 18. SEC -- Subroutine to return the real result of the number of seconds since midnight. The calling sequence is:

```
CALL SEC(X)
```

# 19. SEND -- type out an arbitrary number of characters on the controlling teletypewriter which have been converted to ASCII teletype code from an equal number of fixed and/or floating point arguments. Floating point arguments are first converted to fixed. The calling sequence is:

CALL SEND (CH1,CH2,...,CHn) where CH1,CH2,...,CHn are the variables containing the arguments

# 20. SETDEV -- makes an entry in the FORTRAN Device Table so that references to device number IDEV refer to device name DEV. The calling sequence is:

```
CALL SETDEV('DEV',IDEV)
```

# 21. SHIFT -- shift a word a given number of bits. An optional Mask may also be specified. (SHIFT is convenient for character manipulation in FORTRAN.)

The calling sequence is:

```
CALL SHIFT(ARG,ARG2,ARG3,ARG4)
```

where:

```
ARG = Location of word to be shifted
ARG2 = Shift factor and direction of shift
       (A "+" indicates left and a "-" indicates right)
ARG3 = Destination location for shifted word
ARG4 = Optional Mask factor
```

# 22. SUBTMP -- contains the FORTRAN subroutines RDTMP and WRTMP
# for reading and writing temporary files. The calling sequences are:

```
CALL WRTMP(FILNAM,ADDR,LENGTH,IERROR)
```

CALL RDTMP(FILNAM,ADDR,LENGTH,IREAD,IERROR)

where:

FILNAM -- hollerith string or any variable containing the one to three character alphanumeric name of the TMP file

ADDR -- array name containing or receiving the contents of the TMP file

LENGTH -- number of elements in array to be transferred

IREAD -- optional return argument giving the actual length in words of the TMP file which was read. If the array is too small for the TMP file, as much of the file as can fit is transferred. If the file is shorter than the array, the rest of the array is filled with zeroes.

IERROR -- optional return argument giving the following error codes:

Ø -- no errors
1 -- FILNAM not alphanumeric
2 -- an argument was of the wrong type
3 -- length out of range
4 -- core had insufficient space and disk not available for TMP files
5 -- RDTMP: file not found
6 -- transmission error

# 23. TAN -- Tangent Function which computes TAN(X) by the simple expedient of TAN(X)=SIN(X)/COS(X). The format is:

TAN(X)

# 24. TIM -- Subroutine to return the real result of the time of
# day as HHMM. The calling sequence is:

CALL TIM(X)

# 25. ZETRAN -- sets the random number "initial value" and is used
# to set the origin of the random number sequence and in conjunction with GETRAN to allow a "random" number sequence to be repeated. The first random number generated after a call to GETRAN will be the same as the first random number generated after a call to ZETRAN with the same arguments. The calling sequence is:

CALL ZETRAN(X,Y) where X contains the high order part and Y contains the low order part

OTHER FORTRAN SUBROUTINES

See DECsystem1Ø FORTRAN-IV Manual for further information on subroutines.

FORTRAN Scientific Subroutine Package

The FORTRAN Scientific Subroutine Package is now available on the TENEX System. To incorporate routines from the Package into a FORTRAN Program, use the following command when loading with LOADER after the main program has been loaded and before the alt mode ($) key is struck:

```
     SYS:SSP/L
```

This will invoke a search of the Scientific Subroutine Package. The alt mode ($) invokes a search of the Standard FORTRAN Library, and also terminates the loading procedure.

A complete description of all the subroutines is contained in the IBM application Program Manual entitled, "System/36Ø Scientific Subroutine Package, (36ØA-CM-Ø3X) Version III, Programmer´s Manual". The IBM Manual Number is H2Ø-Ø2Ø5-4. Copies of the manual can be purchased from IBM and also from bookstores of many of the universities.

DATA SCREENING

TALLY - Totals, means, standard deviations, minimums, and maximums
BOUND - Selection of observations within bounds
SUBST - Subset selection from observation matrix
ABSNT - Detection of missing data
TAB1 - Tabulation of data (one variable)
TAB2 - Tabulation of data (two variables)
SUBMX - Building of subset matrix

CORRELATION AND REGRESSION

CORRE - Means, standard deviations, and correlations
MISR - Means, standard deviations, third and fourth moments, correlations, simple regression coefficients and their standard errors; considers that data may be missing
ORDER - Rearrangement of intercorrelations
MULTR - Multiple linear regression
GDATA - Data matrix generation for polynomial regression
STPRG - Stepwise multiple linear regression
PROBT - Probit analysis
CANOR - Canonical correlation

DESIGN ANALYSIS

AVDAT - Data storage allocation
AVCAL - Sigma and Delta operation
MEANQ - Mean square operation

DISCRIMINANT ANALYSIS

DMATX - Means and dispersion matrix
DISCR - Discriminant functions

TRACE - Cumulative percentage of eigenvalues
LOAD - Factor loading
VARMX - Varimax rotation

TIME SERIES

AUTO - Autocovariances
CROSS - Cross covariances
SMO - Application of filter coefficients (weights)
EXSMO - Triple exponential smoothing

NONPARAMETRIC STATISTICS

KOLMO - Kolmogorov-Smirnov one-sample test
KOLM2 - Kolmogorov-Smirnov two-sample test
SMIRN - Kolmogorov-Smirnov limiting distribution values
CHISQ - Chi-square test for contingency tables
KRANK - Kendall Rank correlation
MPAIR - Wilcoxin's signed ranks test
QTEST - Cochran Q-test
RANK - Rank observations
SIGNT - Sign test
SRANK - Spearman rank correlation
TIE - Calculation of ties in ranked observations
TWOAV - Friedman two-way analysis of variance statistic UTEST -
Mann-Whitney U-Test
WTEST - Kendall coefficient of concordance

GENERATION OF RANDOM VARIATES - DISTRIBUTION FUNCTIONS

NDTR - Normal Distribution function
BDTR - Beta distribution function
CDTR - Chi-square distribution function
NDTRI - Inverse of normal distribution function

ELEMENTARY STATISTICS AND MISCELLANY

MOMEN - First four moments
TTEST - Test on population means
BISER - Biserial correlation coefficient
PHI - PHI coefficient
POINT - Point-biserial correlation coefficient

TETRA - Tetrachoric correlation coefficient
SRATE - Survival rates

MATRICES: STORAGE

MCPY - Matrix copy
RCPY - Copy row of matrix into vector
CCPY - Copy column of matrix into vector
DCPY - Copy diagonal of matrix into vector
XCPY - Copy submatrix from given matrix
MSTR - Storage conversion
LOC - Location in compressed-stored matrix
CONVT - Single-precision/double-precision conversion
ARRAY - Vector storage/double-dimensioned storage conversion

MATRICES: OPERATIONS

GMADD - Add two general matrices
GMSUB - Subtract two general matrices
BMPRO - Product of two general matrices
GMTRA - Transpose of a general matrix
GTPRO - Transpose product of two general matrices
MADD - Add two matrices
MSUB - Subtract two matrices
MPRO - Matrix product (row into column)
MTRA - Transpose a matrix
TPRO - Transpose product
MATA - Transpose product of matrix by itself
SADD - Add scalar to matrix
SSUB - Subtract scalar from a matrix
SMPY - Matrix multiplied by a scalar
SDIV - Matrix divided by a scalar
SCLA - Matrix clear and add scalar
DCLA - Replace diagonal with scalar
RADD - Add row of one matrix to row of another matrix
CADD - Add column of one matrix to column of another matrix
SRMA - Scalar multiply row and add to another row
SCMA - Scalar multiply column and add to another column
RINT - Interchange two rows
CINT - Interchange two columns
RSUM - Sum the rows of a matrix
CSUM - Sum the columns of a matrix
RTAB - Tabulate the rows of a matrix
CTAB - Tabulate the columns of a matrix
RSRT - Sort matrix rows
CSRT - Sort matrix columns
RCUT - Partition by row
CCUT - Partition by column
RTIE - Adjoin two matrices by row
CTIE - Adjoin two matrices by column
MPRC,DMPRC - Permute rows or columns

MFUN - Matrix transformation by a function
RECP - Reciprocal function for MFUN

MATRICES: INVERSION, SYSTEMS OF LINEAR EQUATIONS & RELATED TOPICS

MINV - Matrix inversion
SINV, DSINV - Invert a symmetric positive definite matrix
SIMQ - Solution of simultaneous linear, algebraic equations
GELG, DGELG - System of general simultaneous linear equations by gauss elimination
RSLMC - Solution of simultaneous linear equations with iterative refinement
FACTR - Triangular factorization of a nonsingular matrix
MFGR,DMFGR - Matrix factorization and rank determination
GELS,DGELS - System of general simultaneous linear equations with symmetric coefficients
GELB,DGELB - System of general simultaneous linear equations with band structured coefficients
MTDS,DMTDS - Divide a matrix by a triangular matrix
MLSS,DMLSS - Solution of simultaneous linear equations with symmetric positive semidefinite matrix
MCHB,DMCHB - Triangular factorization of a symmetric positive definite band matrix
MFSS,DMFSS - Triangular factorization and rank determination of a symmetric positive semidefinite matrix
MFSD,DMFSD - Triangular factorization of a symmetric positive definite matrix
LLSQ,DLLSQ - Solution of linear least-squares problems

MATRICES: EIGENANALYSIS AND RELATED TOPICS

EIGEN - EIGENVALUES and EIGENVECTORS of a real, symmetric matrix
NROOT - Eigenvalues and eigenvectors of a special nonsymmetric matrix
ATEIG - Eigenvalues of a real almost triangular matrix
HSBG - Reduction of a real matrix to almost triangular form

POLYNOMIALS: OPERATIONS

PADD - Add two polynomials
PSUB - Subtract one polynomial from another
PMPY - Multiply two polynomials
PDIV - Divide one polynomial by another
PCLA - Replace one polynomial by another
PADDM - Multiply polynomial by constant and add to another polynomial
PVAL - Value of a polynomial

PVSUB - Substitute variable of polynomial
PILD - Evaluate polynomial and its first derivative
PDER - Derivative of a polynomial
PINT - Integral of a polynomial
PQSD - Quadratic synthetic division of a polynomial
PCLD - Complete linear synthetic division
PGCD - Greatest common divisor of two polynomials
PNORM - Normalize coefficient vector of polynomial
PECN,OPECN - Economization of a polynomial for symmetric range
PECS,DPECS - Economization of a polynomial for unsymmetric range

POLYNOMIALS: ROOTS

POLRT - Real and complex roots of a real polynomial
PRQD,DPRQD - Roots of a real polynomial by QD algorithm with displacement
PRBM,DPRBM - Roots of a real polynomial by Bairstow´s algorithm
PQFB,DPQFB - Determine a quadratic factor of a real polynomial

POLYNOMIALS: SPECIAL TYPES

CNP,DCNP - Value of N(th) Chebyshev polynomial
CNPS,DCNPS - Value of series expansion in Chebyshev polynomials
TCNP,DTCNP - Transform series expansion in Chebyshev polynomials to a polynomial
CSP,DCSP - Value of N(th) shifted Chebyshev polynomial
CSPS,DCSPS - Value of series expansion in shifted Chebyshev polynomials
TCSP,DTCSP - Transform series expansion in shifted Chebyshev polynomials to a polynomial
HEP,DHEP - Value of hermite polynomial
HEPS,DHEPS - Value of series expansion in hermite polynomials
THEP,OTHEP - Transform series expansion in hermite polynomials to a polynomial
LAP,DLAP - Value of Laguerre polynomial
LAPS,DLAPS - Value of series expansion in Laguerre polynomials
TLAP,DTLAP - Transform series expansion in Laguerre polynomials to a polynomial
LEP,DLEP - Value of Legendre polynomial
LEPS,DLEPS - Value of series expansion in Legendre polynomials
TLEP,DTLEP - Transform a series expansion in Legendre polynomials to a polynomial

ROOTS OF NONLINEAR EQUATIONS

RTWI,DRTWI - Refine estimate of root by Wegstein´s iteration
RTMI,DRTMI - Determine root within a range by Mueller´s iteration
RTNI,DRTNI - Refine estimate of root by Newton´s iteration

EXTREMUM OF FUNCTIONS

FMFP,DFMFP - Unconstrained minimum of a function of several variables -- Davidon method
FMCG,DFMCG - Unconstrained minimum of a function of several variables -- conjugate gradient method

PERMUTATIONS

PPRCN - Composition of permutations
PERM - Operations with permutations and transpositions

SEQUENCES: SUMS AND LIMITS

TEAS,DTEAS - Limit of a given sequence

INTERPOLATION, APPROXIMATION, AND SMOOTHING

ALI,DALI - Aitken-Lagrange Interpolation
AHI,DAHI - Aitken-Hermite Interpolation
ACFI,DACFI - Continued fraction interpolation
ATSG,DATSG - Table selection out of a general table
ATSM,DATSM - Table selection out of a monotonic table
ATSE,DATSE - Table selection out of an equidistant table
SG13,DSG13 - Local least-squares smoothing of tabulated functions

SE13,DSE13
SE15,DSE15
SE35,DSE35 - Local least-squares smoothing of equidistantly tabulated functions
APFS,DAPFS - Solve normal equations for least-squares fit
APCH,DAPCH - Least-squares polynomial approximation
ARAT,DARAT
FRAT,DFRAT - Rational least-squares approximation
APLL,DAPLL - Linear least-squares approximation
FORIF - Fourier analysis of a given function
FORIT - Fourier analysis of a tabulated function
HARM,DHARM - Complex three-dimensional fourier analysis
RHARM,DRHARM - Real one-dimensional Fourier Analysis
APMM,DAPMM - Linear Chebyshev approximation over a discrete range

NUMERICAL QUADRATURE

QTFG,DQTFG - Integration of monotonically tabulated function by trapezoidal rule
QTFE,DQTFE - Integration of equidistantly tabulated function by trapezoidal rule

QSF.DQSF - Integration of equidistantly tabulated function by Simpson's rule
QHFG,DQHFG - Integration of monotonically tabulated function with first derivative by Hermitian formula of first order
QHFE,DQHFE - Integration of equidistantly tabulated function with first derivative by Hermitian formula of first order
QHSG,DQHSK - Integration of monotonically tabulated function with first and second derivatives by Hermitian formula of first order
QHSE,DQHSE - Integration of equidistantly tabulated function with first and second derivatives by Hermetian formula of second order
QATR,DQATR - Integration of a given function by trapezoidal rule together with Romberg's extrapolation method
QG2,QG1Ø,DQG4-DQG32 - Integration of a given function by Gaussian Quadrature formulas
QL2-QL1Ø,DQL4-DQL32 - Integration of a given function by Gaussian-Laguerre quadrature formulas
QH2-QH1Ø,DQH8-DQH64 - Integration of a given function by Gaussian-Hermite quadrature formulas
QA2-QA1Ø,DQA4,DQA32 - Integration of a given function by associated Gaussian-Laguerre quadrature formulas

NUMERICAL DIFFERENTIATION

DGT3,DDGT3 - Differentiation of a tabulated function by parabolic interpolation
DET3,DDET3
DET5,DDET5 - Differentiation of an equidistantly tabulated function
DCAR,DDCAR - Derivative of a function at the center of an interval
DBAR,DDBAR - Derivative of a function at the border of an interval

ORDINARY DIFFERENTIAL EQUATIONS

RK1 - Solution of first-order differential equation by Runge-Kutta method
RK2 - Tabulated solution of first-order differential equation by Runge-Kutta method
RKGS,DRKGS - Solution of system of first-order ordinary differential equations with given initial values by the Runge-Kutta method
HPCG,DHPCG - Solution of general system of first-order ordinary differential equations with given initial values by Hamming's modified predictor-corrector method
HPCL,DHPCL - Solution of linear system of first-order ordinary differential equations with given initial values by Hamming's modified predictor-corrector method
LBVP,DLBVP - Solution of system of linear first-order ordinary

differential equations with linear boundary conditions by method of adjoint equations

SPECIAL FUNCTIONS

GMMMA - Gamma function
DLGAM - Log of Gamma function
BESJ - J Bessel function
BESY - Y Bessel function
BESI - I Bessel function I(Ø)
BESK - K Bessel function
EXPI - Exponential integral
SICI - Sine cosine integral
CS - Fresnel integrals
CEL1,DCEL1 - Complete elliptic integral of the first kind
CEL2,DCEL2 - Complete elliptic integral of the second kind
ELI1,DELI1 - Generalized elliptic integral of the first kind
ELI2,DELI2 - Generalized elliptic integral of the second kind
JELF,DJELF - Jacobian elliptic functions

EISPACK

EISPACK is the FORTRAN Eigensystem Subroutine Package which was developed at Argonne National Laboratory and is now available on BBN/TENEX. To incorporate routines from the package into a FORTRAN Program, use the following command when loading with LOADER after the main program has been loaded and before the altmode ($) key is struck:

```
SYS:EISPACK/L
```

This will invoke a search of the Eigensystem Subroutine Package. The altmode ($) invokes a search of the Standard FORTRAN Library, and also terminates the loading procedure.

A list of the routines in the Eigensystem Subroutine Package follows.

EIGENSYSTEM SUBROUTINE PACKAGE (EISPACK)

BALANC	BALANCES A REAL GENERAL MATRIX.
BALBAK	BACK TRANSFORMS THE EIGENVECTORS OF THAT REAL MATRIX TRANSFORMED BY BALANC.
CBAL	BALANCES A COMPLEX GENERAL MATRIX.
CBABK2	BACK TRANSFORMS THE EIGENVECTORS OF THAT COMPLEX MATRIX TRANSFORMED BY CBAL.
ELMHES	REDUCES A REAL GENERAL MATRIX TO UPPER HESSENBERG FORM

```
#            USING ELEMENTARY TRANSFORMATIONS.
#
# ELMBAK     BACK  TRANSFORMS  THE  EIGENVECTORS   OF   THAT   UPPER
#            HESSENBERG MATRIX DETERMINED BY ELMHES.
#
# ORTHES     REDUCES A REAL GENERAL MATRIX TO UPPER HESSENBERG  FORM
#            USING ORTHOGONAL TRANSFORMATIONS.
#
# ORTBAK     BACK  TRANSFORMS  THE  EIGENVECTORS   OF   THAT   UPPER
#            HESSENBERG MATRIX DETERMINED BY ORTHES.
#
# TRED1      REDUCES  A  REAL  SYMMETRIC  MATRIX  TO  A  SYMMETRIC
#            TRIDIAGONAL MATRIX USING ORTHOGONAL TRANSFORMATIONS.
#
# TRED2      REDUCES  A  REAL  SYMMETRIC  MATRIX  TO  A  SYMMETRIC
#            TRIDIAGONAL   MATRIX   ACCUMULATING   THE   ORTHOGONAL
#            TRANSFORMATIONS.
#
# TRBAK1     BACK TRANSFORMS  THE  EIGENVECTORS  OF  THAT  SYMMETRIC
#            TRIDIAGONAL MATRIX DETERMINED BY TRED1.
#
# FIGI       TRANSFORMS A  CERTAIN  REAL  NON-SYMMETRIC  TRIDIAGONAL
#            MATRIX TO A SYMMETRIC TRIDIAGONAL MATRIX.
#
# BAKVEC     BACK TRANSFORMS  THE  EIGENVECTORS  OF  THAT  SYMMETRIC
#            TRIDIAGONAL MATRIX DETERMINED BY FIGI.
#
# COMHES     REDUCES A  COMPLEX  GENERAL  MATRIX  TO  COMPLEX  UPPER
#            HESSENBERG FORM USING ELEMENTARY TRANSFORMATIONS.
#
# COMBAK     FORMS THE EIGENVECTORS OF A COMPLEX GENERAL MATRIX FROM
#            THE   EIGENVECTORS  OF  THAT  UPPER  HESSENBERG  MATRIX
#            DETERMINED BY COMHES.
#
# HTRIDI     REDUCES A COMPLEX HERMITIAN MATRIX TO A REAL  SYMMETRIC
#            TRIDIAGONAL MATRIX USING UNITARY TRANSFORMATIONS.
#
# HTRIBK     BACK TRANSFORMS  THE  EIGENVECTORS  OF  THAT  SYMMETRIC
#            TRIDIAGONAL MATRIX DETERMINED BY HTRIDI.
#
# HQR        DETERMINES THE EIGENVALUES OF A REAL  UPPER  HESSENBERG
#            MATRIX.
#
# HQR2       DETERMINES THE EIGENVALUES AND EIGENVECTORS OF  A  REAL
#            UPPER HESSENBERG MATRIX.
#
# INVIT      DETERMINES  THOSE  EIGENVECTORS   OF   A   REAL   UPPER
#            HESSENBERG   MATRIX   CORRESPONDING   TO   SPECIFIED
#            EIGENVALUES.
#
# TQL1       DETERMINES THE EIGENVALUES OF A  SYMMETRIC  TRIDIAGONAL
```

#		MATRIX.
#	TQL2	DETERMINES THE EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	IMTQL1	DETERMINES THE EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	IMTQL2	DETERMINES THE EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	TSTURM	DETERMINES SOME EIGENVALUES AND EIGENVECTORS OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	BISECT	DETERMINES SOME EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	COMLR	DETERMINES THE EIGENVALUES OF A COMPLEX UPPER HESSENBERG MATRIX.
#	COMLR2	DETERMINES THE EIGENVALUES AND EIGENVECTORS OF A COMPLEX UPPER HESSENBERG MATRIX.
#	CINVIT	DETERMINES THOSE EIGENVECTORS OF A COMPLEX UPPER HESSENBERG MATRIX CORRESPONDING TO SPECIFIED EIGENVALUES.
#	RATQR	DETERMINES SOME EXTREME EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX.
#	ELTRAN	ACCUMULATES THE TRANSFORMATIONS IN THE REDUCTION OF A REAL GENERAL MATRIX BY ELMHES.
#	ORTRAN	ACCUMULATES THE TRANSFORMATIONS IN THE REDUCTIONS OF A REAL GENERAL MATRIX BY ORTHES.
#	FIGI2	TRANSFORMS A CERTAIN REAL NON-SYMMETRIC TRIDIAGONAL MATRIX TO A SYMMETRIC TRIDIAGONAL MATRIX ACCUMULATING THE DIAGONAL TRANSFORMATIONS.
#	TINVIT	DETERMINES SOME EIGENVECTORS OF A SYMMETRIC TRIDIAGONAL MATRIX.

FRKCOM

The program FRKCOM allows the user to compare an address space with the address space of a file (such as a subsystem) whether or not the file has shared pages (BINCOM does the wrong thing on such files). FRKCOM is primarily useful for determining if the loading of a number of .REL files matches a SAVE or SSAVE file.

FRKCOM will complain if there are any significant differences in the spaces by telling the starting address where changes begin in a page, then hopping to the next page. No attempt is made to get things back into sync a la SRCCOM since this is virtually impossible with relocated binary code. Comparisons begin at location 140 and end at 775777(inclusive). The top two pages of the EXEC´s immediately inferior fork´s address space are used by FRKCOM itself. An example of FRKCOM´s use follows.

```
@GET <CHIPMAN>RUNOFF
@MERGE <SUBSYS>FRKCOM
@GO 777000

TYPE FILE TO COMPARE WITH FORK CONTENTS:  <SUBSYS>RUNOFF.SAV%

COMPARE FINISHED
@
```

FUDGE2

The FUDGE2 program is used to update files containing one or more relocatable binary programs and to manipulate programs within program files. Three files are used in the updating process.

1. A master file containing the file to be updated.

2. A transaction file containing the file of programs to be used when updating.

3. An output file containing the updated file.

All three files can be on the same device if the device is DSK. The two input files can be on the same DECtape.

The desired function of FUDGE2 is specified by a command code at the end of the command string. Only one command code can be specified in each command string. The command string is then terminated with an ESCApe (ALTMODE). Switches can also be used to manipulate file directories and to position a magnetic tape.

For further information on FUDGE2 please refer to DECsystem10 Assembly Language Handbook, UTILITY Section.

```
@FUDGE2

*output dev:file.ext=master dev:file.ext<programs>, transaction
dev:file.ext<programs> (command)$
```

output dev:	=	the device on which the updated file is written. If omitted, DSK is assumed.
master dev:	=	the device containing the file to be updated. If omitted, the default is DSK.
transaction dev:	=	the device containing the files of programs to be used in the updating process. When more than one file is transferred from magnetic tape or paper tape, a colon must follow the device name for each file. For example, MTA: : : Transfer 3 files If the device is omitted, DSK is assumed.
file.ext	=	the file name and extension of each file.

File names must be specified for directory devices, but the extension can be omitted. If the extension is not given, it is assumed to be .REL unless the /L switch appears in the command string. In this case, the output extension .LST is assumed.

Project-programmer numbers appearing after a file name apply to that file only. If the project-programmer number appears before the file name, it applies to all subsequent files until another device is specified.

<programs>	=	Names of programs (on DSK or DTA only) to be used in the updating process. They are grouped within angle brackets in the same order as they appear in the file and are separated by commas. When manipulating all the programs within a file, only the file name need be specified. Program names cannot appear for the output file.
(command)	=	Code for the function to be performed. This code can be either preceded by a slash or enclosed in parentheses and must appear at the end of the command string. Each command results in the updated file being output to the output device.

Switches are:

/A	append
/B	backspace one file
/C	copy and delete local symbols
/D	delete
/E	extract
/H	type this text
/I	insert
/K	skip one file
/L	list programs
/R	replace
/S	list entries
/T	skip to logical eot
/W	rewind
/X	index
/Z	zero directory

```
#                           GLOB
#
# GLOB reads multiple binary program files produced  by  MACRO  and
# FORTRAN  and generates an alphabetic cross-referenced list of all
# the global symbols encountered.  It  may  also  search  specified
# files  in  Library  Search Mode, checking for globals only if the
# program was loaded by the LOADER in  Library  Search  Mode.   For
# further  information  on  Glob read DECsystem1Ø Assembly Language
# Handbook, Utilities Section.
#
#
# Standard Command String Format:
#
# *outputfile=inputfile,inputfile,inputfile/switch/switch<Altmode>
#
# Switches:
#
# /A  include All global symbols in the GLOB listing (this is the
#  default)
# /E  include only Erroneous global symbols (undefined or multidefined)
#     in the GLOB listing
# /F  include only Fixed global symbols in the GLOB listing
# /H  type the Help message
# /L  Library search mode on
# /M  Library search mode off
# /N  include only Unreferenced global symbols in the GLOB listing
# /P  include multiple specifications in references lists
# /Q  exclude /P
# /R  include only Relocatable global symbols in the GLOB listing
# /S  include only multiply Specified global symbols in the GLOB listing
# /X  complement title switch
#
# Defaults:  DSK:.REL[self] on input
# DSK:same as infile.GLB[self] on output if no out, TTY:
#
# Flags used in listings:
#
# M   multiple definitions
# N   never referenced
# S   defined several times with the same value
# U   undefined
```

HERMES

The HERMES Message System is a computer program for sending and receiving messages over a computer network. The HERMES System has features that help the user read messages, compose messages for sending, and create and manage files of messages.

BASIC INSTRUCTIONS FOR USING THE HERMES SYSTEM

ENTERING AND LEAVING

Type HERMES<CR> to the TOPS-2Ø prompt "@". The HERMES Message System responds with the HERMES prompt ">", and surveys any messages that have arrived since your last session.

For example:

```
@HERMES<CR>
HERMES 4.Ø.22 1-OCT-77
-+    8   252  3Ø Sep 77 MYER at BBN-TENEXA  Plans for visit
```

When you want to leave HERMES, type:

```
>QUIT<CR>
```

If you have DELETED any messages, HERMES asks whether you want to EXPUNGE them. When messages are EXPUNGED, they are physically removed from the file, and the remaining messages are renumbered.

If you see the prompt ">>" or ">>>", you must first type "DONE">

```
>>DONE<CR>
>QUIT<CR>
```

After you QUIT, you may continue by typing CONTINUE to the @ prompt:

```
@CONTINUE<CR>
>
```

If you wish to logout directly from HERMES, type

```
     >LOGOUT<CR>
```

READING MESSAGES

HERMES tells you when you have messages that have arrived recently. HERMES prints a "survey" of each message, when you log in or when a new message arrives:

```
-+   1  536 15-Jul-77 MOOERS at BBN-TENEXA HERMES HELP INFORMATIO
```

For a quick survey of all messages in your current message-file,

```
     >SURVEY<CR>
```

To print the text of a single message:

```
     ><LF>     The LINEFEED key (NOT followed by <CR>)
                              prints the NEXT message, AND
                              sets  it to be the CURRENT message.
```

When a new message arrives, it is generally the NEXT message, unless you have been skipping around in your file.

```
     >^                       The UP-ARROW or CARET key (NOT
                              followed by  <CR>)  prints  the
     PREVIOUS
                              message  AND  sets  the  CURRENT
                              message.

     >PRINT<CR>               prints the CURRENT message.
```

READING SPECIFIC MESSAGES

```
>SURVEY 4,2,5:7<CR>        surveys messages 4,2,5,6,7.

>PRINT 4,2,5:7<CR>         prints the messages in the sequence
                           AND sets the CURRENT message to 7.
```

SENDING MESSAGES

To send a message, type "COMPOSE" to the HERMES prompt ">":

```
     >COMPOSE<CR>
```

COMPOSE gives you a series of prompts that guide you through composing a message. The To: and Cc: fields must be filled in with names of "directories" on BBN computers. The To, Cc and Subject fields end with <CR> but the Text field ends with <CTRL-Z>.

After you type <CTRL-Z>, the system asks

```
     SEND?:                    You may answer YES<CR> or NO<CR>.
```

If you answer YES<CR>, HERMES SENDS your message. If HERMES is able to deliver it immediately, it tells you that the copy of the message to an addressee is "delivered". If not, HERMES queues the message for another program,named MAILER, which picks the message up a few minutes later and delivers it. HERMES does not tell you which messages are queued.

If you type NO<CR>, HERMES does not SEND the message. Instead, you can do more work on the message. For example, you can add another name to the CC: field:

```
     >>CC: YOURNAME<CR>    Notice the double prompt ">>".
```

When you want to send the message, type SEND<CR> and "confirm" with a second <CR>.

```
     >>SEND<CR> (CONFIRM) <CR>
                                    Using the SEND command takes you
                             back to the ">" prompt.
```

If you change your mind about sending, after the computer prints (CONFIRM), you can stop it with <CTRL-U> on TOPS-2Ø.

REPLYING TO MESSAGES

To reply to a message in your message-file:

```
     >REPLY<CR>                gives you prompts for your reply to
                               back to the CURRENT message.
     >REPLY 5<CR>               replies to Message 5.
```

FORWARDING MESSAGES

To forward a message in your message-file to someone else:

```
     >FORWARD <CR>             forwards the CURRENT message, and
                               prompts you for addressees and
                               comments.
```

MESSAGE MANAGEMENT

To organize your messages, HERMES provides facilities for grouping messages into named sequences. A named sequence is an ordered set of messages. Individual messages can appear in as many different named sequences as you wish.

For example, to create a sequence named ANSWERED, you type:

```
     >CREATE SEQUENCE ANSWERED<CR>
```

This enters the sequence editor and HERMES prompts you with a double arrow. To add messages 3,4,5 and 7 to this sequence:

```
     >>ADD 3:5,7<CR>
     >>DONE
```

Once a sequence is created, you can add messages without entering the sequence editor. You use the ADD command and specify both the messages and the named sequence:

```
     >ADD 9 ANSWERED<CR>
```

To find out which sequences a message is in, use the WHEREIS command:

```
     >WHEREIS 9
     9 is in ANSWERED
```

To see a list of all messages in a sequence, for example in ANSWERED, type:

```
     >SHOW ANSWERED<CR>
     3:5,7,9
```

Sequences can be used in all message handling commands. For example, to survey all messages in ANSWERED, type:

```
     >SURVEY ANSWERED<CR>
```

To list all of them on the line-printer, type:

MESSAGE-FILES

The message-file you see when you enter the HERMES system is your INBOX, which has the file-name MESSAGE.

To direct the attention of the HERMES system to another message-file, type

```
          >GET <file-name><CR>
```

If your FILENAME-INPUT switch is set to HERMES, all files have single-word names, which are extended and recognized like other words in HERMES commands. If you have a file named NEWFILE, and no other files beginning with NEW, you can type:

```
     >Get NEW<CR> (CONFIRM)<CR>
```

To see a list of your message-files, type

```
     >Show MESSAGE-FILES<CR>
```

For more information, type

```
     >DESCRIBE FILENAME-INPUT-SWITCH<CR>.
```

PANIC BUTTONS

```
     <CTRL-O>   stops print-out of text anywhere; prints ^O.
     <RUBOUT/DELETE> TENEX: stops a command before the final
     <CR>.
     <CTRL-U>    TOPS-2Ø:  stops a command before the final <CR>.
     <CTRL-E>   stops a command at any time; prints ^E.
```

<CTRL-C>	stops HERMES anywhere and returns you to the operating system; prints ^C. You can recover by typing CONTINUE to the @ prompt.

To type a control character, e.g., <CTRL-C>, hold down the CTRL key while you type the letter C.

NOTE: Occasionally, by accident, you may get into a program that runs under HERMES in a "lower fork". Such a program will have a different prompt (such as "*"). In this situation, <CTRL-C> will return you to HERMES.

EDITING CHARACTERS

<CTRL-A>	TENEX: deletes a single character.
<RUBOUT/DELETE>	TOPS-20: deletes a single character.
<CTRL-W>	deletes a single word.
<CTRL-Q*>	deletes a line of text.
<CTRL-R>	retypes a line of text.
<CTRL-S*>	retypes an entire field.
<CTRL-Z>	ends the Text:-field; prints ^Z.

* On TOPS-20, in "terminal page mode", <CTRL-Q> freezes the screen of the scope and <CTRL-S releases the scope.

TYPING IN COMMANDS

You can use either the space character or the Escape-key (or ALT MODE) to separate the different sections within a command.

The Escape-key <ESC> causes HERMES to print out completely any word used in command and then to print "noise words" to introduce the next word, if it is possible to type one. At such a point, typing <ESC> causes HERMES to insert the "default" word. Most commands have a set of defaults, chosen to be most useful to the beginning user.

For example, PRINT prints the current message in the form specified by the PTEMPLATE onto your terminal (symbol TTY:).

If the current message is No. 1Ø, and you type "PR", followed by a series of <ESC>'s you will see:

```
>PRINT (messages) 1Ø (using template) PTEMPLATE (on file) TTY:
```

You may type "?" at almost any point in the HERMES system to see what words you are allowed to type in next.

```
     >?         gives a list of top-level HERMES commands.
     >SU?       lists commands that begin with "SU".
     >SURVEY ?  lists all the things that you may type in at
     that position in the "SURVEY" command.
```

Use ? and <ESC> freely.

```
          ?           tells what choices you have.
          <ESC>       tells what defaults the system has set up.
```

HERMES "LIFE-STYLES"

The FILENAME-INPUT switch controls whether file-handling is done entirely through the HERMES system or whether TENEX file-names and TENEX commands for showing and deleting files must be used.

The SPACE-FUNCTION switch and the CR-FUNCTION switch, change the way the space character and <CR> behave.

```
     >DESCRIBE LIFE-STYLES<CR>
                for more information.
```

NEWS, HELP AND SUGGESTION COMMANDS To print the latest News on your terminal, type NEWS.

```
     >NEWS<CR>
```

You can also output News or Help to the line-printer or a file.

```
     >NEWS LPT:<CR>
     or
     >NEWS <file-name><CR>
     >HELP LPT:<CR>
     or
     >HELP <file-name><CR>
```

If you have suggestions or questions, please use the SUGGESTION command:

```
     >SUGGESTION<CR>
```

This command prompts you for Subject and Text, then creates a message addressed to the HERMES staff.

ON-LINE DOCUMENTATION The on-line documentation material is basically a reference manual arranged as short topics with associated examples. The topics are organized in a table of contents or OUTLINE.

```
     >OUTLINE<CR>
```

shows the first two layers of topic names, and is equivalent to the command

```
     >OUTLINE HERMES (to depth) 2<CR>
```

You may use the OUTLINE command with any topic, and with the depth argument set to 2, 3, 4 ... All.

To see the contents of a topic, type

```
          >DESCRIBE <topic><CR>
```

The topic defaults to "HERMES" at top command level, to "MESSAGE-EDITOR" when you are creating a message, and to other appropriate topics in other parts of the system. For more information, type

```
          >DESCRIBE DOCUMENTATION<CR>
```

HG Manual

HG is a simple message reading program. It is designed to allow manipulation of files of messages in an unobtrusive manner. The following is a summary of HG's commands. (As of 8-Aug-77)

An item list can be any of the following things:

n - where is a given item
n1,n2,n3 - where each nn is an item number (need not be increasing values
n1-n2 - where n2 > n1 (Note that n1:n2 will also work)
altmode or % - which is shorthand for the highest numbered item
Any combination of the above separated by commas. (n1,n2-n3,<altmode>)
carriage return - use the list that showed up with the last brief print out (1 line) of the messages.

ALWAYS-SHOW - Set a list of items to be always shown when HG starts up; as if they were new items. This is for messages that one might want to be reminded about.

BEFORE - Have the filter show only messages before a given date and time. The time may be omitted, 00:00 of the date will be assumed.

CC - Require a given name in the CC list. This is part of the global filter.

DELETE - Delete a specified list of items. Confirmation will be asked for when the list is given.

DON'T-ALWAYS-SHOW - Negates the setting of ALWAYS SHOW.

FILTER-STATUS - Current settings of the global filter. (BEFORE, CC, SINCE, FROM, SUBJECT, & TO settings).

FORWARD - Forward a list of items to a list of addressees. The following subcommands exist:

CC - Add to the current carbon copy list (even if previously null).

DELETE - Remove any addressees starting with the given string (partial string match) from the TO and/or CC lists.

DISPLAY - Display any or ALL of the various fields.

FROM - Set the "FROM" field to a specific name. The SENDER field remains the logged in user name.

QUIT - Exit, aborting what was in progress.

SUBJECT - Set the subject field to a desired string.

TEXT - Insert text that is to be inserted before the forwarded text. Z (control-Z) to end input, E (control-E) to abort.

TO - Add to the current list of addressees.

A carriage return to the "For>" prompt will cause "[Confirm]". Confirming with another carriage return will cause the message to be sent.

FROM - Have filter show items from a single user name. Enough characters to make the name unique is sufficient.

GET - Bring a specified message file into the working area. If the current message file has been changed (any deletes done), the user will be asked if the file should be updated before the new file is read in for use.

HELP - Help works one of two ways. HELP<cr> gives a short more general help message. HELP <command name> will give a help string specific to that command.

LIST - Give a brief list of the messages. A single line is printed for each item in the file. The filter is applied for this command & those items coming through the filter set the default item list. Each single line contains the item number, the size of the message (**** means > 9999 chars in the message) the date the message was received, the author of the message (unless the user authored it, in this case the first entry of the TO list is given), and as much of the subject line that will fit on the rest of the line. LIST's subcommands may be invoked by using a "," and carriage return to terminate the list command. Subcommands are:

LPT - Put the brief list out to the line printer. This is a short form for OUTPUT (to file) LPT:

OUTPUT - Put the brief list out to a specified file.

NO - Turns off various parts of the filter. (NO SUBJECT, NO BEFORE etc.)

NOT - Negates various commands; like PERPETUAL.

PERPETUAL - Sets a specified list of items to be UNDELETABLE. This perpetual setting is currently only honored by HG.

PROFILE - Allows the user to set various parameters that will be retained from session to session. They are as follows:

CC-LIST - Show the CC list when printing the message (read command)

COMBINE - Ties together the ALWAYS SHOW & PERPETUAL commands. This means that using one of these commands automatically invokes the other.

CONFIRM-UPDATES - Require that the user be asked if a file should be updated. This is asked if items have been deleted. HG will expunge these deleted items if YES is given in response the confirm request.

DELETE - Set default parameters for deletion as follows:

REPLY - Delete the message after a REPLY has been made.

WRITE - Delete the message after it has been written to a file.

DISPLAY-PROFILE - Prints the current settings of the various parameters settable in the profile. Also displays the last date & time various files were read with HG.

DON'T -

COMBINE - Don't tie the ALWAYS SHOW & PERPETUAL commands together.

CONFIRM-UPDATES - Don't request confirmation when an update of a message box is to be done.

DELETE - Negates the various settings of DELETE as follows:

REPLY - Don't delete messages after a REPLY (system default).

WRITE - Don't delete messages after writing them to a file (system default).

ERASE - Erase the screen (of scopes only) with the following string of characters. This allows a user to use the hardware erase feature in his scope.

FORM-FEED - Do a form feed before printing each message. This setting works only on scope terminals.

FROM - Sets the default FROM field. This string will be used whenever none is given when SENDing, REPLYing, or FORWARDing a message.

HG-NEWS - In the normal setting, new items of interest are shown to the user at start up. (Once only). These items are from the file <DOCUMENTATION>HG.CHANGES.

NO - Negates various settings of the profile.

PAUSE - Pause in READ subcommand mode after each message. (This is the default setting)

PROTECTION - Sets the default protection for files written by HG.

QUIT - Return to main command level.

REMOVE - Remove a file name/date & time entry from the profile. Note that this command re-numbers the entries.

TO-LIST - Show the TO list when printing the message (read command)

VERBOSE - Show all of the message during the read command.

QUIT - Exit from HG. If any changes have been made to the file (deletes) the user will be ask if the file should be updated. Respond with a Y or an N and confirm either with a carriage return.

READ - Reads a specified item list. If the user is in PAUSE mode after each message the user will have the following options:

ADDRESS-LIST - Print the TO LIST & CC LIST that the current message was sent to.

ALWAYS-SHOW - Mark this item to always be shown at start up.

CC-LIST - Print the CC LIST the message was sent to.

DELETE - Delete the current item.

DON'T - Negates ALWAYS SHOW or DELETE. Don't delete negates the delete on write & delete on reply settings. default or the DELETE command.

FORWARD - Forwards the current item to the supplied list of addressees. See the main command level documentation about FORWARD for subcommands.

LPT - Write a copy of the current message to the line printer. The DELETE settings are completely ignored on this command (DELETE can't ever happen). The LPT command also attempts to break lines longer than 72 characters so that they don't run off the print page.

NOT - Currently only NOT PERPETUAL. Makes the current item deletable.

PERPETUAL - Make the current item UNdeletable.

QUIT - Abort the rest of the read command.

REPLY - Reply to the current message. Fills in the TO, CC and subject fields with those of the message just read. It also places the string "In response to your message of <date & time>" into the body of the message. Subcommands are as follows:

CC - Add to the current carbon copy list of addressees.

DELETE - Delete any addressees starting with the given string (partial string match) from the TO and/or CC lists.

DISPLAY - Display the current state of the various fields or ALL.

FROM - Fills in the FROM field with the desired string.

QUIT - Exit from REPLY aborting what has been done.

SUBJECT - Override the default subject with the supplied one.

TEXT - Allows the text body of the REPLY to be typed in. B (Control B) maybe used to insert a file or invoke TECO. Z (control-Z) to end input, E (control-E) to abort).

TO - Add to the current list of addressees.

A carriage return will request a confirmation ([Confirm]) which if confirm with another carriage return will send the reply.

SEND - Compose and send a message. See main level commands for details on the commands to the "Send>" prompt.

STATUS - Gives the current status of this item.

TO-LIST - Print the TO list the current message was sent to.

UNDELETE - Undelete a list of items. Items can be undeleted up to the time an "update" is done on the mailbox.

WRITE - Write the current message to a specified file.

SAVE - Write all items to a specified file.

SEND - Compose and send a message. The commands to the "Send>" prompt are:

CC - Add to the current carbon copy list of addressees.

DELETE - Delete any addressees from the TO and/or CC lists that start with the specified string (partial matches).

DISPLAY - Display any of the various fields. ALL will display all of them.

FROM - Specify the FROM field. Used when one wishes to have "Jimmy Carter" rather than "CARTER@WASH" as the from field.

QUIT - Exit aborting anything that been done so far.

SUBJECT - Specify the subject field for this message.

TEXT - Type in the body of the message. B (Control-B) can be used to insert a file or invoke TECO. Z (control-Z) to end input, E (control-E) to abort.

TO - Add to the current list of addressees.

A carriage return will respond with "[Confirm]". Confirming with another carriage return will cause the message to be sent.

SINCE - Set the filter so that item after a given date an time are seen.

STATUS - Reports the current status of specified list of items.

SUBJECT - Set the filter so only items of a specified subject will be seen in the LIST command.

TO - Set the name to be required in the TO list by the global filter.

UNDELETE - Undelete a list of items.

VERBOSE - A temporary (current session only) setting that causes the entire message to be printed in the READ command. (Over-rides the current profile settings).

WRITE - Write an item list to a specified file. Terminating the file name with a comma will give the user the following options:

ABORT - Abort the write & return to command mode.

BEGIN - Start the write. Carriage return serves the same purpose.

DELETE - Causes each item to be deleted after writing it to the file.

SEPARATE - Put each item on its own piece of paper. Make sure you only use this when writing to the LPT: or a file you don't want to read with HG or similar program.

WRAP - Attempts to break lines longer than 72 characters so the end of long lines aren't lost on the LPT:. As with SEPARATE, use this only when writing to the LPT:.

HTYPE

HTYPE is a program for printing files on the XEROX 1700 printer. It utilizes reverse printing and other features of the 1700 printer to print text at a very high speed. Right justified text may be re-adjusted using the variable character pitch feature to improve the quality of such documents. HTYPE also includes a facility for producing multi-column formats.

Preparation of Documents for HTYPE

No special preparation of documents is necessary for using HTYPE unless certain special features of HTYPE are to be utilized. In the absence of such special needs, HTYPE will print documents as exact replicas of those documents as they would appear on an ordinary terminal. There are four special features which might want to be used. These are: rejustification, HTYPE parameter setting, special escape sequences for subscripting and so forth, and multi-column capability. Each of these is discussed below.

HTYPE has the capability of re-justifying text which has been previously justified by one of the RUNOFF programs. The method employed is to first remove the additional spaces which RUNOFF added to achieve right justification and then expand or compress all inter-word spaces equally so that the desired column width is achieved. Generally, the only special step needed in preparing a document which is to be rejustified is to set the right margin about 5 characters greater than the final width desired. For example, if the final width is 65 characters (6.5 inches), then use a right margin of 70. The extra width is desirable so that lines will be compressed as well as expanded so that the average character density is normal. The use of hyphenation might also be considered since excessive expansion may be necessary when long words fall at the end of lines.

The parameters which govern HTYPE's actions may be set from the .DOC file as well as from the terminal. Such parameters are flagged in the file by the appearance of an SOH character (control-A). Each parameter consists of a single letter followed by as many numbers as necessary to specify the parameter setting. Each parameter is terminated by either another SOH (if another parameter follows) or by a carriage-return or by an STX (control-B). Generally these parameters will appear at the very beginning of the file but may appear anywhere within the file.

In this case these parameter changes will take effect when that point is processed for printing. However, beware that if the page on which such parameters appear is not printed, then those parameter changes will not occur. A special case is made for the first page of the file. If that page begins with a SOH, then all the parameters following that SOH up to the following STX or FF (form-feed) are processed prior to entering the parameter setting dialog with the user. This happens whether the first page is printed or not. Also, if that first page consists of nothing but parameters then that page is not counted as a page; the following page is numbered "1".

The parameters which may be set are as follows ([] is used to indicate optional numbers):

J Turns off rejustification.

J[nn[f]] Sets rejustification to column nn with flag character f.

H nn sets the horizontal character spacing to nn.

V nn sets the vertical line spacing to nn.

P turns on pause mode.

N turns off pause mode.

M turns off multiple column mode.

M nn[mm[ll[kk[jj]]]] turns on multiple column feature with column spacing of nn, first line of mm, lines per column of ll, number of columns of kk and rejustification of jj. See below for further explanation of these parameters.

Special Escape Sequences

It is possible to include special sequences of characters in the .DOC file for HTYPE to process. These character sequences are enclosed between and initial DLE (control-P) and control-]. Generally, these sequences should have zero net width; that is, if they position the carriage horizontally, they should return the carriage to its starting point. Exceptions to this rule may be made if a subsequent such sequence on the same line undoes the horizontal movement. If this rule is violated, and rejustification is applied to the line, the result is unpredictable.

The multi-column feature is described below. Preparing a document with multiple columns is largely a trial and error procedure since the page length of each page must be altered to account for the number of lines on the page which are in multiple column format and the number which aren't.

Running HTYPE

To run HTYPE, type to the EXEC:

```
     @HTYPE<cr>
```

In order to determine the parameters needed to do its job, HTYPE interrogates the user with a series of questions. These questions are arranged in order of decreasing need of specification. To terminate the questioning, end the answer with a carriage return. HTYPE will immediately space to the top of the next page and wait for paper to be adjusted etc. Terminating the answer with a comma causes HTYPE to advance to the next question. Terminating with an uparrow (circumflex) goes back to the preceding question. If no answer precedes the carriage return, comma, or circumflex, the default setting (or previously specified setting) is kept. The default setting is printed in brackets at the end of the question.

HTYPE first asks for the name of the file to be printed.

```
     Input file: text.doc
```

This may be any disk file containing text. On TENEX, the default extension is .DOC. On TOPS2Ø, the extensions of .DOC, .TXT are searched for in that order. If no file with either extension is found, then no default is assumed which, if the name is unique, will find that file. Otherwise, the extension must be given.

Next, HTYPE asks for a list of pages to print. These should be specified as a list of page groups separated by commas. A page group is specified as **n:m** and means pages n through m inclusive. The symbol % may be used to indicate the last page of the file and the symbol * may be used to indicate all pages. **1:%** is equivalent to *. The default setting for this parameter is * or all pages. A typical specification might be:

```
     Pages[1:%] 1,3:5,%
```

If the number before the colon is greater than the number after the colon, the pages are listed from the higher number through the lower.

Next, you must say whether the text should be re-justified. If this question is answered **N**, then the resultant output will be an image of the file as if it had been copied to a normal terminal. Answering **Y**, causes text which was right justified by (M)RUNOFF(OUT) to be re-justified as explained above. This results in a much improved appearance. To avoid rejustifying lines which were not justified by RUNOFF, lines will not be rejustified which are shorter than the specified column, which contain tabs, which contain a sequence of more than 6 spaces, or in which the length of the minimum string of spaces is smaller than two less than the length of the maximum string of spaces. These checks serve to detect almost all of the cases where the line was not originally justified. There is the possibility, however, that text may be re-justified which was not justified originally and which is not detected by the above criteria. This is unavoidable.

Two parameters are associated with rejustification. The right margin must be specified. This is usually 65 but may vary from document to document. This parameter determines the location of the rightmost end of the output line. The location of the leftmost end is the same as it is in the file being printed. If a flag character is specified, then that character is considered a space for determining where a line begins and ends. The flag character is often vertical bar. Type just a terminator if there is no flag character. A typical specification for re-justification might be:

```
     Rejustify with variable spacing?[No] Yes,
     Location of right margin:[65] 68,
     Flag character: |,
```

Next you will be asked if you want to pause between pages. The default pause, is suitable for individual sheets of paper. No pause is suitable if continuous forms are being used.

Next, you can specify the horizontal character spacing. Horizontal spacing is often expressed as a pitch of so many characters per inch, for example, Elite 12 pitch font. However, the number given to HTYPE must be the actual character spacing in 1/120 of an inch increments. Thus for a 12 pitch font, you specify 120/12 = 10 as follows:

```
     Horizontal character spacing (1/120 of an inch)[12]: 10
```

For a 10 pitch PICA font, use 120/10 = 12 (this is the default). Larger numbers mean greater spacing of characters. Other values may be used in order to squeeze or expand the text horizontally.

Next, you can specify the vertical spacing. As for horizontal spacing, the number specified is the actual space between lines. In this case, it is expressed in 1/48 of an inch increments. A 10 pitch font usually has 6 lines per inch for single spaced copy. The number you would specify to HTYPE is 48/6 = 8. 8 lines per inch is standard for a 12 pitch print wheel. For this, you would specify 48/8 = 6 as follows:

```
     Vertical line spacing (1/48 of an inch)[8]: 6
```

Other spacings might be used for special purposes.
Next, you may enable the multi-column feature. When this feature is enabled, lines of text may be caused to appear in multiple columns. Text to be printed in this fashion should be prepared with an initial full width section which might contain page headers etc. followed by a reduced width section which will be printed in multiple columns followed by a final full width section. There are four parameters which control the multi-column feature. First is the column spacing which determines the distance (in characters) from the left edge of one column to the left edge of the next. The default value for this is 33 characters. Next is the first line on which multiple columns should begin. This is one greater than the number of lines which are printed full width. Next is the number of lines per column. If two columns are to be printed, then the text being printed should contain twice this number of narrow lines. Next is specified the number of columns. The default for this is two. Finally, the rejustification margin is specified for the multiple column region. This number serves the same purpose as the rejustification margin above except that it applies to the multiple column region of the file. After the final column is

finished, the left margin reverts to the original left side of the paper.
The remainder of HTYPE's operation is similar to that of ZTYPE. Note that the special graphics control characters are different for the 17ØØ than for the Bedford terminal. The appropriate version of the "SUBSUPER" file should be used.

The commands which may be typed at page pauses are summarized below. Note that echoes are turned off as these commands are typed to avoid the necessity of inserting a piece of scrap paper.

Pnnn<sp> Print page nnn next. The set of pages remaining to be printed is not altered and the page which would have been printed next will be printed after the page specified by the command.

X Exit to the exec.

B Jumps back to the beginning of the parameter setting dialog. Thus certain parameters may be altered without re-specifying the rest. The parameter setting dialog is not conducted invisibly.

Any other character indicates that the paper is ready and printing begins.

Two special characters are recognized while HTYPE is printing. Control-X aborts printing the current page and enters a page pause. Use this character if the page has been garbled somehow and you wish to start again. Note, that the next page printed will normally be the next one specified at the initial dialog. If the same page is to be printed, use the **Pnnn<sp>** command. Control-P may be typed to force a page pause at the end of the current page. This is useful when operating in no pause mode and you want to re-adjust the paper or reprint a garbled page.

GRIPE

GRIPE allows you to send complaints or suggestions about subsystems to system personnel.

When it asks you "Griping on subject of ", type the name of the subsystem. If it doesn´t accept that name, try "general" instead of the subsystem name.

When it asks you for "Message", type your message terminated by control-Z. For details on entering the message, see "Message" under SNDMSG in this manual.

IDDT

1. Introduction

IDDT is a debugger for TENEX programs. It has many of the same commands as the standard DDT10X (SDDT and UDDT) and ordinarily may be used without regard to the fact that it is a different debugger. The user is directed to the DDT section of the DECsystem-10 ASSEMBLY LANGUAGE HANDBOOK for information regarding the basic features and use of DDT.

The primary feature of IDDT is that it operates on user programs which run in an inferior fork under IDDT. Thus, an errant user program cannot destroy the debugger or its symbol table because the debugger is in a totally different address space. This relation between the program being debugged and IDDT is much the same as the relation between current user programs (including IDDT) and the EXEC. Because of this, IDDT must simulate many of the services ordinarily provided by the EXEC, such as @GET, @LOADER, @RUN, etc.

The following describes the new features in IDDT and how they may be used for debugging. Some of the features are bound to change, and others will be added.

2. Using IDDT

IDDT may be called into service either before or after programs have been loaded into memory. This is done by telling the EXEC

```
@IDDT
```

This command causes the EXEC to splice a fork containing IDDT in between itself and the program to be debugged. This operation is done in a way that preserves the state of the user's program including its fork structure. It is possible to ^C out of a running program and get IDDT. If this is done, a $P (Proceed) command will resume running the user program.

The EXEC command "NO IDDT" will unsplice the fork containing IDDT in the event the user wishes to continue his program without having an IDDT above it.

A fairly common practice is to get IDDT first and use it to load the program to be debugged. One of three IDDT commands may be used to load the object program: $L (run the LOADER in the user fork), ;L (Loadgo of named file), or ;Y Yank the named file). The first of these is essentially the same as the EXEC command, @LOADER. The second is comparable to @RUN, while the last is similar to @GET.

3. Symbol Table Considerations

When initially started, and after successful execution of a ;L or ;Y command, IDDT will obtain a new symbol table if it exists. It does this by copying (and sometimes sharing) pages of the user fork. Thus, those user programs which need access to their own symbols will behave the same, and IDDT will have its own copy of the symbol table which is protected from the user.

The $L command causes IDDT to run the LOADER in the user´s address space. Upon completion, the LOADER returns control to IDDT. At this point IDDT will have the LOADER´s symbol table. In order to switch to the symbols of the program which was loaded, the ;S command should be typed. ;S tells IDDT to look for a standard symbol table pointer in location 116 (.JBSYM). If the user has merged in a file which contains its own symbols, he may switch to that table by typing a;S where "a" is the address of the location containing a pointer to the new table.

# ;? with no argument causes IDDT to type the error string
associated with the most recent error in the user program fork.
# With an argument (600121;? or IOX5;?), the corresponding error
# string will be typed.

;O Obtains a symbol file directly into IDDT without modifying the user´s memory. The old symbol table is replaced, and a new entry vector is taken only if there was no old one. This makes it possible to debug one file with symbols obtained from a different file.

Symbols may be written out on a specified file by using the ;W command. This saves the symbols in a way that they may be obtained later with the ;O command. Along with the main symbol table, the undefined symbol table is saved in symbol files. One should be careful that symbol files and core image (;U) files are kept paired if any undefined symbols exist. Executing a GET JSYS on a symbol file will get both tables. The default file extension for symbol files written by IDDT is .SYMBOLS.

Note: Symbols added to or deleted from IDDT´s symbol table by commands to IDDT will not be seen by the user program.

4. EXEC-like Features

For convenience, the EXEC has several commands which provide the same services as some EXEC commands. These are:

;A	TYPES THE USER'S ADDRESS SPACE. (MEMSTAT)
;F	DOES A FORKSTAT ON THE USER FORK. (THIS FEATURE WILL BE OPERATIONAL AS SOON AS JSYS 166 IS IMPLEMENTED.)
;Y	@GET
;M	@MERGE
;L	@RUN
$$G	@START
$$1G	@REENTER
$$nG	@START AT n-TH ENTRY VECTOR LOCATION
$P	@CONTINUE
# $L	@LOADER
;U	@SSAV Ø 777 (I.E. "UNGET")
;O	OBTAIN SYMBOL FILE -- NO EXEC EQUIVALENT
;W	WRITE SYMBOL FILE -- NO EXEC EQUIVALENT
;H	@QUIT (HALT, RETURN TO EXEC)

;W, ;M, ;Y, ;O, ;L, and ;U ask for a file name from the user. The default extension will be .SAV or .SYMBOLS as required.

5. Access Control

An EXEC-like feature has been included in IDDT which has no analogy in the current EXEC. This is the $U command (UNPROTECT), which allows the user to manually change the protection on various pages of his fork. This command has several forms:

a<b$nU	CHANGE PROTECTION ON PAGES a THROUGH b INCLUSIVE
a$nU	CHANGE protection of page a
$nU	CHANGE PROTECTION OF THE CURRENT PAGE

(WHERE POINT "." IS)

N is always a three-bit number (Ø-7). The 4-bit means allow read access, the 2-bit should be on to allow write access, and the 1-bit for execute access. If N is not specified at all, it will be taken as 7. Thus, the command $U frees up the current page, giving it read, write and execute access.

$U changes the protection on pages of the user´s fork. It does not affect the protection of a file page which might be mapped into that fork. Because it is sometimes convenient to change the contents of fork pages which have write-protected files mapped into them, $U commands which ask for write access will either get it, or will get write-copy access.

While IDDT is running, it temporarily changes the access of each page that it maps to have read, write, and execute access. The user´s access is reset when the page is mapped out. This allows IDDT to insert breakpoints, retrieve trapping instructions, etc. The $U command allows the user to protect pages from his own program by effecting a permanent change in the page access.

6. Rubout

IDDT arms the RUBOUT button as an interrupt character. If a user program has been started under IDDT, pressing RUBOUT will gracefully suspend that process and give control to IDDT which then types a message of the form

```
XXX:FOO+5/   MOVE A,DAT+21
```

The interrupt is understood to have occurred immediately before this instruction, and that if a $P (proceed) command is typed, this instruction will be the next one executed by the user.

RUBOUT´s typed while in IDDT behave much the same as they do in normal DDT´s. That is, the current command is aborted. This is particularly convenient for stopping long searches ($W, $N and $E COMMANDS), with IDDT because it is an interrupt and does not have to be read by a PBIN to initiate action as it does with old-style DDT´s. RUBOUT´s typed while IDDT is in control cause the terminal output buffer to be cleared.

# Since some programs such as TECO use RUBOUT as a command,
the IDDT ;E command can be used to change the IDDT interrupt
# character from RUBOUT to any other control character.

7. "GO" Commands

IDDT has several variations of the standard $G command available. GO commands with two ALTMODES, such as $$G, FOO$$G, and $$2G, cause the user's pseudo interrupt system to be cleared before they take effect. If there is a number between the ALTMODE(s) and the G, this number is taken as an index into the entry vector of the user's fork, and the program is restarted as indicated by the corresponding entry vector element. Thus, $$ØG is the same as the EXEC command "START", while $$1G is equivalent to "REENTER". The command $$G is an abbreviation for $$ØG.

Ordinary GO commands still exist. They look like FOO$G and BEGIN$$G. The user's program counter is stored in the "GO" register, which is named $G. This can be examined by commands such as $G/ .

8. Control-T

# Frequently the user would like to know whether his program
# is making progress. To facilitate this, the EXEC arms ^T
# (control-T) as an interrupt character, which types the program
# state, the load average, amount of CPU and CONSOLE time used (in
# seconds), and the ratio of console to CPU time (the "activity
# ratio"). Note that the user's program is not stopped while his
# information is being typed and if his program is typing out, ^T
# will result in a garbled typescript.

9. Interface with the EXEC

The EXEC command "FORK n" may be used to switch the EXEC's attention between the fork containing IDDT and the one containing the user's program. This may be done for the purpose of doing a "MEMSTAT" or ^T. The EXEC examine and deposit commands (/ and \) also pertain to the currently selected fork.

Regardless of which fork has been selected, a "CONTINUE" will always resume a ^C . If the user has returned to the EXEC by typing ;H to IDDT, IDDT may be resumed by a "CONTINUE". A HALTF in the user's program will return to IDDT. It may be continued by a $P to IDDT.

1Ø. Zero-ing core

THE $$Z COMMAND behaves the same as it does with old DDT except that if it is used to zero whole pages, they are PMAP-ed out of existence, rather than being actually cleared. If such a page is brought into existence again by a reference, it will be cleared by TENEX when created.

If a $$Z command is used to clear any word(s) between 700000 and 712777, compatibility code for the user is dismissed. Ordinary register operations like slash can be used to examine or modify the compatibility code (PA1050) as usual.

The Zero command has been generalized so that it can fill
# core with a specific decimal number. To fill locations 100
# through 177 with the number 12 (decimal) the user would type
# 100<177$$12Z.

11. Internal Registers

IDDT maintains several "internal registers" which may be manipulated as if they were in the user's address space. These are listed below, and will be described in detail in subsequent sections.

$G	CONTAINS FLAGS,,PC FOR THE USER PROGRAM
$M	MASK FOR SEARCHES
$X	LOCATION FOR SPECIAL EXECUTE
$W	PAGER TRAP STATUS WORD AT MEMORY VIOLATION
$W+1	PAGER WRITE DATA AT MEMORY VIOLATION
$I	INTERRUPT CHANNELS WITH BREAKS WAITING
$I+1	INTERRUPT CHANNELS ASSIGNED FOR USER
$I+2	BREAKS IN PROGRESS WORD
$I+3	0 IF USER'S INTERRUPT SYSTEM IS OFF. NON-0 OTHERWISE.
$I+4	IDDT'S FORK HANDLE ON USER
$I+5	SIXBIT OF SAVED USER SUBSYSTEM NAME (This may be made inaccessible in the future!)
$nB+k	BREAKPOINT REGISTERS. n is between 1 and 8 inclusive (i.e., IDDT has eight breakpoints), k is between 0 and 6. Thus there are seven registers of information associated with each of the breakpoints.

As an example of an internal register reference, consider looking at the proceed count of breakpoint 3:

```
$3B+2/ 105 3
```

The user changed the proceed count from 1Ø5 to 3.

IDDT´s current location may be internal to IDDT. This allows the user to use linefeed and up-arrow to look at internal registers. IDDT has special address printing routines that print things like $I+3 instead of this address in terms of user defined symbols.

Attempts to define address tags when "point" is at an IDDT internal register will be given IDDT´s ubiquitous "?" error. Also, IDDT will not allow expressions with more than one mention of an internal symbol name. Thus, $M+3 is allowed, but $I+$M is not.

12. The User Program PC

The internal register $G contains the user´s PC and FLAGS. This is defined to always point at the next executable instruction. The proceed command ($P) simply starts the user at the address in $G. Illegal instruction traps back up the user´s PC so that it points at the offending instruction, in hopes that he will repair it and proceed. In such a case, the repaired instruction will be executed first.

$G is setup from the entry vector after every ;Y, ;L, and ;S command. Thus, the user can ;Y (yank) a file and immediately start it with a $P.

Bit 5 of the "GO" word $G is the user-mode bit which will normally be on if $G is examined. It may be off due to an interrupt out of a JSYS or after an illegal instruction. Because this bit is essential to the restarting of the user´s fork, it is not left entirely under his control. In particular, the user-mode flag may be turned on by changing the contents of $G, but it may not be turned off. This means that if a JSYS (such as GTJFN) has been interrupted, the usermode flag turned on, and $P typed, that the interrupted JSYS will be re-executed, rather than resumed.

13. Saving a Core Image

The ;U command asks for a file name and then does an SSAVE from page Ø through page 777 on this file. The entry vector will be copied if it exists. If no entry vector has been declared for the fork, IDDT will set a length one entry vector at "." . A message is typed to this effect.

14. Single Instruction Executes

When the user types an instruction followed by $X, IDDT pushes down several words of state information, plants the instruction in the user´s address space followed by three breakpoints, and restarts the user at this special location. When the instruction completes, IDDT types the proper number of $-signs to indicate how many times the instruction skipped, and pops back the saved state information. The state information currently includes the program counter ($G), and which breakpoint (if any) the user was stopped at. This makes it possible to hit a breakpoint, execute an instruction (which might be a PUSHJ to a subroutine), and then, upon completion of the $X, do a $P to proceed the breakpoint.

IDDT´s $X/ register points in the user´s address space to the four words which will be used for $X commands. $X initially contains 777774 so that the top four words are used. The user is free to change the contents of $X.

If a RUBOUT has interrupted the program being debugged while it was in the middle of a JSYS -- usually a "long" JSYS like SOUT or PBIN -- and then an instruction executed with the $X command, a $P will not resume the original sequence back in the middle of the interrupted JSYS. Flag bit 5 will be off if the interrupt came out of a JSYS. A proceed ($P) immediately after a RUBOUT, with no intermediate $X will always resume exactly at the interrupt point however.

15. Breakpoints

Associated with each breakpoint are seven internal registers. The first four of these are the same as those in older DDT´s, while the last three have been added.

#	$nB/	TRACEVALUE,,LOCATION
#		
#	$nB+1/	Ø OR CONDITIONAL BREAK SKIP
#		
#	$nB+2/	PROCEED COUNT (>Ø for normal, <Ø for auto, Ø for none)
#		
#	$nB+3/	Ø OR STRING POINTER (fed to IDDT when this BPT breaks)
#		***Not implemented yet***
#		
#	$nB+4/	SAVED INSTRUCTION WHILE USER IS RUNNING
#		
#	$nB+5/	Ø OR ELSE -1 FOR AUTOPROCEED MODE
#		
#	$nB+6/	ASCII NAME OF THIS BREAKPOINT, USUALLY "$nB"

Usually these values are changed only by setting and clearing breakpoints with the $B command. if he wishes, the user may change these quantities. For instance, if he wants hits on breakpoint three to print as

```
            HELP>> FOO+23
```

he would type the following:

```
            $3B+6!  "/HELP/
```

This stores the ASCII string for the new name in the print name cell of breakpoint three.

Proceeding after a breakpoint hit happens much in the same way as a single instruction execute command ($X). Again four words of memory are written into. However, in this case the four words are the instruction at the break location and three JRST´s to the three locations following the break location. The JRST´s account for possible skips by the break instruction.

If a breakpoint is hit, and the user changes the contents of $G, and then proceeds (with $P), the break instruction is not executed. Control simply resumes at the new location given by $G. Old DDT´s execute the instruction under the breakpoint, and then transfer control to the new place.

16. JSYS Typeout Format

When IDDT attempts to print an opcode 1Ø4 instruction symbolically, it first looks for an exact match in the user´s symbols. If one is found, the corresponding user-supplied name is printed. Otherwise, IDDT checks its own internal JSYS symbol table (hopefully, the same as JSYS´s defined in <SYSTEM>STENEX.MAC) for an exact match. If none is found in either place, the instruction will print as JSYS 5Ø1, i.e., "JSYS" and address.

17. Other Features

The search commands ($W, $N, and $N) have been generalized to take an argument which specifies the maximum number of "finds" that shall occur before the search will terminate. An example is:

```
            FOO<BAR>QQZZ$5E
```

This command will stop after typing five instructions lying between locations "FOO" and "BAR" which have an effective address of "QQZZ".

Internal register $I+4 contains the fork handle that IDDT uses to reference the user. This register is writeable.

$Q has the value of the last quantity typed, as always. $$Q has this value with halves swapped. Thus, ($$Q)= will type the same value as $Q= will.

$V is the value of the left half of the last quantity typed. $$V is the same with the sign extended. Thus, assuming the last value typed to be -3,,FOO , $V= would yield Ø,,-3 whereas, $$V= would type -3.

;? prints the most recent error encountered in the program.
#
# ;<space> prints the contents of $Q in the current typeout
# mode.

```
#                          IMGPTP
#
#
# Copies image-mode binary files (one byte per word), such as those
# written by PALX, to the paper tape punch.
#
# IMGPTP is designed  to  operate  in  quasi-command  style  (TENEX
# Executive Manual, p. 13).  To use the program in this manner:
#
#
# @IMGPTP (INPUT FILE) filename.ext [CONFIRM]%
# @
#
#
# Filename recognition operates for entry of the input  file  name,
# with  the  extension  .BIN supplied automatically (if it exists).
# If the filename typed is in error, a ?  is  typed  and  the  name
# entry  must  be  begun  again.   If  the PTP:  is unavailable the
# subsystem so informs the user and terminates.
#
# If there are no errors, IMGPTP first punches some blank leader at
# the  PTP:,  then  punches  a visible (human-readable) identifying
# leader, then copies the contents of the input file to the  punch,
# and finally punches blank trailer before returning control to the
# EXEC.  The identifying leader  consists  of  the  full  name  and
# date-last-written of the input file, e.g.,
# <DIREC>NAME.BIN;2  1Ø-JUN-73.
```

LBLOCK

LBLOCK is a small subsystem to perform "lineblocking" of text files. Its function is to read a text file, and write another text file, after inserting NULL characters where necessary to satisfy the condition that no line of text may be split across a record boundary (as defined in the DEC terminology).

The input file may be any TENEX text file, with the usual name recognition and defaults. The output file is specified in the shorter DEC form of "device:name.ext" where DSK is assumed for device and "name" may be up to 6 characters long, and "ext" may be up to 3 characters long.

Sample dialogue:

```
@LBLOCK
INPUT FROM FILE UNBLOCKED.DATA;3
OUTPUT TO FILE BLOKED.DAI
DONE.
@
```

LINK1Ø

```
# LINK-1Ø, the DECsystem-1Ø Linking Loader is a utility program which
# can merge independently-translated modules of a person´s program into
# a single module.  It prepares and links this input with other
# modules required by the user into a form that can be executed by
# the operating system.
#
# The general command format is
# */Switches File spec /Switches , /Switches File spec /Switches -
# , /Switches File spec /Switches /GO
#
# Lines may be continued by use of hyphen (minus sign) at end of line.
#
# A File spec is some of DEVICE:FILE.EXT[directory]
# Output file specs may be separated from input ones by = ,
# in which case they must be first on the line.
#
# Switches take optional arguments, these are :-
# Keyword        a symbolic keyword
# Symbol         a sixbit symbol names (in ascii)
# Value          in decimal (octal if preceded by # )
# Value          in octal
# Core size     in K (2ØØØ words) or P (1ØØØ words)
#               as in nK or 1K+hK
# Version       standard version number
#
# Switches and keywords preceded by * are unique to  one character.
# Switches enclosed in parentheses are switches known to SCAN but
# not used by LINK-1Ø.
#
# Switches enclosed in angle brackets are only available with
# non-standard assembly options.
#
# Switches are :-
# BACKSPACE     decimal value
# (BEFORE)
# COMMON         symbol:decimal value
# CORE          core size
# CONTENTS      keyword
#               Default, All, None, Global, Noglobal, Local, Nolocal,
#               Entry, Noentry, Relocatable, Norelocatable, Absolute,
#               Noabsolute, Common, Nocommon, Zero, Nozero
# COUNTER
# DATA
# *DEBUG         keyword
#               Macro, Ddt, Fortran, Mantis, *Cobol, Cobddt
# DEFAULT        keyword
#               Input, Output
# DEFINE         symbol:decimal value
# (DENSITY)
# ENTRY
# (ERNONE)
# (ERPROTECTION)
# ERRORLEVEL   decimal value (Ø-3Ø)
# ESTIMATE     decimal value
# EXCLUDE
# *EXECUTE
# FOROTS
# FORSE
# FRECOR        decimal value
# *GO
# HASHSIZE     decimal value
# *HELP
# INCLUDE       symbol
# *LOCALS
# <KLUDGE>
# LOG
# LOGLEVEL     decimal value (Ø-3Ø)
# *MAP         keyword
#              End, Now, Error
# MAXCOR        core size
# MPSORT        keyword
#              Alphabetical, Numerical
# MTAPE         keyword
#              Mtwat, Mtrew, Mteof, wmtskr, wMskt, Mtbsr, Mteot
#              Mtunl, Mtblk, Mtskf, Mtbsf, Mtdec, Mtind
# NEWPAGE       keyword
#              Low, High
# NOENTRY       symbol
# NOINITIAL
# <NOKLUDGE>
# *NOLOCAL
# (NOOPTION)
# (NOPHYSICAL)
# NOREQUEST    symbol
# NOSEARCH
# NOSTART
# (NOSTRS)
# NOSYMBOL
# NOSYSLIB     keyword
#              Default,F4Ø,Cobol,Algol,Neliac,Fortran
# (OKNONE)
# (OKPROTECTION)
# (OPTION)
# OTSEGMENT    keyword
#              Default, Low, High
# (PARITY)
# PATCHSIZE    decimal value
# (PHYSICAL)
# PROTECTION   octal value
# REQUEST
```

#	REQUIRE	symbol
#	REWIND	
#	RUN	file spec
#	RUNAME	symbol
#	RUNCOR	core size
#	(RUNOFFSET)	
#	SAVE	core size
#	*SEARCH	
#	SEGMENT	keyword
#		Default, Low, High
#	SEVERITY	decimal value (Ø-31)
#	SET	symbol:symbol or octal value
#	(SINCE)	
#	SKIP	decimal value
#	SSAVE	core size
#	START	symbol or octal value
#	(STRS)	
#	SYMBOL	keyword
#		radix5Ø, triplet
#	SYMSEG	keyword
#		Default, Low, High
#	SYSLIB	keyword
#		Default,F4Ø,Cobol,Algol,Neliac,Fortran
#	SYSORT	keyword
#		Alphabetical, Numerical
#	TEST	keyword
#		Macro, Ddt, Fortran, Mantis, *Cobol, Cobddt
#	*UNDEFINED	
#	UNLOAD	
#	VERBOSITY	keyword
#		Short, Medium, Long
#	VALUE	symbol
#	VERSION	version
#	XPN	
#	ZERO	

LOADER

The LOADER is essentially the most recent PDP-1Ø LOADER release with a few modifications for use on TENEX. It is capable of loading REL files produced by all language processors including FAIL, SAIL and ALGOL.

1) The /Y switch causes the high segment load point to be moved to the next 1ØØØ word page boundary. /Y does the same for the low segment.

2) /nH where n is a big number, behaves for the high segment the way /nO does for the low segment.

3) Multiple listings of undefined and multiply defined globals is suppressed.

4) A special block type is implemented to handle the ASSIGN pseudo-op. See MACRO writeup for more details.

5) /S and /B switches are defaulted to be "ON".

6) Just before the LOADER finishes, it sets the current subsystem name to "(PRIV)". This name is saved and restored by the EXEC and IDDT when the loaded program is started.

For further information refer to DECsystem1Ø Assembly Language handbook.

LOGOS (The LOGO Language)

LOGO is an interpreted, procedure-based language with strings as its fundamental data type. It was designed at BBN to be used by students across a wide range of educational situations. LOGO has been used by students at all levels from the second grade through college to write programs ranging from a couple of lines to complex procedure structures embodying several hundred lines in all. Such use is enhanced by LOGO's very clean syntax and good error evaluation.

LOGO permits recursion in user-defined procedures, in fact, this is the basic form for repetition. Recursion, together with a primitive which evaluates its input as a LOGO command string (an EVAL) and with primitives which access procedure lines make possible fairly complex program structures.

A complete reference manual describing LOGO has been written. The introduction is appended here to give a "sense" of the language.

LOGO REFERENCE MANUAL

1. A Look at LOGO

We introduce LOGO by writing several small procedures. The following examples serve to show what LOGO "looks like". Several features are used without definition or even explanation, where we think their meanings are clear from context. All of LOGO is comprehensively described in later sections.

LOGO, as an interpretive language, can execute single commands directly. Thus,

```
PRINT SUM OF 2 AND 2    (The user's typing is underlined) 4
```

But, the most important feature of LOGO is that such commands can be incorporated in user-written procedures. The definition of any procedure results in an object which is treated just like any primitive. Thus, in a very real sense, as the user writes his own procedures, he is gradually extending the basic language to more exactly fill his needs.

A very simple (although by no means simplest) procedure, for example, prints the double of its input.

```
TO DOUBLE :N:
10 PRINT SUM OF :N: and :N:
END
```

```
# This procedure, DOUBLE, is now "part" of LOGO.
#
#  DOUBLE 123
# 246
#  DOUBLE WORD OF 1 AND 1
# 22
#
# If the concatenation of DOUBLE with other procedures is  desired,
# DOUBLE  should  OUTPUT  rather  than  PRINT  its results;  OUTPUT
# meaning that the result is given to the calling  procedure.   The
# modified procedure is:
#
# TO DOUBLE :N:
# 1Ø OUTPUT SUM OF :N: and :N:
# END
#
# This new version of DOUBLE can be used in direct commands,
#
#  PRINT DOUBLE DOUBLE DOUBLE 3
# 24
#
# or can be used as the basis for other procedures,
#
# TO QUADRUPLE :N:
# 1Ø OUTPUT DOUBLE OF DOUBLE OF :N:
# END
#
# and so on.  This  very  natural  use  of  functions  in  LOGO  is
# particularly valuable, since to program a problem a user can keep
# on breaking it up until he sees subproblems which he  feels  will
# be  easy  to  program.  This  heuristic is used by sophisticated
# problem-solvers generally, whether or not computer programming is
# involved.
#
# An extension of  this  LOGO  facility  for  using  procedures  in
# defining  other  procedures  is  its  ability to handle recursive
# procedure definitions.  The recursive can be a linear one,  which
# is  equivalent  to iteration, as in the following procedure which
# calculates the factorial function:
#
#                n! = n . (n-1) ... 2.1.
#
# TO FACTORIAL :NUMBER:
# 1Ø TEST IS :NUMBER: 1
# 2Ø IF TRUE OUTPUT 1                (1! = 1)
# 3Ø OUTPUT PRODUCT OF :NUMBER: AND
#        (FACTORIAL OF DIFFERENCE OF  (n! = n . (n-1)!)
#        :NUMBER: AND 1)
# END
#  PRINT FACTORIAL OF 100
# 93326215443944152681699238856266700490715968264381621468592963895217
# 5999932299156089414639761565182862536979208272237582511852109168640
# 00000000000000000000000
#
# LOGO makes few distinctions between numbers and more general types  of
# string.   Thus,  a  procedure to reverse a string looks very much like
# FACTORIAL.
#
# TO REVERSE :STRING:
# 10 TEST IS :STRING: :EMPTY:         (The reverse of the empty string
# 20 IF TRUE OUTPUT :EMPTY:           is the empty string.)
# 30 OUTPUT WORD OF LAST OF :STRING:  (The reverse of the nonempty
#       AND (REVERSE OF BUTLAST OF    string is the string formed by
#       :STRING:)                     following the last character by
# END                                 the reverse of the rest of the
#                                     string.)
#
# ("AND", "OF", and parentheses  are  optional  "noise  words"  for
# convenience in writing expressions.)
#
#  PRINT REVERSE OF "JABBERWOCKY"
# YKCOWREBBAJ
#
```

LPTPLOT

Permits plotting graphical data on the line printer.

```
@RUN LPTPLOT.SAVE;1

WOULD YOU LIKE INSTRUCTIONS? ANSWER Y OR N:Y

THIS PROGRAM PERMITS A FAIRLY COARSE X-Y PLOT OF DATA
CONTAINED IN ANY ASCII DISK FILE TO BE PREPARED FOR LISTING ON
THE LINE PRINTER.  THE PLOTTING GRID CONTAINS 61 POINTS
(6 INCHES) HORIZONTALLY AND 49 POINTS (8 INCHES) VERTICALLY.
THE PROGRAM REQUESTS XMIN AND YMIN COORDINATES (WHICH WILL
FALL IN THE LOWER LEFT CORNER OF THE PLOT) AND SCALE FACTORS
XINT AND YINT FOR EACH ONE-INCH INTERVAL HORIZONTALLY AND
VERTICALLY.  A ONE-LINE TITLE MAY BE ENTERED.  THEN YOU CAN
CYCLE THROUGH THE PROGRAM AN UNLIMITED NUMBER OF TIMES,
DRAWING LINES (BY SPECIFYING THE COORDINATES OF THE END
POINTS), DRAWING CIRCLES (BY SPECIFYING THE ORIGIN
COORDINATES AND THE RADIUS), OR PLOTTING X-Y DATA FROM AN
ASCII DISK FILE.  AT EACH STEP, ARBITRARY PLOTTING SYMBOLS
(*,@,X,ETC) CAN BE SPECIFIED, AND THE PLOTTED DATA WILL
OVERLAY PREVIOUSLY PLOTTED DATA.  DISK FILE NAMES ARE LIMITED
TO 5 CHARACTERS PLUS A 3 CHARACTER EXTENSION.  AN ARBITRARY
NUMBER OF FILE LINES CAN BE SKIPPED TO AVOID TITLE
INFORMATION, AND AN ARBITRARY FORTRAN FORMAT (UP TO 25
CHARACTERS) CAN BE SPECIFIED FOR READING DATA.  AN EXAMPLE:
(45X,2F7.0).  THE PARENTHESES MUST BE PRESENT.  NORMALLY, THE
FIRST NUMBER READ WILL BE CONSIDERED THE X-VARIABLE AND
THE SECOND NUMBER THE Y-VARIABLE.  BUT IF A "Y" ANSWER IS
GIVEN TO "REVERSE X AND Y?", THE SECOND NUMBER WILL BE
CONSIDERED THE X-VARIABLE.  WHEN ALL PLOTTING IS DONE, ANSWER
"N" TO "MORE?".  THE PROGRAM WILL THEN PRODUCE A DISK FILE
NAMED PLOT.DAT, WHICH CAN BE LISTED.

XMIN AND INTERVAL (THERE ARE 6 INTERVALS):-3. 1.

YMIN AND INTERVAL (THERE ARE 8 INTERVALS):-4. 1.

TITLE:
THIS IS A SAMPLE PLOT OF A CIRCLE, A SINE WAVE, AND A LINE.

DRAW LINE?N

DRAW CIRCLE?Y

X0,Y0=0. 0.
```

```
RADIUS=3.

PLOTTING SYMBOL=@

READ INPUT DATA?Y

FILE NAME(5 CHARS)=SINE

FILE EXT (3 CHARS)=TEL

SKIP FIRST N LINES. N=Ø

INPUT FORMAT FOR X,Y=(2F6.Ø)

REVERSE X AND Y?N

PLOTTING SYMBOL=*

MORE?Y

DRAW LINE?Y

X1,Y1=-3. 4.

X2,Y2=3. -4.

PLOTTING SYMBOL=Ø

DRAW CIRCLE?N

READ INPUT DATA?N

MORE?N

CPU TIME: 7.47 ELAPSED TIME: 5:2Ø.ØØ
NO EXECUTION ERRORS DETECTED

EXIT.
```

MACRO

The TENEX MACRO is a slightly modified version of the most recent DEC MACRO-10.

The modifications are:

1) The /D switch causes tabs to be included in MACRO arguments. This is the DEC standard and may be necessary for assembling certain programs such as SNOBOL.

2) ASSIGN SYM1, SYM2, N pseudo-op causes the global SYM1 to have the same value as SYM2 and then increments the value of SYM2 by N. If N is not specified, it is assumed to be one. Note: a special REL file block type and appropriate LOADER modification were created to complement ASSIGN.

3) JSYS and UMOVEx are included in the OP-code table.

4) ORG ADDR pseudo-op behaves like either a LOC or RELOC depending on the mode of ADDR. ADDR may be an expression.

5) The version of MACRO on the system has all of the STENEX (JSYS and error mnemonics) definitions already included by use of the UNIVERSAL pseudo-op. This means that any program previously assembled with SYS:STENEX can now be assembled by including the statement SEARCH STENEX source file at the beginning. That is, if you add the search statement to the beginning of the program (immediately after the TITLE is a good place), you don't have to include SYS:STENEX in the command input. This results in a considerable saving of time, particularly significant in the case of small programs.

The only difference in this procedure is that MACRO will include in the output symbol table (for DDT) only those JSYS and error mnemonic definitions actually used in the program. This means that DDT* will not know about the other JSYS definitions which you may have occasion to use while debugging. If you want DDT to know all JSYS and error definitions, you can include SYS:STENEX.REL with the loading of your program. This has all of the JSYS and error mnemonic definitions defined as global symbols.

----

*IDDT does have the JSYS definitions built in, but it does not have the error mnemonics. For further information refer to DECsystem10 Assembly Language handbook.

MAILER

MAILER is the subsystem that delivers queued mail (i.e. mail which has been queued by SNDMSG). It runs in the background automatically, so it is ordinarily unnecessary for a user to run it. The following description is in two parts: the first gives general information on the handling of mail and is relevant to all users; the second is only relevant to a person who runs MAILER.

1. Handling of mail

Mail that has been queued (by SNDMSG) is placed in the file [--UNSENT-MAIL--].address in the directory of the user who sent it. (If a single message was sent to more than one address, there is a separate queued file for each.) MAILER, running automatically at frequent intervals (several times an hour) attempts to deliver each such file to the specified address.

If it succeeds, it deletes the file.

If it fails for some transient reason (such as the host being unavailable), it leaves the file alone and tries it again next time it runs.

If it decides the file cannot be delivered (for example, there is no such address) it renames it to be /UNDELIVERABLE-MAIL/.address (thus MAILER does not try again to send it, but it is not destroyed). MAILER also sends you (i.e. to your mailbox) a message (called a negative acknowledgement) telling you that the message was undeliverable and why. You may delete the undeliverable file, requeue it with a corrected address, etc. using MAILSTAT. If for some reason the negative acknowledgement cannot be delivered (for example, your mailbox is in use), it too is queued (it is placed in the file]--UNSENT-NEGATIVE-ACKNOWLEDGEMENT--[.address in your directory, where "address" is still the address of the original, undeliverable message) and delivered later.

2. Running MAILER

Here is a sample run, which was done while logged in as and connected to SUSSMAN.

```
@MAILER

No acknowledgments for SUSSMAN
Queued mail from SUSSMAN
BURCHFIEL@, sent ok, deleted.
CLEMENTS@, sent ok, deleted.
@
```

```
# MAILER processes queued mail and queued negative acknowledgements
# in the logged in directory and the connected directory (if
# different from the logged in directory).  Before processing a
# message, it types its address. After processing the message, it
# types the outcome, which is one of:
#
#
#   1) deleted - the file was deleted (after successful
#      transmission)
#   2) requeued - the file was left alone - it couldn´t be
#      delivered now, but will be processed again later
#   3) renames - the message was undeliverable, and the file was
#      renamed as described above
#
# Between the address and the outcome, messages may tell you why
# the file was undeliverable, that it was sent OK, etc.
#                              MAILSTAT
#
#
# MAILSTAT lists all queued and undeliverable mail in the connected
# directory.    It    also    accepts    commands    to   manipulate   the
# undeliverable messages - they can be deleted or put back  on  the
# queue to be mailed (with a different address if desired).
#
# See Examples below of MAILSTAT features upon usage.
#
#
#
# EXAMPLES ----    (user typing is underlined)
# @mailstat
# MAILSTAT 1B(11)
# Type ? for help
# *?
# MAILSTAT lists existing queued and undeliverable mail in the
# connected directory.  It also permits manipulation of
# undeliverable mail.
#
# Commands are:
#
# Q - list queued mail
# U - list or manipulate undeliverable mail
# H - halt - exits program
# ? - types this message
# carriage return - lists both queued and undeliverable mail
#
#
# *Queued mail:
#
# AIGHES     27-JAN-75 12:04:53
# AMSDEN     27-JAN-75 12:04:54
# CWILLIAMS     27-JAN-75 12:04:57
#
# No undeliverable mail.
#
# *u
# type ? for help
# **?
# Commands are:
#
# L - list undeliverable mail
# carriage return - same as L
# H - halt - exit to higher level (back to *)
# ? - types this message
# M - manipulate undeliverable mail -
#     Same as L, but after each item is listed, waits for you to choose
#     one of the following options.  Except as noted below, the
#     option must be confirmed with a carriage return, or may be
#     aborted with rubout (del).
#
#   S - save - does nothing - status of mail is unchanged
#   carriage return - same as S, but no confirmation required
#   D - delete message
#   Q - requeue message
#   A - address change - allows you to specify a different address for the
#       mail and requeues the mail for the modified addressee.
```

MIDAS

MIDAS is the assembler normally used at M.I.T. PROJECT MAC on their PDP10´s, and under their ITS timesharing system. A version of MIDAS has been modified to run under TENEX. It does not use 10/50 UUO´s or ITS UUO´s. It uses TENEX JSYS´s.

Documentation on the MIDAS language will be found in PROJECT MAC AI memo number 90, "MIDAS", by Peter Samson".

The following are the only changes to MIDAS for the TENEX version.

1. The default input file extension is "MID" instead of ">".

2. MIDAS´s initial symbol table contains the TENEX JSYS definitions (but not the error codes).

It should be noted that the binary output of MIDAS is not compatible with LOADER. PROJECT MAC´s loader has not yet been TENEX-ized. A loader for the SBLK format produced by MIDAS is being worked on.

MAILBOX

The "mailbox finder" is a TENEX subsystem used to determine the appropriate user name and site name to which an individual's mail should be sent. It may be run directly from a terminal or called as an inferior procedure. This latter mode may allow mail forwarding by FTP, SNDMSG and MAILER. For instance,

The mailbox data is a text file stored at each TENEX site supporting the mailbox finder. Individual sites may wish to maintain mailbox information particular to their needs. An example is the forwarding of mail to either the BBN-TENEX system or the BBN-TENEXA system, depending on where the addressee maintains his mailbox, and independent of which system initially receives the mail.

The mailbox finder returns a mailbox specification as "user@site". The input to the mailbox finder is also a user@site pair, where the site may be omitted. (The default site is the local site.) When a person wishes to receive mail under any of several synonyms, he has the text file updated to include all such synonyms. Whenever a person or program asks the mailbox finder about such a synonym (including the mailbox itself), the mailbox user@site pair will be returned. Thus, for example, the text file entry for user JONES with mailbox on BBNA could look like

```
     \                            ;backslash delimits mailboxes
     JONES@BBN-TENEXA             first entry is mailbox
     JONES@BBN-TENEX              ;synonym for mail forwarding
     JJJ@(NIC ID)                 ;network information center identification
     \
```

A person at a terminal could produce the typescript

```
     @MAILBOX JONES@BBN
     JONES@BBN-TENEXA
     *
     @
```

An additional feature of the mailbox finder is the ability to limit the search to the specified site. This is accomplished by a "V" and an "S" on the second argument line, which is shown blank in the example above. The V says begin matching only to the specific site given, and the S says suppress hunting. If hunting is allowed (which is the default case), the mailbox finder would find all matches to JONES@BBN, and then continue to find any matches to JONES at any other site, and finally any NIC IDs which are "JONES".

Other possible arguments are "N", which says begin matching only NIC IDs, and "A", whih says begin matching all actual sites (but not NIC IDs). The default, as shown in the example above, is "A", begin matching all sites, then hunt to NIC IDs.

The text file format includes a mechanism for flagging a mailbox as belonging not to a person, but rather to a group. In this case a file name is returned. The group's secretary may update the file so named as the group's constituency changes. Systems programs such as FTP may use this feature to automatically distribute copies of group mail. When run from a terminal, the mailbox finder types an error comment when the user name supplied is a group name.

As suggested by the example and discussion above, NIC IDs are treated as users at a site named "(NIC ID)". Matching to them is under separate control, however, as specified by the N argument. For example, either

```
     @MAILBOX JJJ
     NS
```
or
```
     @MAILBOX JJJ@(NIC ID)
     VS
```
would produce the single reply
```
     JONES@BBN-TENEXA
     *
     @
```
and would not match, for instance, a mailbox
```
     \
     JJJ@BBN-TENEXA ;Johnson, J.J.
     \
```
On the other hand,
```
     @MAILBOX JJJ
     AS
```
would match only Johnson
```
     JJJ@BBN-TENEXA
     *
     @
```
To match both, a user could type
```
     @MAILBOX JJJ
     JJJ@BBN-TENEXA
     JONES@BBN-TENEXA
     *
     @
```

MRUNOFF

MRUNOFF is a program which will produce a formatted manuscript from a source file. With MRUNOFF it is quite simple to achieve presentable results. However, MRUNOFF contains a number of sophisticated capabilities which allow the seasoned user a great deal of flexibility and control over the resulting manuscript.

The original MRUNOFF was called ROFF and was written for Multics by Doug McIlroy of Bell Labs in March, 1969. Art Evans made extensive modifications to it in May and June, 1969. Dennis Capps added footnoting in 1970. Harwell Thrasher maintained the program during 1971. Bob Mabee added many new features (during 1971-1972) and brought the program to the 6180 Multics (from the 645) in 1973. Bernard Cosell converted the program to TENEX BCPL in February, 1975.

Running MRUNOFF

When you call MRUNOFF, it will first ask you for an input file - the main input file to be formatted - and then will await the specifictions of any desired options. Typing in a null line will start the processing of the input file. The option acceptor uses a standard TENEX command recognition routine.

Output to file **filename**
: Manuscript output will be directed to the indicated file. Any underscoring will be done on a line-by-line basis rather than the default per-character basis. In the absence of this option, output will appear on the controlling terminal.

Paginate
: Page breaks in the output are retained. This is the normal mode. Page breaks in the output can be suppressed by use of the "Don't Paginate" option.

Pause between pages
or
Stop between pages
: The program will wait for a carriage-return on the controlling terminal before beginning a new output page. The default mode is "Don't Stop between pages".

Wait before beginning
: The program will wait for a carriage-return on the controlling terminal before starting the first page, but will not otherwise pause. The default mode is "Don't Wait before beginning".

Number lines
: Source line numbers wil be printed in the left margin. This option forces a minimum indentation of ten spaces. The default mode is "Don't Number lines".

Indent **nnn**
or
Margin **nnn**
: The output will be indented "nnn" spaces from the left margin. The default mode is "Margin Ø".

From page **nnn**
: Specifies the lowest numbered page of the output that will actually be printed. Default is "From page Ø".

To page **nnn**
Specifies the highest numbered page of the output that will actually be printed. Default is "To page 99999".

Passes to be done **nnn**
Specifies that nnn passes should be made over the input file. Output will be produced only on the last pass. The default mode is "Passes to be done 1".

Parameter to be set to **xxxxxx**
Establishes an initial value for the variable "Parameter". The default is for "Parameter" to be set to the null string.

Hyphenate
Enables the hyphenation package. The operation of the hyphenation package is discussed on page 36. The default mode is "Don't hyphenate".

Chars File
When this option is selected, various key characters in the output file will be flagged. The flags will be written to a file with the same directory and name as the main source file, but with extension .CHARS. The default mode is "No Chars File".

Start numbering with page **nnn**
Output page numbering will begin with the first page of output being assigned number nnn. The default mode is "Start numbering with page 1".

Don't
or
No As indicated in the affected options above, this command may be used with some options to reverse their sense. For example, a "Don't hyphenate" command would undo the effects of any preceding "Hyphenate" command.

The Basics of MRUNOFF

An MRUNOFF source file contains two type of lines: control lines and text lines. A control line begins with a period; all other lines are considered to be text lines. The control lines are, in general, directives to MRUNOFF and will not appear in the output file. Text lines are formatted as directed by preceding control lines and program-initiation options and written to the controlling terminal or the selected output file.

File name defaults. MRunoff attempts to make its defaults for the various fields in the file names it uses as reasonable as possible in order to allow the User as much abbreviation as possible. For its main input file, MRunoff will look in the connected directory for extension **.MRN;** the file name must always be specified. For all other files, MRunoff will always default to the directory used for the main input file. If hyphenation is desired, the default file for the dictionary is **HYPHENATION.DICTIONARY.** The output and chars files default to have the same name as the main input file with extensions **.DOC** and **.CHARS,** respectively. Inserted files have default extensions of **.MRN,** and the file name must always be specified.

Filling. When the text is being "filled" all ends-of-lines in the input file are ignored beyond identifying control lines. Material is taken from the input file and added to an output line as long as there is room. A line will be written only when either there is insufficient room for the next input file word or a "break" is encountered. If the output is being "adjusted" (the default mode), before the filled output line is written it will be padded with sufficient spaces so that it exactly fills the specified output line length. Successive lines are padded alternately from the left end and from the right end.

Breaks.. A "break" insures that the text following it will not be run together with the text that preceded it. Breaks can be introduced into the text explicitly by use of a .br control line. In addition, many other control lines will cause a break. If an input line begins with a space, it will cause a break.

Tabs. MRUNOFF maintains a table of tab stop locations. Tab stops are initialized to occur at locations 11, 21, 31, etc., and can be changed with a .tb control line. As text is prepared for output, tab characters (I, ASCII Ø11) are replaced with the required number of spaces (but always at least one) to fill out to the next tab stop counting locations from the first print position on the line, independent of where that print position may fall on the page due to .in or .un control lines. Tab characters in control lines are not handled any differently than any other character.

Sentences. When an input text line ends with any of the characters ".", "?", "!", ";", or ":", or with ".", "?", or "!" followed by a double quote or ")", two blanks will precede the following word (if it is placed on the same output line), instead of the normal single blank. te that if these characters appear anywhere else on the input line no special action will be taken; however, within a text line, any multiple spaces are preserved. Thus, if there are three spaces between a pair of words and they get put on the same output line, then they will be separated by at least three spaces, and perhaps more if the line is adjusted.

The layout of the page. Verticlly, an output page contains three areas. The first consists of **headers,** which are printed at the top of each page. There may be up to twenty lines of headers. Header lines are numbered from the top down. Next comes the text body, which contains both text and footnotes. Lastly, come **footers,** which are printed at the bottom of each page. Again, there may be up to twenty lines of footers, but footers are numbered from the bottom line upwards. These three areas are spaced by four margins. The first margin on the page is the number of blank lines above the first header; the second is the number of blank lines between the last header and the first line of text; the third is the number of blank lines between the last line of text and the last footer; the fourth is the number of blank lines below the first footer. These margins are set up, respectively, by **.m1, .m2, .m3,** and **.m4** control lines, and the current values of these may be referenced through the symbols **Mr1, Mr2, Mr3** and **Mr4;** the default margins are four lines at the top and bottom of the page, and two lines around the text body.

Line spacing is always done just before a line is printed. For example, when a "page break" occurs (say, because of a **.bp** control line), the footnotes and footers for the current page are written immediately, but the headers on the new page are not written until the first line of text is ready to be written on it. In particular, when MRUNOFF begins processing a file, it considers itself just "above" the first page, and if any headers are set up before any text is placed on the first output page, the headers will be printed on the first page.

Character Set. MRUNOFF expects its input to utilize the full ASCII character set. However, it will perform case conversion if that is necessary. The variable **CASECONTROL** should be set to the ASCII code for the character to be used for signalling case information; setting **CASECONTROL** to zero disables the case conversion facility, and it is initialized that way. MRUNOFF performs case conversion by interchanging characters with ASCII codes #1ØØ through #137 with those of codes #14Ø through #177. Double occurrences of **CASECONTROL** will complement the overall mode of converting either all characters or none, while single occurrences will complement the effect of the overall mode for the immediately following character only.

MRUNOFF is nominally set up to produce output using the full set of ASCII graphics. However, MRUNOFF contains various facilities for dealing with devices with other character sets. The variable **TrTable** is a table whose **n**th entry provides the translation for the character with ASCII code **n**. If the entry is a number, then that number is the ASCII code which will be added to the output file (or sent to the terminal); if the entry is a string then the entire string will be used as the translation. Note that these translations are used for **all** output including the formfeed (via the translation for ASCII 12.) used to eject pages and the carriage return - line feed sequence (via the translations for ASCII 13. and 1Ø.) used to advance from line to line. **TrTable** is initialized to have all of the ASCII graphics and all of the ASCII format effectors translate to themselves, and all other codes translate to Ø (ASCII NUL). By use of **.tr** control lines or **.st** control lines, **TrTable** can be modified to suit MRUNOFF's output to the desired device. For example, if the User's device has some special graphics the User has the choice of using the appropriate control code directly and declaring the code to translate to itself, or for more convenient source preparation in a standard ASCII environment some little used ASCII graphic could be translated into the appropriate control code. Further, if the output device has some special capability (e.g., subscripting), the User can have any convenient character translate into the appropriate escape sequence.

In order to do filling and justifying correctly, MRUNOFF must know the printing width each ASCII code takes up on the output device. This is done by means of another table, **Widths.** The **n**th entry of this table specifies how much space the character with code **n** will occupy on a line. This width is calculated before any **TrTable** translation is done; that is, the code **n** refers to the ASCII code of a source file character.

MRUNOFF also has a facility to assist in the preparation of manuscripts for devices with an inadequate collection of graphics for the job at hand. For these cases, by requesting the **Chars file** option and using **.ch** control lines, or **.st** control lines directly upon the table **CharsTable,** the User can specify that whenever MRUNOFF encounters a particular character, in addition to translating it into the output, MRUNOFF should make an entry into a second output file to indicate that that particular location in the manuscript will require some attention after it is typed out. **CharsTable** is indexed by the ASCII code of the input file character (i.e., before it has been translated). If the entry for a code contains a SPACE (ASCII 32.), then the character will be passed to the .CHARS file with its **TrTable** translation and will not be flagged. If the entry for a character contains any other number, then that code will be sent to the .CHARS file instead of the normal translation and the character will be flagged. Unless an output line contains at least one flagged character, it will be suppressed from the .CHARS file. MRUNOFF initializes **CharsTable** so that all of the ASCII graphics and format effectors print as themselves, unflagged, the other control characters print as an appropriate lower case letter, flagged, NUL becomes a flagged accent grave, and RUBOUT becomes a majuscule R, flagged.

Page numbering. As the output is being prepared, a page number counter is kept. In addition to being set by the **Start numbering with page** option, the counter can be incremented or set directly from the source file (e.g., by a **.pa** control line). A page is called odd (even) if the current value of the counter is odd (even). To facilitate preparing manuscripts to be printed "double sided", the headers and footers can be specified independently for odd and even pages.

Symbols. Many of the variables internal to MRUNOFF are avaiable to the user by means of the symbol facility. In addition, the user may define and use his own symbols. This facility combined with the conditional command **(.ts)** and the looping commands **(.gf, .gb** and **.la)** allow the user, in effect, to write a program within MRUNOFF to "build" his output text. Details on the use and definition of symbol appears on page 18. In general, though, there is a special character which is reserved to indicate symbol substitution. This character is nominally **%**, but can be changed by a **.cc** control line or by resetting the value of the symbol **SpecChar.** When a symbol is to be substituted, the name of the symbol is surrounded by **SpecChars**. The **SpecChars** and the name of the symbol are replaced in the line with the value of the symbol before the line is processed. In order to include a **SpecChar** in the text, it must be doubled. Also, if a single **SpecChar** occurs in the text, it will b replaced by the current page number. For example, **.ts %=42** would test whether the current page number were 42, or **.ts "%Parameter%"="%%"** would test whether the symbol **Parameter** was equal to a per-cent sign.

Underscoring and Multiple Printing. MRUNOFF contains a facility for underscoring the output text. When this facility is enabled, a character is usurped for use as an underscoring flag. MRUNOFF maintains a global underscoring mode which indicates whether all output characters are being underscored or not. Underscoring is enabled, and a control char is selected, by means of a **.uc <char>** control line. Double occurrences of the selected character reverse the global underscoring mode, while single occurrences of the selected character reverse the global mode for the following character, only. For example, if the user included a **.uc $** control line at the beginning of his input file, then the text **$$Now is t$he $$time for $a$l$l good men** would appear in the output as Now is the time for all good men.

While MRunoff does not contain any facility for specifying changes of font (and for that matter, most output devices don't have the capability, anyway), MRunoff does have the ability to multiply strike characters which results in a fairly attractive simulation of a boldface font. The facility can be enabled by setting the variable **OverprintCount** to be greater than zero; is disabled by setting **OverprintCount** to be less than or equal to zero. This facility replaces the normal underscoring facility when it is being used. In particular, when enabled any character which would normally be underscored will instead be repetitively struck a number of additional times as given by the value of **OverprintCount.** Setting **OverprintCount** zero or negative will restore normal underscoring; disabling underscoring does not affect **OverprintCount** (and thus multiple printing, and not underscoring, would continue when underscoring is next enabled).

Titles. Several command lines accept a **title line** as an argument. A title line consists of a line with three distinct fields: the first is text to be placed flush with the left margin, the second centered, and the third flush with the right margin. The first non-blank character in the title will be used as the delimiter to separate the three fields. If fewer than four delimiters are present on the line, the missing parts of the title are taken to be blank. The justification and centering is done relative to the line length and margins in effect at the time the title line is processed, and is independent of their values at the time of use. For example, this document used the commands: **.he 1 'MRUNOFF''%FancyDate%'** and **.fo 1 ''- % -'.**

Flagging blocks of text. MRunoff has the capability to print a marginal flag (usually a vertical bar, **|**) along side the output text. The flagging character may be put to the left of the text or to the right of the text, or even in the middle of the text. If on an even numbered page and the variable **EvenFlagCol** is neither less than nor equal to zero, then its value gives the absolute print position in which a flag is to be printed, otherwise, no flags are added; the variable **OddFlagCol** functions similarly for the odd-numbered pages. The character which would normally be printed in that position is replaced by the character whose ASCII code is given by the value of **FlagChar.**

An MRunoff output line will always have **ExtraMargin** spaces at its beginning which are totally invisible to the formatting processes. An absolute print position is one which takes these "free" spaces into account (as, for example, the tab stop specification does not). Thus, if **ExtraMargin** were set to three, then a flag specified to occur in column 1 would appear three spaces to the left of the output text. If the flagging column is beyond the right end of the line, MRunoff will add spaces to the line as necessary to place the flagging character in the correct column. MRunoff will never flag headers or footers.

To make the use of this facility more convenient, a **.fl** control line will set **FlagChar** and both **EvenFlagCol** and **OddFlagCol.**

Footnotes. The occurrence of a **.ft** control line will cause all of the text and commands until the next occurrence of a **.ft** control to be collected as a footnote. The text of all footnotes defined on a page is accumulated and written together at the bottom of the page. The footnote text is completely set up at the time MRUNOFF encounters the defining **.ft** control line; thus, any commands embedded within the footnote will be executed while the footnote is read, not when the footnote is eventually inserted into the output. In particular, it is not possible to nest footnotes.

In order to make it more convenient to number footnotes, MRUNOFF maintains a special symbol, **Foot.** The value of this symbol is incremented by one whenever a **.ft** control line which ends a footnote is encountered. Thus, while in the main text, **Foot** will yield the next number to be assigned to a footnote, and while in the body of a footnote %Foot% yields the number of the footnote in progress. The user may select whether he prefers footnotes to be numbered consecutively through the entire document or whether he prefers that the numbering revert to "1" with each new page. Also, the automatic incrementing of **Foot** may be suspended entirely if desired.

The body of a footnote should occur immediately after the word which is to include the footnote reference. MRUNOFF will insure that the output line containing the reference will be kept on the page which receives the footnote. When MRUNOFF encounters a **.ft** control line beginning a footnote, it will append the value of the value of the **TextRef** to the last word on the preceding line **TextRef** is initialized to be **(%Foot%)**. Since a footnote does not cause a break in the main text, the only effect on the output is that the footnote reference has been appended to a word. Similarly, the value of the value of the symbol **FootRef** will be included as the first line of the footnote input text; **FootRef** is also initialized to be **(%Foot%)**.

Pictures. The User may set, and format, blocks of space for pictures. The formatting possible includes top- or bottom-of-the-picture captions, footnotes within the captions, etc; in fact, the full power of MRUNOFF* is available for use in laying

*With the exception of the **.gb** command. Due to the organization of the deferred text processor there is no way to loop back and re-execute any of the deferred text. If this is necessary, it out a space for a picture. A pair of **.pc** control lines ("picture with caption") are used to delimit a section of text and commands which should do any desired formatting (including actually leaving any blank space needed). The first **.pc** control contains an expression which specifies how much space the picture (including all captions, etc.) will require. The entire section between the **.pc** lines will be queued and deferred until a point is reached where all previously queued pictures have been processed and the indicated amount of space is available. Then the queued text and commands will be processed before any further text is taken from the input file. Notice that no space is "automatically" set aside; any desired empty space must be explicitly provided for. The space requested in the opening .pc control and the space actually taken up during the eventual processing of the caption need have no relation to one another. Also, regardless of how much space is actually requested, if MRUNOFF reaches the top of a new page at least one caption will be processed as the first thing (after the headers) on the new page; beyond the first caption, though, captions will only be processed if the space left on the page is at least as large as was originally requested for the caption.

can be done by putting the loop in a file and using .if to include it in the caption.

Expressions

Several commands require numeric or string arguments. For all such commands, the argument may be the result of evaluating an expression. The basic arithmetic operators, in order of decreasing precedence, are:

```
(, ) grouping
~ (bit-wise negation), - (unary)
*, /, // (remainder)
+, - (binary)
=, <> and >< (not equals), <, >, <=, >=, =<, =>
   (Comparison operators.  Yield -1 (if true) or
   Ø (if false))
& (bit-wise AND)
\ (bit-wise OR)
```

Octal number are indicated by prefixing the number with a "#". All other numbers are considered to be decimal.

Strings can also be dealt with. However, the only way that strings are handled is by building up a single string "constant" from other string constants in a strictly left-to-right manner. The basic string constant is a sequence of characters surrounded by double quote characters. Certain special characters are represented by sequences, as follows:

**	asterisk
*"	double quote
*b	backspace
*n	new-line
*t	horizontal tab
*s	space
***cnnn**	character whose ASCII code (in decimal) is **nnn** (one to three digits).

If two string constants are placed next to one another, the result is a new string constant which is the concatenation of the original two strings. Placing **(i)** or **(i,k)**, where **i** and **k** are arithmetic expressions, after a string constant results in a new string constant consisting of a substring of the original constant, defined as follows: **i** indicates the character position at which the substring is to begin, if **i** is negative, it indicates an offset from the right end of the string; **k** (or **-1** if **k** is not supplied) indicates the length of the substring to be extracted, or if **k** is negative it specifies the distance from the right end of the string at which the substring is to end. There are **no** other string operations; the presence of **any** arithmetic operator in the expression will convert it to being a numeric expression. The comparison operators work on strings: if "=" (or "<>" or "><") is used to compare two strings, the comparison will be done over their entire lengths, in order to result in "true" (or "false") the two strings must **exactly** match - in both content and length; if any other comparison operator is used between two strings, the result will be appropriate to the collating sequence (independent of case) of the two strings. One other numeric operation is defined for strings: if a string constant is immediately followed by a "#", it will evaluate to be the length of the string. In any other context in a numeric expression, a string is converted to a number in such a way that a one-character string results in the ASCII numeric value of the character; the numeric value of multi-character string constants is undefined.

For example, "ab" "cd" = "abcd"; "abcdefg"(3) = "cdefg"; "abc" "defg"(2,5) = "bcdef"; "abcd" "efgh"(3) "ijkl"(2,-2) = "defghijk".

Definition and Substitution of Variables.

Names of variables are composed of upper- and lower-case alphabetic characters, decimal digits, and "_" (underscore). Variables have either "string" or "numeric" values.

Values will be substituted for variables under the following circumstances:

1) In expressions if a **SpecChar** (normally %) is found as either the first or second character following the spacing after the control word. Note: any such % is not used as a flag and removed. Thus, there will be expressions in which symbol substitution is desired but which cannot be reordered to bring a (desired) symbol reference to the beginning. For example, **.sr Symbol "(%Date%)"**. In such cases, you must specify substitution explicitly with a **.ur** control line.
2) All **.ur** control lines.
3) In all title lines.

To indicate a symbol substitution, you enclose the name of the variable in % characters. % in any other context will be replaced by the value of the page counter; to include a %, you must code it as **%%**. If it is inconvenient to use %, you can change the substitution control character by means of a **.cc** control line.

When a substitution takes place, the **%name%** is removed from the line being processed and is replaced with the value of the variable: either the value itself if the variable is a string, or the appropriate string of (decimal) digits if the variable is numeric. Notice, then, that if you wish to preserve the "stringness" of a variable in an expression, you must surround the variable reference in double-quote characters. For example, to test if **%Parameter%** is equal to the string **"abc"**, you must write **.ts "%Parameter% = "abc"**.

If a variable is defined to be a table, references to its entries have a somewhat different syntax. The sequence **%Name|nnn%** will be replaced by the **nnn**th entry of the table **Name**. The index must be a decimal number; it may neither be an expression nor supplied by another substitution. Thus, to retrieve an element whose index has been saved in another variable, some technique like **.ur ... %%Name|%Index%%%...** would have to be used. Each element of a table may be a string or a number independent of the table's other elements.

Many of the variables internal to MRUNOFF are available to the user (a complete list will be found on page 33); these include control argument values (or their defaults), values of switches and counters, and certain special functions. Most are user settable, but for the ones that are not, any attempt to redefine them will merely make their system values inaccessible. This will cause no harm, and the user may freely use the name for a private variable.

A special symbol is provided for use in footnote numbering: **Foot** contains the value of the next footnote number available (or the current footnote if referred to from within the body of a footnote). The value of **Foot** is incremented by one when the closing **.ft** control line of a footnote is encountered. The user may select whether **Foot** should maintain a running count throughout the manuscript or whether it should be reset at the end of each page.

Also, by use of **.mc** control lines, variables can be set up to be counters. Any reference to such a variable provides its current value, and causes its value to be incremented by one automatically. This facility is useful for numbering equations, numbering references and other applications where the ability to maintain a sequential counter through the manuscript is convenient.

Control Line Formats

This section gives a description of each of the control lines which may be interspersed with the text for format control. Control lines do not cause an automatic break unless so specified. If a control line's command is defined as a string valued variable, the value of the variable will replace the period and the variable name and the line will be reprocessed (and hence might not be a control line when rescanned). This substitution and rescanning takes precedence over any definition of a particular control as described below; thus, this capability would allow one to redefined the built-in commands. Arguments of the control words are in the following form:

#	integer constant
n	integer expression
+n	integer expression preceded by optionial plus sign or minus sign
exp	arbitrary expression (string or integer)
c	single character
cd	sequence of character pairs
f	file name
ttl	a title line
xxx	remainder of the control line
+n,	a sequence of integer expressions (with optional preceding signs), separated by commas
name	a character sequence to be interpreted as a variable name (Note: the name should not be surrounded by %s)
str	string expression

(blank line)
A blank line occurring in the text is treated as if it were a **.sp** 1 control line unless the blank line would be the first line on a page, in which case the line is treated as though it were a **.br** control line.

(form feed)
Any line beginning with a form feed will be treated as though it were a **.bp** control line.

.name xxx
If **name** is string valued, then the **.name** portion of the line will be replaced by the value of **name** and the line will be reprocessed. This line is similar to the control line **.ur %name% xxx** excepp that no symbol substitution in the **xxx** portion of the line will take place.

.ad Adjust: text is printed right justified. Fill mode must be in effect for right justification to occur. Fill mode and adjust mode are the default conditions. This control line causes a break.

.ap name str
Append: If **name** is a string valued symbol, then **str** will be appended to that string (and left as the value of **name**). Otherwise, this control line has the same effect as **.sr sym str**.

.ar Arabic page numbers: when the page number counter is substituted into text or control lines by means of **%** (not by **%Np%**) it will be done in Arabic notation. All other numeric substitutions are always done in Arabic. This is the default condition.

.bp Begin page: the next line of text will be put on a new output page. This control line causes a break.

.br Break: the current output line is written as is, and the next input text line will be put on a new output line.

.cc c Control charcter: this control line changes the character used to surround the names of variables when they are referenced to be **c**. This changes all uses of the control character: a single **c** will become the current page number, and **c** will have to be doubled to be get a **c** into the output text. The default control character is **%**. If **c** is omitted, the control character will be reset to **%**.

.ce n Center: the next **n** lines of source text are centered between the current margins. If **n** is omitted, **1** is assumed. Filling and adjusting are suspended until the centering is completed. This control line implies **.ne %Ms%***n so that all lines centered will be on the same page. This control line causes a break.

.ch cd Characters: each occurrence of the character **c** will be replaced in the .CHARS file by the character **d**, and the character will be flagged. If the **d** character is blank, or an unpaired **c** character appears at the end of the line, the **c** character will not be flagged, and will occur as itself in the .CHARS file, or not at all if no other character in the line is to be flagged.

.ds Double space: begin double spacing the text. This control line is equivalent to **.ms 2.** This control line causes a break.

.dt name n
Define a table: **Name** will be defined to be a table of length **n.** The elements of **name** will be initiaized to zero (not the null string) and can be assigned values by means of a .st control line. The elements can be accessed by use of the **%name|nnn%** construct. The indexes for the table range from zero to **n**-1.

.eh # ttl
Even footer: this defines even page footer line number **#.** If the title line is omitted, the footer line with that number is cancelled.

.eh # ttl
Even header: this defines even page header line number **#.** If the title line is omitted, the header line with that number is cancelled.

.eq n Equation: the next **n** text lines are taken to be equations. If n is omitted, 1 is assumed. This ccntrol line implies **.ne %Ms%***n so that all equations will be on the same page. Each equation shold be formatted as a title line.

.fh ttl
Footnote header: before the first footnote on a page is printed, a demarcation line is printed to separate it from the text. The format of this demarcation line is specified by the **ttl.** The default footnote header is a line of underscores from column one to the right margin.

.fi Fill: this control line enables fill mode. This is the default condition. This control line causes a break.

.fl c, n
Flag control: The character **c** is set up as the flagging character. If **n** evaluates to be negative or zero flagging will be turned off, otherwise a flag will be printed in column **n** of every output line.

.fo # ttl
Footer: even and odd footers are set at the same time. This control line is equivalent to:

```
        .ef # ttl
        .of # ttl
```

.fr c Footnote reset: this control line specifies footnote numbering according to the argument **c.** Permissible values of the argument are:

- t Footnote counter is reset at the top of each page. This is the default condition.
- f Footnote counter runs continuously through the text.
- u Suppress numbering the next footnote. That is, the next closing **.ft** line will not change the value of **Foot.**

If any other value for **c** is supplied, **f** is assumed.

.ft Footnote: when **.ft** is first encountered, all subsequent text until the next **.ft** line is treated as a footnote. Any further text on the **.ft** line will be ignored. If a footnote occurring near the bottom of a page will not fit on the page, as much as necessary will be continued at the bottom of the next page. If a .ft line occurs when there are fewer than two lines remaining on the current page, the last text line on the current page will be removed, the page will be terminated, and the text line will be written as the first line on the next page.

.gb xxx
Go back: the current input file is searched from the beginning until a line of the form **.la xxx** is found. Processing is continued with the line following the matching **.la** control line.

.gf xxx
Go forward: same as **.gb** except that the search begins at the current position in the input file.

.he # ttl
Header: even and odd header lines **#** are set at the same time. This control line is equivalent to:

.eh # ttl
.oh # ttl

.if f exp
Insert file: the indicated file is inserted into the text at the point of the **.if** control line. The default directory is the currently connected one, and the default extension is ".MRN"; it is wise to cultivate the habit of always including <directory> in the file name. The inserted file may contain both text and control lines. No break occurs. Insertions may be nested to a maximum depth of thirty. If **exp** is provided, it will be evaluated and its value and type will be assigned to the variable **Parameter**; otherwise, the value of **Parameter** remains unchanged.

.in +n **In**dent: the left margin is indented **n** spaces by padding **n** leading spaces on each line. The right margin remains unchanged. If **n** is omitted, 0 is used. If **n** is preceded by a plus sign or a minus sign, the indentation is changed by **n** rather than reset. This control line causes a break.

.la xxx
Label: defines the lable **xxx** for use as the target of either **.gb** or **.gf** control lines.

.li n **Li**teral: this request causes the next **n** lines to be treated as text, even if they begin with ".". If **n** is omitted, 1 is assumed.

.ll +n Line length: the line length is set to **n.** The left margin stays the same, and no break occurs. The default for **n** is 65 both initially and if **n** is omitted. If **n** is preceded by a plus sign or a minus sign, the line length is changed by **n** rather than reset.

.lm +n **L**eft **m**argin: this control line has the same effect as a **.in +n** control line.

.ma +n **Ma**rgins: top and bottom margins are set to **n** lines. If **n** is preceded by a plus sign or a minus sign, the margin is changed by **n** rather than reset. The margin is the number of blank lines left above the first header and below the last footer. The default is four lines. This control line is equivalent to

.m1 +n
.m4 +n

.mc name

Make **c**ounter: defines **name** to be a counter variable. **Name** will be initialized to one. If **name** is reset (by a **.sr** control line), it will retain its counter property so long as it is only reset to a numeric value. When a counter variable is referenced its value is returned first, and then incremented.

.mp +n Multiple pages: format the output text so that it prints on every nth page. The defaut value is 1.

.ms +n Multiple space: begin spacing the text so as to leave (n-1) blank lines between text lines. If **n** is preceded by a plus sign or a minus sign, the acing is changed by **n** rather than reset. If **n** is not given, 1 is assumed. This control line causes a break.

.m1 +n Margin 1: the number of blank lines left above the first header is set to **n**. If **n** is preceded by a plus sign or a minus sign, the margin is changed by **n** rather than reset. The default is 4.

.m2 +n Margin 2: the number of blank lines left between the last header and the first line of text is set to **n**. If **n** is preceded by a plus sign or a minus sign, the margin is changed by **m** rather than rset. The default is 2.

.m3 +n Margin 3: the number of blank lines left between the last line of text and the 1st footer is set to **n**. If **n** is preceded by a plus sign or a minus sign, the margin is changed by **n** rather than reset. The default is 2.

.m4 +n Margin 4: the number of blank lines left beteen the first footer and the bottom of the page is set to **n**. If **n** is preceded by a plus sign or a minus sign, the margin is changed by **n** rather than reset.

.na No adjust: justification mode is disabled. This control line causes a break.

.ne n Need: a block of **n** lines is needed. If **n** or more lines remain on the current page, text continues as before; otherwise, the current page is ejected and text is continued on the next page. No break occurs unless it is necessary to advance to a new page.

.nf No fill: fill mode is disabled, so that a break is caused after every text line. Adjustment mode is ignored, and adjusting is not performed. Upon a .fi control line, adjustment will continue if it is enabled at that time.

.of # ttl
Odd footer: this defines odd page footer line number **#**. If the title line is omitted, footer line **#** is cancelled.

.oh # ttl
Odd header: this defines odd page header line number **#**. If the title line is omitted, header line **#** is cancelled.

.op Odd page: the next page is forced to be odd by adding one to the page number counter if necessary. A break occurs and the current page is ejected.

.pa +n Page: The page number counter is set to **n**. If **n** is preceded by a plus sign or a minus sign, the counter is changed by **n** rather than reset. A break occurs and the current page is ejected.

.pc n	Picture with caption: if **n** lines remain on the present page, then the text and commands up to the next .pc control line are immediately processed, after which normal text processing continues without a break occurring (independent of anything that might appear within the body of the caption). Otherwise, the text and commands up to the next .pc control line are queued up and will be processed when the next page is reached.
.pi n	Picture: Set aside a blank space **n** lines long. This control line is equivalent to: **.pc n** **.sp n** **.pc**
.pl +n	Page length: the page length is set to be **n** lines. The default is 66. If **n** is preced by a plus sign or a minus sign, the page length is changed by **n** rather than reset.
.rd	**Read:** one line of input is read from the controlling terminal. This input line is then processed as if it had been encountered instead of the .rd control line. Thus, it may be either a text line or a control line. A break occurs only if the processing of the terminal line causes one.
.rm +n	Right **m**argin: this control line has the same effect as a **.ll +n** control line.
.ro	**Ro**man numerals: establishes that whenever the page counter is substituted for a **%** (not for **%Np%**) it will be done with Roman, rather than the default Arabic, numerals.
.rt	Return: terminates processing characters from the current input file and continues processing from the line following the .if control line of the previous input file.
.sk n	**Skip: n** page numbers are skipped before the new page by adding **n** to the current page number counter. No break in the text occurs. This control line may be used to leave out a page for a figure. If **n** is omitted, 1 is assumed.
.sp n	**Space**: space **n** lines. If **n** is not given, 1 is assumed. If not enough lines remain on the current page, footers are printed and the page ejected, but the remaining space is not carried over to the next page. However, if MRunoff is already at the top of a new page (even though it may not have printed the headers on that page yet), the space will be left immediately after the headers are printed. This control line causes a break.

.sr name exp
Set reference: associate the value (and type) of **exp** with the variable **name.** **Name** may be either a user-defined identifier or one of the built-in symbols.

.ss Single space: begin single spacing text. This control line is equivalent to **.ms 1.** This is the default condition. This control line causes a break.

.st name n,exp
Set a table entry: Entry **n** of table **name** will be set to **exp.** Notice that the comma is required.

.tb +n,
Tab stops: Sets a tab stop at the position indicated by the value of each expression in the remainder of the control line. If an expression is preceded by a plus sign or a minus sign, the expression will specify an offset from the previous tab stop location.

.tr cd **Tr**anslate: the nonblank character **c** is translated to **d** in the output. An arbitrary number of **cd** pairs can follow the initial pair on the same line without intervening spaces. An unpaired **c** character at the end of the line will cause **c** to be translaaed to a blank. (translation of a character to a blank in the output is useful for preserving the identity of a string of characters, so that the string will not be split across a line, nor have padding inserted within it.)

.ts n Test: the next input line will be ignored if the value of **n** equals zero (false). The default value is non-zero.

.ty xxx
Type: write **xxx** onto the ccntrolling terminal. Substitution of variables may occur if the first or second character of **xxx** is **%.**

.uc c Underscore character: establishes the character to be used to indicate underscoring. If **c** is omitted, underscoring is completlely disabled and no character is intercepted for this purpose. This is the default mode.

.un n **Un**dent: the next ouput line is indented **n** spaces less than the current indentation. Adjustment, if in effect, will occur only on that part of the line between the normal left indentation and the right maagin. If **n** is not specified, its value is the current indentation (i.e., the next output line will begin at the left margin). This control line causes a break.

.ur xxx
Use reference: **xxx** will be scanned, and any indicated variable substitutions will be performed. The line thus constructed is then processed as if it had been encountered in the original input file (e.g., it may be another control line - even possibly aaother **.ur**).

.wt **Wait**: read one line from the controlling terminal and discard it.

.* This line is treated as a comment and ignored. No break occurs.

.~ This line is treated as a comment and ignored. However, the line is copied into the .CHARS file.

Control Line Summary

The following conventions are used to specify arguments on control lines:

c	a single character
cd	a sequence of character pairs
exp	expression (either numeric or string)
#	integer constant
n	numeric expression
+n	+ or - indicates update by n, otherwise set to n
f	file name
t	title line ('part1'part2'part3')
+n,	sequence of +n expressions separated by commas
sym	variable name
str	string valued expression

An asterisk at the beginning of "Meaning" indicates that the control line causes a break.

Request	Default	Meaning
(blank line)		*Equivalent to **.sp 1**
(form feed)		*Equivalent to **.bp**
.ad		*Right justify text
.ap sym str		Append **str** to **sym.**
.ar		Arabic page numbers
.bp		*Begin New page
.br		*Break, begin new line
.cc c	%	Change special character to c
.ce n	n=1	*Center next n lines
.ch cd		Flag **c** in .CHARS files as **d**
.ds		*Equivalent to: .ms 2
.dt **sym n**		Define sym to be a table of length n.
.ef # t		Define even footer line #
.eh # t		Define even header line #
.eq n	n=1	*Next n lines are equations
.fh t	line of underscores	Format of footnote demarcation line
.fi		*Fill output lines .fl c,n Set to flag output with a **c** in column **n.**
.fo # t		Equivalent to: .ef # t .of # t
.fr c	t	Controls footnote numbering: "t": reset each page; "f": continuous numbering; "u": numbering suppresed for next footnote
.ft		Delimits footnotes.
.gb xxx		"go back" to label xxx
.gf xxx		"go forward" to label xxx

Control Line Summary

Request	Default	Meaning
.he # t		Equivalent to: .eh # t .oh # t
.if f exp		File **f** inserted; value of **exp** assigned to **Parameter**
.in +n	n=Ø	*Indent left margin n spaces
.la xxx		Define label xxx
.li n	n=1	Next n lines treated as text
.ll +n	n=65	Set line length to n
.lm +n	n=Ø	*Indent left margin n spaces
.ma +n	n=4	Set top and bottom margins to n
.mc sym		Define sym to be a counter.
.mp +n	n=1	n-1 blank pages between output pages
.ms +n	n=1	*Multiple space of n lines
.m1 +n	n=4	Margin above headers set to n
.m2 +n	n=2	Margin between headers and text set to n
.m3 +n	n=2	Margin between text and footers set to n
.m4 +n	n=4	Margin below footers set to n
.na		*Do not right justify
.ne n	n=1	Need n lines; begin new page if not enough remain
.nf		*Don't fill output lines
.of # t		Define odd footer line #
.oh # t		Define odd header line #
.op		*Next page number is odd
.pa +n	n=+1	*Begin page n
.pc n	n=1	If n lines remain on the page, continue processing text and ignore the next .pc control line. Otherwise, save all the text and commands up to the next .pc control line and process it at the top of the next page.
.pi n	n=1	Equivalent to: .pc n .sp n .pc
.pl +n	n=66	Page length is n
.rd		Read one line of text from the terminal and process it
.ro		Roman numeral page numbers
.rm +n	n=65	Set line length to n
.rt		"Return" from this input file
.sk nn=1		Skip n page numbers before next new page
.sp n	n=1	*Space n lines
.sr sym exp		Assign value of exp to variable named sym
.ss		*Equivalent to: .ms 1
.st sym n,exp		Set entry n of table sym to be exp.
.tb +n,		Set a tab stop at each n in the list
.tr cd		Translate c into d on output

Control Line Summary

Request	Default	Meaning
.ts n	n=1	Process next input line only if _n_ is non-zero.
.ty xxx		Write **xxx** onto the terminal
.uc c	none	Sets underscore control char; if _c_ is omitted, underscoring disabled.
.un n	left margin	*Indent next text line **n** spaces less
.ur xxx		Substitute values of variable in **xxx** and then process **xxx**.
.wt		Read one line of text from the terminal and discard it
.*		Comment line; ignored
.~		Comment line; ignored, but written to .CHARS file

Built-in symbols

Only those symbols marked with an asterisk before their value are settable by the user. All symbols are numeric unless they are specified to be a string or a table. Control words and control arguments which afffect the values of the variables are indicated in parentheses: (x/y) indicates that x sets the switch to true (-1), and y sets if false (Ø); (a) or (a,b,c) indicates that it is affected by a, or by a, b and c.

Symbol	Value
Ad	*Adjust (.ad/.na)
AskHyphenations	*User is to be interrogated about word hyphenations.
CASECONTROL	*ASCII code of character signalling case conversion
Ce	*Number of lines remaining to be centered (.ce)
CHANGECASE	*Overall mode is to interchange upper- and lower-case letters.
CharsTable	*Translation table for .CHARS file [table] (.ch)
Charsw	*".CHARS" file is being created (character option)
Console	Reads one line from the terminal [string]
Date	Date of this invocation of MRUNOFF; format is mm/dd/yy [string]
Eq	*Equation line counter (.eq)
EvenFlagCol	*Column in which to flag even pages (.fl)
ExtraMargin	*Indent entire text this many spaces (margin option, indent option)
Fi	*Filling (.fi/.nf)
FileName	Name of primary input file [string]
Filesw	Output is going to a file (output option)
FlagChar	*ASCII code of character to be used to flag output lines (.fl)
Foot	*Footnote counter (.ft, .fr)
FootRef	*Footnote reference string to be inserted in footnote body [string]
Fp	*First page to print (reset each pass to "From")
Fr	*Footnote counter reset switch (.fr)
From	*Argument of "from" option

Built-in Symbols

Symbol	Value
Hyphenating	*Hyphenation switch (hyphenate option)
In	Amount to indent (.in, .lm)
InputFileName	Name of current input file [string] (.if)
InputLines	Line number in current input file
LinesLeft	Number of usable text lines left on this page
Ll	*Line length (.ll, .rm)
Lp	*Last page to print (reset each pass to "To")
Ma1	*Space above header (.ma, .ml)
Ma2	*Space below header (.m2)
Ma3	*Space above footer (.m3)
Ma4	*Space below footer (.ma, .m4)
Ms	*Spacing between lines (.ms, .ss, .ds)
MultiplePagecount	*Number of form feeds between output pages (.mp)
NestingDepth	Index into stack of input files (.if)
Nl	Number of last used output line
NNp	*Next page number (.pa, start option)
NoFtNo	*Don't number next footnote (.fr)
NoPaging	*Suppress page breaks (paginate option)
Np	Current page number (.pa, start option, re-set each pass from "Start")
OddFlagCol	*Column in which to flag odd pages (.fl)
OverprintCount	*Number of addtional times characters should be overprinted
PadLeft	*Pad from left end of line (.ad, complemented at end of each output line, set false at each break)
Parameter	*Passed argument betwen files (.if, paramter option)
Passes	*Number of passes left to make (=1 when printing is being performed) (passes option)
Pl	*Page length (.pl)
Print	*Print output ((Fp $\leq$ Np $\leq$ Lp) & (Passes $\leq$ 1))
PrintLineNumbers	*Source line numbers are being printed (number option)
Roman	*Number pages with Roman numerals (.ro/.ar)
Start	*Initial page number (start option)
Stopsw	*Pause between output pages (pause, stop and wait options)
TabStops	*Locations of tab stops [table] (.tb)
TextRef	*Footnote reference string inserted in main text [string]
Time	TENEX internal format of Date
To	*Last page to be printed (to option)
TrTable	*Translation table for output substitutions [table] (.tr)
Un	*Number of positions to decrease indenting (.un)

Built-in Symbols

Symbol	Value
UnderChar	*Character which is controlling underscoring (.uc)
Underscoring	*All characters are to be underscored (.uc, occurrences of double "UnderChar"s complement this)
Waitsw	*Wait before beginning output (wait option)
Widths	*Print width of output characters [table]

Hyphenation

By use of the "hyphenate" option, the user may request that the hyphenation routines attempt to break a word whenever the space available on an output line is less than the length of the next word (including attached punctuation, if any).

The hyphenation routines will split an already hyphenated word <u>only</u> at its hyphens. Beyond that, they do <u>no</u> syntactic analysis. They will interrogate the user for permissible hyphenations of every word that is a candidate to be split. When the user indicates the permissible hyphenation locations within a word, the word together with its hyphenations is entered into a dictionary so that if multiple passes are being done or if the word happens to be eligible again, the user will not be re-interrogated. The lookup in the dictionary for a match is a full string compare.

Before processing any of the input file, the user will be asked if he would like to pre-load the hyphenator's dictionary with a saved dictionary. If he does, the requested dictionary will be loaded and the user will not be interrogated about any words appearing in it. The format of a saved dictionary is an ASCII file containing words with embedded hyphens to indicate permissible hyphenation points. Tabs, carriage returns or form-feeds can be used to separate the words in the dictionary. All words should be entirely upper-case.

When during the course of processing the input file the user is interrogated for a hyphenation, the word in question will be typed out and the user should type in the same word, with hyphens to indicate where the word may be divided, or the user may type just a "." to indicate that the word should not be split. The user may edit his hyphenation with A, R or Q, and should terminate his input with a carriage return. Be careful that hyphenations are typed in upper-case; the hyphenator will always interrogate with words in that form.

At the end of each pass, if there have been any additions to the dictionary the user will be asked if he would like the accumulated dictionary saved. If so, the entire contents of the hyphenator's dictionary will be written out in a format suitable for later reloading.

If the user would like the capability of having words hyphenated as per a pre-loaded dictionary and at internal hyphens, but would rather not be interrogated about additional hyphenations, he should set AskHyphenations false.

MSG MANUAL

MSG was written by John Vittal for USC Information Sciences Institute.

MSG is a program for reading, writing, and subsectioning files which have a "message file format". It is very simple and straightforward to use. Commands are initiated by typing one character, which causes the program to type out the rest of the command name and wait for input from you.

Before the commands are described, there are a few general statements about how MSG works and some conventions used in describing the commands that you should know about. The prompt characters letting you know that MSG is waiting for a command character to be typed are "<-". When MSG is started up (by typing MSG<return> to the EXEC) it will first try to read your MESSAGE.TXT;1 file in your directory. If this file does not exist MSG will say so. If you were not connected to your login directory, MSG will try to find a MESSAGE.TXT;1 there. If that also fails, it will say so and wait for a command to be typed. If you have a MESSAGE.TXT;1, it will scan it and type out the header information (i.e. the date, from, and subject fields) for each message since the file was last read, preceded by a message number sequentially assigned by MSG. These message numbers are used in association with the various commands.

However, if you started MSG by typing MSG<space> to the EXEC, it will ask you for a file to be read. Typing an escape as the first character will cause MESSAGE.TXT;1 to be typed out, and confirmation requested from the user to ensure that that was what was intended. Once a file name has been specified and positively acknowledged, then the same information as described in the previous paragraph will be output to your terminal.

When reading a message file in MSG, either when starting up MSG or with the Read command described below, the file must be in the so-called "message file format". If MSG recognizes that the file does NOT conform to this format, you will be told so. However, you will be given the opportunity to keep everything that has been read so far, but NOT overwrite the 'bad' file. These two exceptional circumstances and some suggestions for getting around them are described at the end of this manual.

The following conventions and symbols are used in the command descriptions below. There are only five types of input MSG expects:

(1) a MSG command (or sub-command) character
(2) a message sequence specification
(3) a TENEX file name
(4) a confirmation character

(5) a local user name or remote site name
To abort output to the terminal type O (control-O). If MSG does not understand your input, it will return to command input mode, or reprompt you. The following are symbols and their associated meanings used in the command descriptions:

<FILE-NAME>
Stands for any TENEX file descriptor, including TTY: or LPT:. If you are requested to input a file name, the appropriate TENEX confirm will be given (e.g. [Old version]).

<MSG-SEQUENCE>
This input is prompted by the string (message sequence) in verbose typeout mode. A sequence of message numbers has the following format.

(1) Any single message number.
(2) Any two numbers separated by ">" or ":". This means message numbers delimited by the two outside numbers (e.g. 2>5 means messages 2,3,4,and 5 in that order). NOTE: if the first number is greater than the second number, it means the sequence in reverse order (e.g. 5>2 means messages 5,4,3, and 2).
(3) A pair of numbers separated by "-". This is so that the standard interpretation of the string "21-4" (that is not "21-24") means message numbers 21, 22, 23, and 24. Using this interpretation, the string and "24-1" is an error.
(4) Any sequence of the previous three types separated by commas. This is the way to group several non-adjacent messages together. For example: 1,3,5:7,10 means messages 1 and 3 and 5 through 7 and 10.

<MSG-SEQUENCE> of the types described above are ALWAYS terminated by <return>.

(5) However, there are special types of message sequences. All are determined by the first character that you type in the <MSG-SEQUENCE> stream. The following are the twelve possibilities:
   a. <escape> is typed, which causes the current message number to be echoed to you and the relevant process performed on that message only.
   b. <control-I> is typed, which causes the previous completely specified <MSG-SEQUENCE> to be echoed and processing performed on that message stream.
   c. R which stands for "Recent messages" only.
   d. O standing for "Old messages" only.
   e. A standing for "All messages" and which is equivalent to 1:(last message number).
   f. D standing for "Deleted messages". This is valid ONLY in the context of the Headers, Undelete, and Delete commands. Everywhere else, the headers of the deleted messages will be printed. Of course, you can delete the typeout of those headers by typing control-O.

g. U standing for "Undeleted messages".
h. I standing for messages in inverse order. This is the opposite of the A (for all messages) sub-command.
i. S for "Subject field search for string" which asks you to provide a string which will be used as a mask match on the subject field of the message headers.
j. F for "From field search for string" which is like S but searches the Author field of the message headers instead. NOTE: the header command prints the initial part of the To: line of the message (if it exists) is the message was sent by the login-directory. Therefore, to search for messages sent by yourself, specify the string "To:" rather than the login directory name.
k. E standing for "Examined messages", i.e. all messages which have been completely typed (with the T command) or listed (with the L command).
l. N standing for "Not examined messages", which is the opposite of the E sub-command.
m. L standing for the "Last message sequence" that was completely specified.

Types (i) and (j) require you to type a string terminated by <return>. Typing just a <return> (i.e. the null string) means that searching is not to be performed. Otherwise, the search will be performed on the string typed up to (but not including) the <return>. The string you type must be an exact match to some substring of the appropriate field, but all alphabetic characters are treated as being upper case. (Note: carriage-retuns in the subject field of the header listing are ignored.)

(6) (Note: This is an experimental feature which may change in the future.) If you type comma or "M" as the first character of the message sequence that you are specifying, you will be able to specify more than one of the options drawn from the first five items mentioned here. You will then be entered into a sub-command mode. Any of the standard message sequences are acceptable as input. To terminate the specification of the list of message sequences, just type a carriage return in response to the prompt. If you wish to abort the acquisition at any time, type "Q" (for Quit) or control-N (N̲). To abort the acquisition of a single message sequence (like 3:14), type rubout. Typing rubout at the sub-command level (i.e. at the prompt without typing anything first) will have the same effect as typing control-N.

The default message sequence is 'All messages'. Any message sequence specified causes an intersection to be taken between that single message sequence (like 'Examined'), and the previous total. For example, the sequence:

```
<- Headers ,
<<- Examined
<<- From string: SYSTEM
<<-
```

would cause only the headers corresponding to messages from SYSTEM which have already been typed to get listed on your terminal.

If you just want to add a message sequence to the list, preface the actual message sequence with a "P" (for Plus) or "+". If you want to just subtract a message sequence from the list, preface the actual message sequence with an "M" (for Minus) or "-". For example,

```
<- Headers Multiple message sequences
<<- Examined
<<- From string: SYSTEM
<<- Plus: Subject string: MSG
<<- Minus: Deleted
<<-
```

will list the headers for all undeleted messages about MSG or which are examined messages from SYSTEM. No further associations between msg-sequence specifications are currently allowed.

In the command format below, everything that the program types will be lower case and everything you type will be in UPPER CASE. This is not the case when using MSG, but is used here for clarity.

MSG COMMANDS

<- Headers (message sequence) <MSG-SEQUENCE>

The headers for messages will be typed out for those messages defined by the message sequence typed. Headers corresponding to deleted messages have an asterisk printed before the header for that particular message. The headers for recent messages are preceded by a plus sign (+); messages which have not yet been typed are preceded by a minus sign (-), and deleted messages are preceded by an asterisk (*). If the message was sent by the user of the LOGIN directory, the initial part of the To: field of the message will be printed in the author field of the header, if the To: field exists in the message. In order to get the length of the message typed out along with the header, use the I command (which stands for Inclusion of length in header).

<- Delete (message sequence) <MSG-SEQUENCE>
This command will indicate (by a preceding asterisk) in the header information for the messages specified by <MSG-SEQUENCE> that those message are deleted. NOTE: This command marks each message in the actual message file indicating that it is deleted. If you reread the file for some reason, the messages will still be marked (and treated) as deleted (but not expunged). This command does however effect message numbers specified in later commands in the following way. If you have deleted message number 5 and then try to "Type" or "Put" message number 5 either directly or implied by the use of the ":" option, the deleted messages will NOT be included.

<- Undelete (message sequence) <MSG-SEQUENCE>
Of course! If you can delete a message, you certainly ought to be able to undelete it. This command undoes the action of the Delete command for the messages specified by this <MSG-SEQUENCE>.

Commands to See and Move Messages

<- Type (message sequence) <MSG-SEQUENCE>
This command will type on your terminal the messages specified by <MSG-SEQUENCE>. All messages which are completely typed are treated as having been 'examined'.

<- Put (message sequence) <MSG-SEQUENCE>
into file name: <FILE-NAME>
This command will put the messages specified by <MSG-SEQUENCE> into the file specified by <FILE-NAME>. If the file does not exist, it will create that file and write the messages into it. If the file already exists, it will append the messages to the messages already in the file. This command is useful if you want to keep separate files containing messages concerning different topics.

<- Move (message sequence) <MSG-SEQUENCE>
into file name: <FILE-NAME>
This command is a convenient combination of the Put and Delete commands. As the messages are put into the file, they are marked for deletion. If any of the messages are already deleted, you will be informed, and those messages will NOT be moved to the file.

<- List (message sequence) <MSG-SEQUENCE>
on file: <FILE-NAME>
This command lists all the specified messages on the file

specified. All messages specified by the <MSG-SEQUENCE> are treated as having been examined (typed). If you are listing more than one message, a preface page on the file will be created which contains the headers for those messages. You will be asked if you want each message on a separate page. This command is intended to allow a user to obtain a reasonable hard copy listing of some messages. (Note: the preface page of headers might have the length of each message included depending on the setting by the I(nclusion of length ih header) command.)

Commands to Update Your Message Files

<- Overwrite old file <FILE-NAME> [confirm]
This command will overwrite the current file (specified by <FILE-NAME>), reflecting the fact that you have deleted messages. That is, if you delete message 2 and then "overwrite" your file, message 2 will disappear from that file. It also rereads your file, renumbering your messages. You are warned if any unexamined messages (which are also not deleted) exist in the file that you are overwriting.

<- Quit [confirm]
This command returns you to the TENEX EXEC without rewriting any file (almost equivalent to typing control-C). You are warned if any unexamined messages (which are also not deleted) exist in the current message file.

<- Exit and update old file <FILE-NAME> [confirm]
This command is another way to Overwrite your old message file, but instead of rereading the file it returns you to the TENEX EXEC. This is equivalent to doing an overwrite followed by a Quit, but without the overhead of rereading the file. You are warned if any unexamined messages (which are also not deleted) exist in the file that you are overwriting.

<- Write file <FILE-NAME> sorted by message arrival time
This is similar in nature to the Overwrite command, except that the messages are sorted into ascending sequence by their arrival time before the overwriting is attempted. The file is then rescanned. You are warned if any unexamined messages (which are also not deleted) exist in the file that you are sorting.

Commands to Read Other Message Files

<- Read file name: <FILE-NAME>
You can use MSG on any file which has a "message format." This means you can peruse or modify files created with the "Put" or "Move" commands (but NOT the "List" command). If, for example, you have a file containing messages pertaining to MSG problems, you can read it to make sure you've taken care of them. Read is the command which lets you read files other than your standard mailbox, file MESSAGE.TXT;1. It also prints out the recent header information for that file. If that file has old messages which have not yet been 'examined', you will be informed. You will also be told if any of the old messages in the file are deleted.

Commands to Sequence through the Messages

<- Current message is nn of mm messages.
in file: <FILE-NAME>
This command tells you (1) the number of the current message, (2) the total number of messages, and (3) the file name of the currently active file. The current message is either the last message typed on your terminal or, if you have not typed one yet, either after the last message if the file had no recent messages, or before the first recent message. This command will let you know where the Next and Backing up commands will start, i.e. the first message they will type if used. Finally, it will tell you what the currently active message file is.

<- Go to message number: <NUMBER>
This will allow you to change the Current message number explicitly. If <NUMBER> is not in the range of acceptable numbers (i.e. it is less than 1 or greater than the number of messages in the file), or you did not type a number, you will be told and the Current message number will not be changed. However, there are several other options which are specified by the FIRST character typed:
a. E for the end of messages (the last message)
b. L for the last message (same as E).
c. B for the beginning of messages (message number 1)
d. escape (alt-mode) for current message number

<- Next message is:
This command types the message following the current message (if one exists) and sets the "current message" to be that message. Deleted messages will not be typed, but the "current message

number" will still be incremented.

<- <line feed>
Same as Next. Types the message following the current message, and sets the current message to be that message.

<- Backing up -- previous message is:
This command always types the previous message (i.e., Current message number - 1). It is the inverse of the Next command. It always decrements Current message number.

<-
This is equivalent to the Back command. It types the previous message and sets the current message to be that message.

<- H
The <control-H> (or New-line) command is equivalent to the Back command. It types the previous message and sets the current message to be that message.

Other commands

<- Verbose
This is a binary switch which causes the program to go into either 'Short typeout mode' or 'Long typeout mode', and tells you which is the setting that it changes to. The default is 'Short typeout mode'. Long typeout mode gives additional prompting regarding what is expected to be typed in.

<- Koncise
This is a binary switch which causes the program to go into either 'Concise typeout mode' or 'Short typeout mode' (the default), and tells you which is the setting that it changes to. Concise typeout mode shortens some of the typeout that MSG gives when it is interacting with the user. It is meant for 'advanced' users only.

<- Inclusion of length in header
This command is a binary switch which causes the program to go into a mode where header listings caused by the Header command will have the number of characters in the message included as part of the subject field. The default is that the length will

not be included. Note that when you read a file initially, the length of 'recent' messages will always be included in the initial listing of recent headers.

<- Zap profile [Confirm]
The Zap profile command will allow you to set up a user profile file for yourself without having to know the format of such a file. For the time being, the profile information will be limited. Typing control-N will exit you to the command level of MSG. Typing E at any point will 'Exit' the dialogue and ask you if you want the changes made permanently. At any point, type "?" to determine the appropriate responses. The following is a summary of the questions posed:

1. Normal, Verbose or Koncise typeout mode?
2. Always include the length of messages in all headers listings?
3. When in SNDMSG (from any of the Sndmsg, Forward or Answer commands), when you type control-N, do you want to abort Sndmsg without confirmation?
4. Do you want to be required to confirm all commands with a single carriage return?
5. Do you want to be told that some messages have been 'not-examined' whenever you try to quit MSG (by any of the Quit, Overwrite, or Exit and Update commands)?
6. Do you want to receive copies of your 'answers' to messages?
7. If the answer to (6) is yes, then you will be asked if you want to save all your 'answers' on the file SAVED.MESSAGES.
8. Do you want a list of headers for all messages:
   a. being deleted with either the "Move" or "Delete" commands
   b. being moved with the "Put" command
   c. being listed with the "List" or "Xerox" command
   d. being marked or unmarked with the "'" or "-" commands.

At the end of the dialogue, you will be asked if you want these changes in mode settings to be made permanent. If you answer 'Y' then each time you start up MSG, the settings of the modes noted here will be set to the values you indicated. Otherwise, the settings are set only for this session. They are NOT permanent, and can be changed any time.

<- : (prints current time and date)

<- ' Mark messages as examined (message sequence) <MSG-SEQUENCE>
This command will mark all the messages specified by <MSG-SEQUENCE> to be "examined", so MSG will think they have been typed or listed even though they may not have been.

<- - Unmark messages to be Not examined (message sequence)
<MSG-SEQUENCE>

This command will mark all the messages specified by <MSG-SEQUENCE> to be "NOT examined", so MSG will think they have NOT been typed or listed, even though they might have already been seen.

<- ; <COMMENT>

This command is mainly intended to allow you to talk with somebody over a link while you are in MSG. It eats all characters except <return>, control-Z (Z) and control-N (N), which return you to the command level of MSG. Two other characters have special effects. <delete> (<rub-out>) will type the string ' XXX ' and is useful in indicating that the previous word (or phrase) should be ignored. <line-feed> will cause effectively a carriage return and tab sequence to be typed. This way you can type more than one line of text. NOTE: the standard TENEX editing characters (e.g. control-A) are treated as any other character and perform no special function.

Command to Run Other Programs

<- Sndmsg [confirm]

This command will start up SNDMSG and give control of the terminal to it. When SNDMSG is finished (i.e. when you have sent the message), it will turn control back to MSG in the same state as it was before you sent the message. Control-N (N) will ask if you wish to abort. If you provide a positive confirmation, then you will be returned to the top level of MSG. Otherwise, you will be returned to SNDMSG.

<- Answer message number: <MESSAGE-NUMBER>
Reply to those whom the message is: <ANSWER SUB-COMMAND>

This facility allows you to send a message to the sender of a message, and (at your discretion) those people to whom that message was sent, without having to type their addresses to Sndmsg.

The <ANSWER SUB-COMMAND> can be any of the following:
- F -- From (indicating the sender of the message only)
- T -- To (indicating the sender of the message and those addresses in the To: list)
- C -- Cc (indicating all recipients of the original message in addition to the sender in the message)

Typing anything else aborts the command.

The <MESSAGE-NUMBER> can be any argument that the Go command takes:

a. a message number
b. E for the end of messages
c. L for the last message (same as E).
d. B for the beginning of messages (message number 1)
e. escape (alt-mode) for current message number
f. <return> for current message number.

The header of the message specified is also typed so that you may be sure you are answering the correct message. In fact, the header is typed after you have specified the message number, but before you are asked to supply the sub-command.

When prompted for additional addresses, any that you specify will be passed to SNDMSG as part of the cc: list. Some of the SNDMSG conventions are NOT implemented. These are the control-B feature which allows specification of a file, and the feature which allows you to specify a global host name (which spreads across several user names). Also, control-N aborts the Answer command! Local user names and remote host names are checked for validity.

An attempt is made to insure that all addresses are valid (i.e. all host names on remote addresses, and user names on local addresses), and that no duplications are present. If clarification is necessary from the user, you may be asked some questions. If these questions are posed, all type-ahead is deleted. If relevant, MSG will issue a warning if either the To: or cc: destination fields of the message have a destination list as part of the field (like LISP-USERS:). When control is given to you to type your answer, you will be typing to the message acquisition portion of SNDMSG (i.e. that part which normally would prompt you by typing "Message (? for help):"). Control-N (N̲) will ask if you wish to abort. If you give [illegible]itive confirmation, then you will be returned to the top level [illegible] MSG. Otherwise, you will be returned to SNDMSG.

There are two relevant profile mode settings. One is whether you wish to receive copies of the answers you send. It is initially assumed that you do. If you do not want copies of the replys you send, then your name will not appear in any of the destination lists unless you specify it as part of the additional carbon-copy list. However, if you do want copies of your messages, you will always receive one. In addition, if you have also indicated in your profile that you want all your responses to go to a file called SAVED.MESSAGES, then if that file exists, your responses will go in that file and NOT into your MESSAGE.TXT.

However, if you do not always want your answers to go to SAVED.MESSAGES, but do want copies of your answers, if there is a file named SAVED.MESSAGES in the login directory, you will be asked if you want your copy of the message to go to that file. If a positive response is given, then the login directory name will NOT appear in the destination lists.

<- Forward (message sequence) <MSG-SEQUENCE>
This facility will allow you to send copies of messages you have received to other people. The headers of the messages being forwarded are typed, after which you will be asked to provide the subject of this forwarded message. Then it will hand SNDMSG the subject and those messages you want forwarded, and leave you in SNDMSG in such a way that the message being forwarded can be edited, or your own comments added. You will be left in SNDMSG as though you had typed the forwarded message in yourself. When done, type a control-Z and then specify, in the standard way, to whom the mail is going. Once in SNDMSG, typing control-N (N) will ask if you wish to abort. If you give a positive confirmation in the standard way, then you will be returned to the top level of MSG. Otherwise, you will be returned to SNDMSG.

<- Jump into lower fork running: <FILE-NAME>
This command is an escape in MSG in case you wish to run another program such as TECO, PUB, the EXEC, and so on. It searches directories to try to find the program you are asking it to run. The search list is, in order, <SUBSYS>, <SYSTEM>, your connected directory, and the login directory (if it is different from your connected directory). This way, you can run EXEC without having to type the complete information (i.e. <SYSTEM>EXEC.SAV).

If you decide to leave the lower fork, but want to continue it at a later time, all you need do is type an escape as the first character of the file name you are requested to provide. This will cause the old file name (preceded by an appropriate message) to be printed, and then you will be asked to confirm in the standard way. If you provide a positive confirmation, you will be asked if you want to continue or start that program. Typing 'C' for continue will put you back in the lower fork at the place where you exited; typing 'S' for start will restart the program.

<- Xed (editor) [confirm]
This command will start up XED (a text editor written at ISI). It has the capability to give SNDMSG the text built while in the editor as the body of the message. When you "Quit" XED you will return to MSG. Each additional time that you execute the XED command, you will be returned to the SAME copy of XED that you previously left with the XED "Quit" command (the old text buffers

are left intact).

<- $\underline{E}$xec [confirm]
When you type control-E, the program will type "Exec" to you and ask for confirmation. This command is intended to give you a new copy of the EXEC with a minimum of hassles To leave that EXEC and return to MSG, type "QUIT". If you decide that you want a copy of the EXEC again, and you use this command, you will be given the same EXEC with all of your context intact.

This completes the list of MSG commands. There is only one item left to mention.

Receiving New Messages While Using MSG

MSG, on typing a command or returning from the execution of a command, checks to see if your currently active message file usually MESSAGE.TXT;1, has been written into. If it has, it prints out that fact and the headers for the new messages. The "current message number" is not modified. It then executes your command or returns to command mode, accordingly

Command Summary

Cmnd. Char.	Meaning
A	Answer message number: <MESSAGE-NUMBER> Reply to whom the message is: F - From <return> -- same as F T - To list plus original sender C -- Cc list plus To: list plus original sender
B	Backing up - previous message is
$\overline{}$	Same as Backing up
$\overline{H}$	Same as Backing up
$\overline{C}$	Current message is nn of mm messages in file: <FILE-NAME>
D	Delete (message sequence) <MSG-SEQUENCE>
$\underline{E}$	Exec [confirm]
$\overline{E}$	Exit and update old file <FILE-NAME> [confirm]
F	Forward (message sequence) <MSG-SEQUENCE>
G	Go to message number: <MESSAGE-NUMBER>
H	Headers (message sequence) <MSG-SEQUENCE>
I	Inclusion of length in header
J	Jump into lower fork running file: <program name> [confirm]
K	Koncise -- provides shorter prompting

```
     L   List (message sequence) <MSG-SEQUENCE>
         on file name: <FILE-NAME>
     M   Move (message sequence) <MSG-SEQUENCE>
         into file name: <FILE-NAME>
     N   Next message is:
   <lf>            (line feed) same as Next message is:
     O   Overwrite old file <FILE-NAME> [confirm]
     P   Put (message sequence) <MSG-SEQUENCE>
         into file name: <FILE-NAME>
     Q   Quit [confirm]
     R   Read file name:  <FILE-NAME>
     S   Sndmsg [confirm]
     T   Type (message sequence) <MSG-SEQUENCE>
     U   Undelete (message sequence) <MSG-SEQUENCE>
     V   Verbose -- provides more prompting
     W   Write file <FILE-NAME> sorted by message arrival time [confirm]
     X   Xed  [confirm]
     Z   Zap profile [confirm]
     '   '   Mark messages as 'examined' (message sequence) <MSG-SEQUENCE>
     -   -   Unmark messages to be NOT 'examined' (message sequence)
                                               <MSG-SEQUENCE>
     :   :   (the time and date is then printed)
     ?   ?   Type command character for its description, ? for summary
     ;   ;   Comment -- <return> or Z returns you to command level
```

Abort commands on typein and terminal output with control-N (N). Confirm with Y or <return>.

Errors While Reading a Message File

When reading a file in MSG (either at startup or with the 'Read' command), the file MUST be in the so-called "message file format". If MSG recognizes that the file does NOT conform to this format, you will be told so. The following are the circumstances which might cause the file to become unreadable, and some suggestions for getting around the problems.

> The file is a message file (that is, one or more valid messages have been read from it), but somewhere in the middle it does not conform to the message file format. It could be: (1) It has a hole in it. Read the file with a text editor to get rid of the hole, and write it back out, and reuse MSG. Try this first. If this doesn't work, MSG will give you an error at the same place. Then you can try the second suggestion: (2) If suggestion 1 didn't work, then the file has internal byte counts which do not match the actual file. Either you used a text editor on your message file changing the number of bytes but not the byte counts or your file was mysteriously altered. The date of a message could not be read. Either the byte count for the last message read was wrong, or there is junk between the last message read

and the one with the error. Using some editor, find the last message read. The first line of that message contains a date-and-time followed by a byte count indicating how many characters are in the message body starting on the following line. Skip that many characters of the message body. You should be at the date-and-time line of the next message. If there is junk there, delete it. Otherwise, try to fix the count so it is pointing at the date-and-time of the next message.

The beginning of the file does not conform to the message file format. It could be: (1) the file is not a message file -- sorry, we can't help you there. (2) It is a message file with a bad first line -- probably a blank line. Read the file with a text editor. If the second line begins with a time and date then delete the first line and reuse MSG on the new file. (3) It is a message file with a hole at the beginning. Read it with a text editor to get rid of the hole, write it out and reuse MSG.

MINCOP

MINCOP asks the user to respond to the request:

"TYPE C TO COPY, V TO VERIFY, L TO LIST ONLY."

COPY makes a copy of a MINIDUMPER format tape. VERIFY compares two MINIDUMPER tapes and complains if there are any differences. LIST ONLY simply lists the files on a MINIDUMPER format tape (which requires a read pass on the entire tape).

The user is also asked to type the tape unit numbers and the densities of the tapes involved (L for Low(200), M for Medium (556), H for High (800)).

The MINIDUMPER Format tape being read is checked for both tape errors and format errors. MINCOP will not copy arbitrary format tapes!

MTACPY

MTACPY is a program which reads 7-track magtapes in any format and writes tapes in DECsystem10 format.

MTACPY initializes with:

```
     MAGTAPE UNIT NO.=
```

The user should respond 0 or 1 depending on where his tape is mounted. After moving the tape to its load point, MTACPY asks:

```
     USE 556 BPI?(Y OR N):
```

An N response here causes an additional query:

```
     DESIRED DENSITY (200 OR 800):
```

After the density response, MTACPY asks:

```
     NORMAL ODD PARITY?(Y OR N):
```

Most tapes are odd parity, with the exception of BCD card image tapes, which are usually even parity.

The next question is:

```
     TO OR FROM MAGTAPE?  (T OR F):
```

The response to this specifies which way information is to be moved. If the answer was "T", it would ask:

```
     SOURCE FILE(S):
```

At this point, it accepts a list of file names separated by commas and terminated by carriage return. File names may contain "*" in the name or extension field to specify automatic iteration over all files which match the other fields.

For each file transferred, MTACPY types out the number of six-bit bytes written onto the tape. When the list of source files is exhausted, (all have been transferred), it again asks:

```
     SOURCE FILE(S):
```

in order to collect another list of source files. If the user supplies an empty list by typing carriage return, the program asks:

```
     DONE?(Y OR N):
```

An N response returns to ask for another list of source files. A Y response finishes up the magtape by writing ten successive end-of-file marks, then rewinding the tape.

If data were to be read from the tape, the answer to the question:

    TO OR FROM MAGTAPE(T OR F):

would have been "F". Then the program asks:

TARGET FILE: and the user should respond with a writable file name. If the user responds with, for example, file name "XYZ", the program will move an image of the tape data into file XYZ. In addition, to preserve formatting information, MTACPY creates an auxiliary file with name XYZ.RECSIZ, which is an ASCII file which reports the number of six-bit bytes read from each successive record of the magtape. Programs which do later code conversion of each record make use of this auxiliary file to define record boundaries.

It is possible to skip over uninteresting files on the tape by responding with null target file names, i.e. carriage returns. Answering the question:

    TARGET FILE:

with three carriage returns will cause the first three files on the tape to be skipped over, then the question is repeated.

For each file transferred, MTACPY reports the number of six-bit bytes read from the tape.

The logical end of tape is denoted by two successive end-of-file marks. On encountering this, the program asks:

LOGICAL END OF TAPE.KEEP READING?(Y OR N)

A "Y" response here causes the program to continue. An "N" causes the tape to be rewound, and the program exits.

EXAMPLE:

```
@MTACPY

MAGTAPE UNIT NO.=Ø
USE 556 BPI?(Y OR N):Y
NORMAL ODD PARITY?(Y OR N):Y
TO OR FROM MAGTAPE(T OR F):F
TARGET FILE:ABC[NEW FILE]
16ØØ (DECIMAL) SIX-BIT BYTES.
```

```
TARGET FILE:
LOGICAL END OF TAPE.KEEP READING?(Y OR N)N
EXIT.
^C
@
```

At this point, the file ABC.RECSIZ might contain, for example:

```
DECIMAL LENGTH OF EACH RECORD IN SIX-BIT BYTES

80      80      80      80      80      80      80      80      80      80
80      80      80      80      80      80      80      80      80      80
```

NOTIFY

Notify is a program which allows for "canned" messages. Thus often used messages can be prepared in advance of the time of need. Notify allows the composition of messages with all the editing capabilities of SNDMSG. A description of commands follows.

APPEND TO CURRENT MSG - This allows the user to append to the currently buffer message. This includes "canned" and composed messages.

COMPOSE MSG - This clears the current buffer and allows the user to type up a message.

GET CANNED MSG - Brings a "canned" message into the buffer. It will be appended to any message already there. These canned messages are kept in the file CANNED.MSGS. The file format is discussed later.

LIST CANNED MSG´S - Types a list of the current canned messages. What is listed is a summary of the actual message. The number is an index for use in the GET command.

PRINT CURRENT MSG - Prints the currently buffered message (that which will be sent) on the user´s teletype.

SEND MSG - Sends the message to the desired set of terminals. Two questions will be asked here, 1) Emergency (Y or N)? This is asking how important the notice is. If answered no, those users who have REFUSE LINKS on will not receive the message. Answering yes will cause all users to get the message. 2) TTYS: This is as before, either a list of tty´s or a -1 for all teletypes on the system.

The format for the CANNED.MSGS file is: MESSAGE SUMMARY<CRLF> MESSAGE TEXT<Z> ...... MESSAGE SUMMARY<CRLF> MESSAGE TEXT<Z> The text of each message can be any number of lines and contain any characters except Z (control z). The program can currently handle up to 30 canned messages.

PA1Ø/5Ø - Compatible Operation for 1Ø/5Ø CUSPS

In order to make immediate use of the large body of existing user programs written for the 1Ø/5Ø system and to accommodate future distributions of CUSP programs from DEC, TENEX provides facilities for operating in a 1Ø/5Ø compatible fashion. Under such operation, most existing user programs (including CUSPS) can be assembled without change, loaded, and run. In fact, it is possible to RUN immediately most SAVED binary DECtape files.

Limitations

There are two types of limitations on the programs to be run under 1Ø/5Ø compatibility: system call (UUO) limitations and file system limitations.

Much of the information maintained by the TENEX monitor is kept in tables completely different from the tables of the 1Ø/5Ø monitor. It is not practical to provide complete translation of the TENEX monitor tables into 1Ø/5Ø format, so the GETTAB, PEEK, SPY, LOGIN, and LOGOUT UUO's are not permitted, and trap to an error routine.

These restrictions mean that the 1Ø/5Ø Support Cusps (LOGIN, LOGOUT, REACT, FAILSAFE, MONEY, CHECKPOINT, and COMPILE) are not supported under 1Ø/5Ø compatibility. In fact, these programs are superseded by the TENEX EXEC, BSYS (file backup system), CHKPNT, and ACCT1Ø (accounting system).

The second major limitation on compatibility operation is due to the fact that the TENEX file system is different from 1Ø/5Ø; in particular, the user file directory is not available as a ordinary file (c.f. UFD in 1Ø/5Ø). Programs which rely on reading sequentially through such a file, such as PIP directory listing, are therefore not supported. PIP is, however, superseded by the TENEX EXEC.

Implementation

Since TENEX itself makes no use of the MONITOR UUO calls (opcodes 4Ø through 77) these are reserved for compatibility operation. When any program executes one of these calls, it is defined as a compatibility program and the TENEX monitor notes this in the PSB
# of that fork. The compatibility support code is then mapped from
# the file <SUBSYS> PA1Ø5Ø.SAV into pages 7ØØ through 717 of that
# fork's address space.

RECENT expansions of the compatibility package have made use of two of the Terminal Interrupt channels of the PSI system. Unfortunately, the choice of channels conflicted with existing programs which share the PSI channel table with the compatibility package. To remedy this we took a survey of known users in this category to determine what channels were mutually available. As a result of this survey we changed over to using channels 3Ø and 31 (decimal) in the current compatibility package, and will define that future expansion will use the group from 3Ø through 35.

Note that this should have no effect on most programs, which are written strictly for TENEX or strictly for 1Ø/5Ø operation. The only problem is with those attempting to use the PSI system <u>AND</u> the compatibility package.

The initial UUO is trapped by the monitor into the second location of the entry vector specified in PA1Ø5Ø.SAV. Subsequent
UUO´s are trapped into the first location of the entry vector.
During UUO calls, 4Ø is copied to the location pointed to by
# entry vector 14. The return PC is stored in the location pointed
# to by entry vector 15. (See SCVEC) in the JSYS Manual)

The UUO calls which are not supported give an "ILLEGAL UUO AT LOCATION" message and halt the fork. All other UUO´s are simulated correctly.

Since addresses >7ØØØØØ are not accessible in most medium sized 1Ø/5Ø systems, merging compatibility code into the user´s fork poses no new limitation. The compatibility code itself, of course, is reentrant and shared.

<u>TECHNICAL INFORMATION - INSTALLATION INSTRUCTIONS</u>

THE SOURCE FOR PA1Ø5Ø.SAV is <SOURCES>PAT.MAC. This code has a
´PHASE 7ØØØØØ´ around it so it will load into the low segment
# using LOADER. After completion of loading, enter DDT and start
# at LINIT. This first BLT´s the symbols down on top of the
# program, PMAPS the previous compatibility code (pages PATORG to
# 777) out of existence, then BLT´s the new program (plus symbols)
# up to PATORG. The entry vector is declared and an updated symbol
# table pointer is copied into DDT, and control returns to the
# EXEC.

At this point, it is convenient to SSAV pages 7ØØ to 777 on PAT.SAV, which provides a copy of the program with symbols and DDT for debugging purposes.

To put up this new compatibility package as the current operating version, enter DDT and start at MAKEPF. This SSAVEs the code with read and execute permission only as a new version of PA1Ø5Ø.SAV. This can then be placed on <SUBSYS> as PA1Ø5Ø.SAV.

PAL10

PAL10 offers several advantages over the antiquated PALX assembler. Some of these improvements are:

A. PAl10 runs in 4K as compared to 7K for PALX.

B. Has a binary search on symbol table which significantly reduces processor time for any given assembly.

C. Stacks symbol table listing in four vertical columns per page, reducing listing time and size.

D. Has conditional assembly and literals.

E. Two flavors of text handling.

F. Assembler generated literals or links have been modified to flag the generated statement with a hash mark next to the object code and be counted as a link and not as an error.

PAL10 CODES

PSEUDO	OP	CODES
*EXPUNGE	DELETE ALL SYMBOLS EXCEPT PSEUDO OPS	
*FIXMRI	ARG=X000 DEFINE MEMORY REFERENCE INSTRUCTION HAVING VALUE "X000"	
*FIXTAB	ESTABLISH ALL SYMBOLS CURRENTLY DEFINED AS ASSEMBLER SYMBOLS	
*NOTE, PAL10	INITIALIZES ITS SYMBOL TABLE BEFORE EACH ASSEMBLY I.E. FIXTAB, FIXMRI, AND EXPUNGE FUNCTIONS ARE VALID ONLY FOR THE ASSEMBLY IN WHICH THEY OCCUR	
ZBLOCK	ARG	GENERATE BLOCK OF ZEROES
DECIMAL		CHANGE TO RADIX 10
OCTAL		CHANGE TO RADIX 8
DEFINE		DEFINE MACRO
*DTORG		GENERATE DECTAPE ORIGIN
DUBL	ARG	ACCEPT DOUBLE PRECISION
ENPUNCH		ENABLE BINARY OUTPUT
NOPUNCH		DISABLE BINARY OUTPUT
FIELD	ARG	GENERATE FIELD ORIGIN

CONDITIONAL ASSEMBLY PSEUDO OPS

IFDEF TAB <ARG> ASSEMBLE "ARG" IF "TAG" IS DEFINED.

```
# IFNDEF    TAG <ARG>  ASSEMBLE "ARG" IF "TAG" IS NOT DEFINED.
# IFZERO    TAG <ARG>  ASSEMBLE "ARG" IF "TAG" IS DEFINED AS ZERO.
# IFNZRO    TAG <ARG>  ASSEMBLE "ARG" IF "TAG" IS DEFINED AS NON-ZERO.
#
# PAGE       Automatic origin at beginning of next core page
# PAUSE      NO-OPERATION
# TEXT      /ARG? Generate text table.  "/" is the next delimiter.
#            If number of characters is odd, table will be
#            terminated with a 6 bit zero, if even table will
#            be terminated by a 12 bit zero.
# XLIST      Toggle listing flip flop.
#
# Literal Facility
#
# [ARG]      Defines a location (determined by assembler) on page Ø
#            which contains "ARG".
# (ARG)      Defines a location (determined by assembler) on current
#            page which contains "ARG".
#
# Operators
# ---------
# !         Logical Shift 6-bits left
#           Logical And
# *         Integer Multiply
# <>        Special, Alpha only, text mode when not used in Macro
#           definitions or conditional assembly delimiters.
# "         Generate a word which contains the eight bit ASCII
#           equivalent of the next character.
#
# Error Indicators
# ----- ----------
# C         Illegal Character Error (this includes comment field)
# D         Illegal Redefinition
# G         Link Generated
# I         Illegal Indirect
# L         Literal error (overlap)
# M         Macro Error
# N         Number Error
# O         OP code error
# P         Phase or Multiply defined error
# Q         Questionable format?
# T         Illegal Text Character
# U         Undefined Character
# Z         Page Zero Literal or Exceeded Error
#
# COMMAND STRING SWITCHES
#           A     Advanced Magtape One File
#           B     Back Space Magtape One File
#           C     Enable "CREF" Format (Switch must be set
#                 before listing device)
#           D     Enable DECtape Origins i.e. "DTORG"
```

#	M	Suppress Symbol Table Output
#	N	Suppress Errors on TTY
#	T	Skip to Logical End of Tape
#	W	Rewind Magtape
#	Z	Zero DECtape Directory

#
# In addition to the command string switches mentioned
# above, there are a group of special switches which are designed
# to tailor the assemblers symbol table to match up with other PAL
# assemblers. These switches modify only the symbol table
# including Pseudo-op codes. Operators including and ! remain as
# their current function.
#

#	/P3	PAL III
#	/P8	MACRO 8
#	/PD	PALD
#	/PS	Super PAL or PAL 8

#

PCSAMP

A program to measure the operation of other user programs.

PCSAMP is used to take runtime statistics on another user program by sampling its program counter at fixed intervals. The sampled PC values are sorted in ascending order and written to a file. By comparing this file with a LOADER map of the user program, one can get an idea of the relative runtime spent by the user program in each of its subroutines, thus discovering bottlenecks in the program.

Upon starting PCSAMP asks:

PROGRAM TO BE SAMPLED: (answer with any RUNable SAV file)

FILE TO STORE SAMPLE PCS: (any writable ASCII file)

SAMPLING INTERVAL (MSEC): (20 to 50 is reasonable)

Then PCSAMP types

SAMPLING MAY BE FINISHED BY CNTL-Z.

and starts up the requested user program. Type input to the program in the normal fashion. If the program terminates with a HALTF, the sorting and writing out of the PC samples happens automatically. If the program instead hangs waiting for more input, type a control-Z to finish the sorting and outputting of the sample PC values.

The result is an ASCII file which may be LIST´ed or COPY´ed to LPT:.

PPL

PPL is an interactive, extensible programming language. It was designed by T. A. Standish and implemented for the PDP-10 by Standish and E. A. Taft at Harvard.

The version of PPL on TENEX consists of a typeless conversational language similar to Iverson´s APL in which the Iversonian mechanics of arrays have been subtracted out, and to which data definition facilities as well as operator definition facilities have been added. PPL´s conversational mechanics include the abilities to trace running programs, to set and remove breakpoints, to write programs that converse with users, and to edit the text of programs. Running time is proportional to the demand for computation, and, in particular, trivial requests are satisfied rapidly. Defined operations may be associated with unary and binary operators of the user´s own choosing, including redefining the meaning of operators given in the initial state of the language.

The program below is an example of PPL programming using both the operator definition capabilities and the data definition capabilities of PPL to define a small language extension that differentiates formulas. Formulas are tree-like objects that constitute an example of structured data. The example reveals how PPL can be used to write programs that manipulate structured data and how the normal syntax for arithmetic expressions can be shared for new types of arithmetic, such as formula arithmetic. In the following transcript, comments are preceded by three dashes. Abbreviations are as follows:

FORM means formula
UF means Unary Formula
BF means Binary Formula
LO means Left Operand
OP means Operator
RO means Right Operand
INT means Integer
CHAR means Character

```
                    ---Data Definitions

$FORM = UF!BF!ATOM
$UF = [OP:CHAR,RO:FORM]
$BF = [LO:FORM,OP:CHAR,RO:FORM]
$ATOM = INT!REAL!CHAR

                    ---Operator Definitions

BINARY("+",FADD)
BINARY("-".FSUB)
UNARY("-",FMINUS)
BINARY("*",FMUL)
BINARY("/",FDIV)
BINARY("\",DERIV)
UNARY("@",SHOW)
```

---The Derivative of F with respect to X

```
     $DERIV(F,X)
[1]  DERIV_(IF F==ATOM THEN (IF F=X THEN 1 ELSE Ø) ELSE
       IF F==UF   THEN  UF(OP(F),RO(F)\X)      ELSE
       L_LO(F);R_RO(F);O_OP(F);
          IF O=´+ THEN (L\X)+(R\X) ELSE
          IF O=´- THEN (L\X)-(R\X) ELSE
          IF O=´* THEN (L*R\X)+(R*L\X) ELSE
          IF O=´/ THEN ((R*L\X)-(L*R\X))/(R*R)   )
          $
```

---Functions that Print Formulas

```
     $SHOW(F)
[1]  FPRINT(F);""
     $


     $FPRINT(F)
[1]  IF F==ATOM THEN PRINT(F); GOTO %Ø
[2]  PRINT(´(); IF F==UF THEN GOTO %4
[3]  FPRINT(LO(F))
[4]  PRINT(OP(F));FPRINT(RO(F));PRINT(´))
```

---Functions that Construct Formulas

```
     $FADD(A,B)
[1]  FADD_BF(A,´+,B)
     $

     $FSUB(A,B)
[1]  FSUB_BF(A,´-,B)
     $

     $FMUL(A,B)
[1]  FMUL_BF(A,´*,B)
     $

     $FDIV(A,B)
```

```
                     [1]   FDIV_BF(A,´/,B)
                           $

                           $FMINUS(A)
                     [1]   FMINUS_UF(´-,A)
                           $
```

---Examples

```
         X_´X           ---Assign character ´x to be value of variable x
         F_(X*(X-3))    ---Assign formula (X*(X-3)) to be value of F
         @F             ---Print formula F using defined formula print
                        operator @
(X*(X-3))
         F              ---Print F without using defined print operator @
[LO:X,OP:*,RO:[LO:X,OP:-,RO:3]]
         @F\X           ---Print Derivative of F with respect to X
((X*(1-0))+((X-3)*1))
```

(FOR USE ONLY AT BBN CAMBRIDGE SITE)

PTIP User's Information

Getting the PTIP's attention Once you have connected your terminal to the PTIP, by turning it on and switching it on-line (or by dialing a dataset and pressing "DATA"), you must get the PTIP's attention and tell it your terminal's speed. This is done by typing the Hunt Character.

The Hunt character for the PTIP is "Control Q".

After you connect your terminal and type a control Q, you should receive the immediate response "BBN RCC n.m" where n and m are the PTIP software version number. If you do not receive this response, or if you receive a garbled response, press the "BREAK" key, wait a second, and type the control Q again.

Connecting to TENEX After receiving the PTIP version number, type a Control C. This will cause the PTIP to attempt a connection to a TENEX. The PTIP message "Trying" means that a connection attempt is being started. When the connection opens, you will receive a standard TENEX greeting from one of the BBN systems. Check the message to make sure you are on the right TENEX.

If you are not on the TENEX you want, use the the TENEX "MOVE" command to move your connection to the right system. Type "MOVE ?" to see the list of systems to which you can move. It is just the list of BBN TENEXes. Example: Q BBN RCC I.7__ C Trying__ BBN-TENEX 1.99.99, BBN-SYSTEM-B EXEC 1.98.98 @MOVE$ (TERMINAL TO HOST) C<return>
BBN-TENEX 1.97.97, BBN-SYSTEM-C EXEC 1.96.96 @

Note that the PTIP will not connect you to any site except a BBN TENEX site.

Disconnecting from TENEX There is nothing special to do to disconnect from the PTIP. After logging out of TENEX, and after a minute's delay to allow you to log back in, the connection from PTIP to TENEX will be closed. The PTIP message "Closed" will be printed, but there is no need to wait for it. After the "Closed" is printed, you are back to the beginning and should type "Control Q" to use the PTIP again.

Error messages and problems After a "MOVE" command, the PTIP will wait indefinitely for a response from the requested host. This removes the need to try connecting every few minutes after a host has crashed. If you want to give up and quit waiting, type the "BREAK" key. This will produce the message "Aborting", and a moment later the message "Connection failed, trying other hosts", followed by a connection to some other TENEX.

After a "MOVE" command, you might get a "TENEX RESTARTING, WAIT" message or a "TENEX NOT AVAILABLE" message from the TENEX system. You can break the connection in this situation, too, and in any situation before logging in, by typing "BREAK".

If you are connected to a TENEX, you may receive the message "Host reset, connection closed" if the TENEX crashes. This will usually not come immediately after the crash, but will be delayed until the restart begins or until the operator forces the PTIP to discard connections to the host.

If the PTIP crashes, you may receive the "BBN RCC n.m" message, or some garble, as if you had typed "Control Q". In this case, the PTIP has reinitialized and forgotten your connection. You should reconnect to the TENEX you were using and you will probably find your job there detached (unless, of course, the crash caused a complete TENEX restart).

Other features or their lack The PTIP has no command language, per se. The only characters of interest to the PTIP are Control Q, Control C and BREAK. Even those are of no interest to the PTIP once you have logged in to a TENEX. There is no need to double "at-sign"s as on the TIP. At the same time, there is no way to find out whether the PTIP is still there but the TENEX is not there when your terminal stops responding, as you could do on the TIP by typing "@X".

The PTIP does not do padding for terminals. Rather than using @D C E and the like on the TIP, you use TERMINAL (TYPE IS) on the TENEX to get your padding. The PTIP does not support 2741's or other non-ASCII terminals. It does support a wide range of baud rates, and can hunt to rates up through 24ØØ baud. This is why the new hunt character, control Q, was chosen rather than the "E" of the TIP or some other more familiar character.

Default hosts and rates When the installation of lines to the PTIP has been completed, we will set default baud rates and default hosts for users who request them. This will remove the need for typing Control Q to set a baud rate, and will cause the Control C to request a connection from the TENEX you normally use, if that TENEX is up.

RD

The RD subsystem is used to look at and manipulate the mail entries in your MESSAGE.TXT files (see also READMAIL and SNDMSG).

It is actually a version of TECO with a set of TECO macros preloaded. All commands are thus calls of these macros, or other normal TECO commands. Some knowledge of TECO is desirable if you plan to use RD.

When started it reads your MESSAGE.TXT file and, unless there are no entries, types a summary of the heading information (date, author, subject) for each entry. You may then go through them, listing and/or copying them to other files for later use.

The intention is that you will dispose of all entries in your MESSAGE.TXT file by reading and/or copying them to other files, so that you can use the MK command to clear your MESSAGE.TXT file at the end of each RD session.

Below is a typescript of a session with RD showing what is typed at startup, and the help text string (given by MH$) which describes all the commands. Underlined characters were typed by the user.

```
@RD

1941 CHARS
1  29-MAY-73 HGM at CCA-TENEX - -
2  29-MAY-73 ROBERTS   RD
3  29-MAY-73 HGM at CCA-TENEX   RSSER - ´UP´ TIMING
4  29-MAY-73 PLUMMER   RD
5  29-MAY-73 PLUMMER   NETLOAD
6  29-MAY-73 PLUMMER   RSYSTAT
TYPE MH$ FOR HELP
MH$
ALL COMMANDS ARE TECO MACROS, TERMINATE WITH ESCAPE KEY
TO ABORT PRINTOUTS HIT DELETE KEY TWICE
DISPLAY COMMANDS

 MN   TYPE NEXT MESSAGE
#MJ   JUMP TO MESSAGE NUMBERED #
#MT   TYPE MESSAGE NUMBERED #
 ML   LISTS ALL MESSAGES IN BUFFER
```

```
# FILE READ COMMANDS
#
#  M@    READS IN MESSAGE.TXT AND LISTS IT (AUTOMATIC ON START)
#  MY    ASKS FOR FILE NAME-READS IN AND LISTS
# #MR    READS IN SPECIAL MSG FILES #=Ø :MESSAGE.TXT
#                                  #=1 :ACTION.MSG
#                                  #=2 :FILE.MSG
#
# FILE WRITE COMMANDS
#
# #MW    WRITES MESSAGE NUMBERED # TO NEW FILE T.MSG;T
#          EACH USE CREATES A NEW FILE (:1,;2, ..)
#
#   TO APPEND MESSAGES TO SPECIAL FILES
#
# #MS    SAVE- APPEND CURRENT MSG (THE ONE JUST READ VIA MT OR MN)
#           TO SPECIAL FILE #=1 or 2.  (OPENS, APPENDS, AND CLOSES)
#
# #MV    OPEN NEW VERSION OF SPECIAL FILES.
#                  #=1 :ACTION.MSG
#                  #=2 :FILE.MSG
#
# #MO    OPEN OLD VERSION OF FILE - #=1 : ACTION.MSG
#                                  #=2 : FILE.MSG
#        (TO OPEN OTHER FILES FOR APPENDING, USE :
#          EAFILE.NAME$    FOLLOWED BY #MA'S.)
# #MA    APPEND MESSAGE NUMBERED # TO OPEN FILE
#  EF    CLOSE FILE
#
# (NOTE: ACTION.MSG & FILE.MSG MUST EXIST FOR MO TO WORK.
#   TO START THEM, USE THE #MV COMMAND.)
#
#
# EXIT COMMAND
#
#  MK    CHECKS TO SEE IF CURRENT BUFFER IS SAME LENGTH AS
#          MESSAGE.TXT AND IF SO CLEARS (KILLS) MESSAGE.TXT AND EXITS
#          TO EXEC.  IF NOT IT LEAVES NEW MESSAGE.TXT IN BUFFER AND LISTS.
#        WATCH OUT!  THIS COMMAND IS INTENDED TO BE
#        USED AFTER WRITING OUT USEFUL MSGS SINCE IT WIPES OUT
#        MESSAGE.TXT (BUT DOES NOT RESET SIZE)
#
# *;H$
#
# @
```

READMAIL

READMAIL is a subsystem which does exactly what is implied in the name. When a user logs in and finds he has a message he then calls for READMAIL. Below is a sample of the dialogue.

```
# @readmail
# READMAIL 2C(13)
# TYPE ? FOR HELP
# *?
# Type just a CR or EOL to get your MESSAGE.TXT printed from point of
# last reading on your terminal.  Use "D" to specify another date.  Use
# "F" to specify another input file.  Use "O" to  specify another output
# file.  Use "R" to specify reverse order (ie. oldest last).  While
# printing, typing rubout will skip to the next message in the file if
# the output file specification is not used.
#
#
# Commands to READMAIL are a single character.
#
# Reading   does   not   actually   begin   until   you   give   the
# carriage-return  command,  at  which point READMAIL prints on the
# output file in the specified order all messages  from  the  input
# file received after the date.
#
# Default values (i.e.  for those items you do  not  give  commands
# for) are:
#
#         Input file:  <connected-directory>MESSAGE.TXT;1
#         date:  read date of file
#         output file:  TTY:  (your terminal)
#         forward order (oldest first)
#
# Dates may be entered in almost  any  reasonable  form,  but  must
# include  month,  date,  and  year.   A time may also be specified
# after the date, separated from it by a space.
```

Normal TENEX editing characters apply when typing in file names, but no editing is possible while typing dates.

RELRIM

Converts a .REL file created by MACRO, FAIL, or FORTRAN into a RIM10B self-loading paper tape containing standard check-summed blocks.

When started, RELRIM asks:

```
     INPUT FILE:
```

to which the user responds appropriately. After the paper tape is punched, the program EXITS to the EXEC.

RENBR, the FORTRAN Renumbering Program

RENBR is a program written in hardware independent FORTRAN which sequentially statement numbers and/or forms cross-reference listings of FORTRAN programs read as data.

Instructions for Use

When started, RENBR will ask the user to supply the input, output, lister and scratch unit numbers and file names as well as other necessary information. If the only response to a request is a carriage return (blank line), a default answer is assumed. A negative response to any request for numeric information (except statement number increment) will cause the default values to be assumed for the present and all remaining requests the asking of which will then be suppressed. The default names of all files except the scratch file are defined by DATA statements in subroutine REUNIT. The default value of the lowest statement number is the absolute value of the statement number increment. All other default values are defined by executable statements prior to the call of subroutine REUNIT in the main program. Possible requests and their default values are as follow:

Request	Default
IS RENUMBERING DESIRED (Y OR N)	YES
IS OUTPUT TO BE IN CARD FORMAT (Y OR N)	YES
IS LISTING DESIRED (Y OR N)	YES
TITLE FOR LISTING	
INPUT UNIT NUMBER	1
INPUT FILE NAME	INPUT
LISTER UNIT NUMBER	20
LISTER FILE NAME	LIST
OUTPUT UNIT NUMBER	21
OUTPUT FILE NAME	OUTPU
SCRATCH UNIT NUMBER	22
SCRATCH FILE NAME	SPARE
INITIAL PAGE NUMBER	1
LOWEST STATEMENT NUMBER	1
STATEMENT NUMBER INCREMENT (NEGATIVE IS LEGAL)	1

The tab character will separate statement number fields from statement text in the output if the question concerning output format is answered with an "N".

Input File Limitations

The input file must have a name of 5 or fewer characters and the extension DAT, must be line blocked and must not contain internal

```
# control characters such as form feeds.  A final  form  feed  will
# cause no difficulty.
#
# The input file can contain tag characters if the FORTRAN compiler
# and  operating  system  allow  these.   If  the  compiler  and/or
# operating system do not allow tabs but the input program contains
# tabs, such tabs should first be converted by the text editor to 6
# or more blanks or else be converted to sufficient blanks to  fill
# to  the  normal  tab  stops if the left tab stop is in, or to the
# right of, column  7.  The  tab  character  is  used  in  FORTRAN
# programs  on  some  non-card  systems  to  separate the statement
# number field from the statement.
#
# RENBR detects the end of the input file by end-of-file  tests  in
# its  READ  statements.   If  the  end-of-file test feature is not
# available, the input file should be terminated by  an  extra  END
# statement which will not appear in the generated output.
#
# Restriction
# Continuation lines following  a  comment  line  are  taken  as  a
# continuation  of  the  comment  and  are  written into the output
# unchanged.  For  this  reason,  comment  lines  can  separate
# statements, but cannot appear within a single statement.
#
# Restriction
# A line with a non-blank non-tab non-digit character other than  C
# (which  would  indicate a comment) in column 1 followed by a tab,
# or by a number (formed of no more than 4 digits) and a tab, or by
# 4  characters  formed of any combination of blanks (also known as
# the space character) and/or digits  will  be  taken  as  a  legal
# FORTRAN  statement.  If the line begins a new statement, then the
# initial character will appear in the output at the start of  each
# line  of  the statement including all continuation lines (whether
# or not the character originally appeared at the  start  of  these
# continuation  lines).  Some compilers require B, D or I in column
# 1 to specify variable type.  Also, DEC PDP-10 FORTRAN allows D in
# column 1 to indicate a debugging line the compilation of which is
# optional.  In DEC FORTRAN, the use of the D at the start of lines
# which continue a debugging line is acceptable but not necessary.
#
# Restriction
# Blanks are trimmed from the right end of lines  prior  to  output
# regardless of syntax.  If alphameric strings are specified as the
# number of characters followed by the letter H and the  characters
# of  the  string,  then  lines  which  end  in  alphameric strings
# containing  terminal  blanks  will  have  these  blanks  removed.
# Therefore,  unless  the output is written onto cards, a statement
# such as
#      A=1H
#
# should instead be written as
#      A=(1H )
# or
#      A=´ ´
#
# Restriction
# A single statement can be continued on no  more  than  19  lines.
# Unless  the  dimensions  of  the  storage arrays are increased, a
# single main program or routine being renumbered  can  contain  at
# most  1000 numbered statements (or 25 less than this if a listing
# is also being made).  There is no restriction on  the  number  of
# statement  number  references  within  a  single  main program or
# routine.
#
# Restriction
# The END statement at the end of a program must appear on a single
# line  (although the letters of the word END can be preceded by or
# be separated by blanks or  tabs).   The  input  can  contain  any
# number  of programs or routines, each with its own END statement.
# If the end-of-file test in a read statement is not available,  an
# additional  END  statement, after the final program or routine in
# the input, can be used to force  a  normal  exit  which  includes
# printing of the table of contents.
#
# Restriction
# A line beginning a new statement must have one of  the  following
# formats.
#  A)    A line beginning with  a  non-tab  character  other  than  C
#        followed by 4 blanks and/or digits followed in column 6 by a
#        blank or by a zero.  It is possible  for  the  character  in
#        column 1 to be a blank or a digit of the statement number.
#
#  B)    A line beginning with a tab followed by the first  character
#        of the statement which cannot be a digit.
#
#  C)    A  line  beginning  with  a  non-tab  non-blank  non-digit
#        character  other  than  C  followed by a tab followed by the
#        first character of the statement.
#
#  D)    A line beginning with a digit or  digits  of  the  statement
#        number  followed by a tab followed by the first character of
#        the statement.
#
#  E)    A  line  beginning  with  a  non-tab  non-blank  non-digit
#        character  other  than  C followed by the digit or digits of
#        the statement number followed by a tab followed by the first
#        character of the statement.
#
# If (BLANK) represents a blank, (TAB) represents a tab and  (TEXT)
# represents  the  text  of  the  statement, then the following are
# typical lines which start new statements.  Of course, the text of
# the statement can itself begin with one or more blanks or tabs.
```

```
(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)(TEXT)
(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)Ø(TEXT)
D(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)(TEXT)
D(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)Ø(TEXT)
(BLANK)(BLANK)(BLANK)22(BLANK)(TEXT)
(BLANK)(BLANK)(BLANK)22Ø(TEXT)
D(BLANK)(BLANK)22(BLANK)(TEXT)
D(BLANK)(BLANK)22Ø(TEXT)
22(BLANK)(BLANK)(BLANK)(BLANK)(TEXT)
22(BLANK)(BLANK)(BLANK)Ø(TEXT)
D22(BLANK)(BLANK)(BLANK)(TEXT)
D22(BLANK)(BLANK)Ø(TEXT)
(TAB)(TEXT)
D(TAB)(TEXT)
22(TAB)(TEXT)
D22(TAB)(TEXT)
```

Restriction
A continuation line must have one of the following formats.

A) A line beginning with a non-tab non-digit character other than C followed by 4 blanks followed by a non-blank non-tab non-zero character which is ignored. The initial character can, of course, be a blank.

B) A line beginning with a tab (or with 5 or more blanks) followed by a non-zero digit which is ignored.

C) A line beginning with a non-tab non-blank non-digit character other than C followed by a tab (or by 4 or more blanks) followed by a non-zero digit which is ignored.

The following are typical continuation lines.

```
(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)2(TEXT)
(BLANK)(BLANK)(BLANK)(BLANK)(BLANK)A(TEXT)
D(BLANK)(BLANK)(BLANK)(BLANK)2(TEXT)
D(BLANK)(BLANK)(BLANK)(BLANK)A(TEXT)
(TAB)2(TEXT)
D(TAB)2(TEXT)
```

Description of the Listing Produced by RENBR

RENBR can, at the user´s request, produce a listing of the programs read as data. (IF renumbering has not been requested, RENBR assumes by default that a listing is to be made.) The listing of each separate main program or routine is begun on a new page. In the upper right corner of each page are printed both the current page number and the name of the program or routine. To the left of each non-comment statement is printed the count or sequence number of the statement within the current program or routine.

# Following the listing of the program or routine is a list of
# statement numbers and of the sequence numbers of the statements
# in which these statement numbers are referenced. If renumbering
# is not being performed, this list will also include as negative
# numbers the sequence numbers of the numbered statements
# themselves. These negative numbers are not included in the list
# of statement number references if renumbering is being performed,
# since then the resulting statement numbers will be in order so
# that it is assumed that further assistance is not needed to
# locate them.
#
# After the list of statement number references is a list of all
# key words, names and constants used in the program or routine and
# of the sequence numbers of the statements in which these are
# used. A sequence number is given as negative if the referenced
# variable or array is defined in the statement by the equals sign
# operator. Key words such as GO TO are considered to be single
# words in the index, but blanks and tabs are otherwise used as
# word separators. Except for the initial key words, items
# appearing either in a DATA statement or in a FORMAT statement are
# not included in the list.
#
# Changing Length of Uninterrupted Listing
#
# The tables of statement number references and indexes are printed
# at the end of each program or routine, or when the necessary
# storage fills. As supplied, RENBR will produce uninterrupted
# listings of FORTRAN routines of approximately 800 non-comment
# lines (dependent on programmer style). This requires 2 arrays of
# 1000 locations each to store the statement references, and 1
# array of 7000 locations to store the symbol dictionary. If the
# storage necessary to run RENBR must be reduced, it is suggested
# that all appearances of the number 1000 be changed to 500 and
# that all appearances of 7000 be changed to 3000. This will cause
# the tables and indexes to be dumped after processing of about 250
# non-comment lines, but the listings will still be correct.
#
# Maintaining Logical Blocks of Statement Numbers
#
# Some programmers select statement numbers used within a logical
# division of a program from a different range than those used
# elsewhere within the same program. RENBR can maintain these
# regions when renumbering. However, since a single normal
# renumbering would destroy such regions, a command card within the
# program is used to specify the step size rather than querying the
# user for this information. This command card is a comment card
# with C in column 1 followed by the word RENBR and those switches
# for which values are being specified. A typical command card
# would be
#
CRENBR I-10 B 10 O 200 N 400

```
# The switches are
#
# B = the base number.  The number following this switch will be
#     the smallest generated statement number which will appear
#     within the renumbered program or routine.  Normally this
#     would be the first statement number in the renumbered
#     program, but if the increment between generated statement
#     numbers is specified to be negative, then the base number
#     will be the number of the final numbered statement within the
#     program.  If the B switch does not appear on the CRENBR card,
#     then the value used as the base number will be that specified
#     by the user when RENBR was started, or will be the absolute
#     value of the statement number increment if the user did not
#     specify a base number when RENBR was started.
#
# I = the statement number increment.  The number following this
#     switch will be the increment between generated statement
#     numbers.  If the I switch does not appear on the CRENBR card,
#     then the value used as the increment will be that specified
#     by the user when RENBR was started, or will be 1 if the user
#     did not specify a statement number increment when RENBR was
#     started.
#
# N = the new region size.  The number following this switch
#     specifies the jump in generated statement numbers from the
#     start of one region to the next in the renumbered program.
#     If the N switch does not appear on the CRENBR card, then the
#     old region size as specified by the O switch is also used as
#     the new region size.
#
# O = the old region size.  The number following this switch
#     specifies the jump in the original statement numbers from the
#     start of one region to the next in the program which is being
#     renumbered.  If the values specified by the N and O switches
#     differ, then the value of the O switch should probably be
#     converted after renumbering to that which the N switch had
#     before renumBering.  If the O switch does not appear on the
#     CRENBR card, then no regions will be preserved.
#
# The switches can appear in any order on the CRENBR card.  Only
# the I switch will accept a negative value.  Blanks or tabs can
# appear anywhere after the initial C of the CRENBR card, but are
# not necessary.
#
# The values specified by the CRENBR card apply to all the
# statement numbers within a single program, but the CRENBR card
# can appear anywhere in the program before the terminal END
# statement.  If multiple CRENBR cards appear within a single
# program or routine, the values used are the final values
# specified for each switch.  The CRENBR card applies only to the
# program or routine in which it appears.  The default values of
# the switches (set by the user when RENBR was started) are
# restored for the next program or routine read.  In the case of
# the N and O switches, these defaults are to not preserve regions.
#
# The sample CRENBR card show above would specify that original
# regions 1-199, 200-399, 400-599 etc.  would be translated to
# regions such as 1-399, 400-799 and 800-1199.  The actual
# translation would depend on the relative placement of the
# original regions.  For example, using the above sample CRENBR
# specification, the statement number sequence
#
#      270  350  260  351   15    2    6  150   99 1600 1622
#
# would be translated to the following sequence
#
#      830  820  810  800  440  430  420  410  400   20   10
#
# To allow addition of new statements to a region, statement
# numbers which would normally be outside the region are taken as
# part of the surrounding region if statement numbers both before
# and after the addition are within the region.  Therefore, the
# statement number sequence
#
#      270 8350 4260 351   15    2    6  150   99 1600 1622
#
# would be renumbered to the same sequence as given before.
#
#                Statement Types Recognized by RENBR
#
# RENBR will renumber statement numbers contained in the following
# types of FORTRAN-II and FORTRAN-IV statements.  These key words
# can also be used for variable names or for array names without
# causing any difficulties.  The ability to handle statement
# numbers in other types of statements can be easily added.
#
#      ACCEPT                   ACCEPT TAPE
#      ASSIGN
#      CALL
#      DECODE
#      DO
#      ENCODE
#      FREQUENCY
#      GO TO
#      IF                       IF ACCUMULATOR OVERFLOW
#      IF DIVIDE CHECK          IF QUOTIENT OVERFLOW
#      PRINT
#      PUNCH                    PUNCH TAPE
#      READ                     READ INPUT TAPE
#      REREAD
#      TYPE
#      WRITE                    WRITE OUTPUT TAPE
```

In addition to the above, the FIND statement is recognized so that the unit and record numbers can be separated before being entered into the index. To prevent filling and index with often useless information, the DATA and FORMAT statements are also recognized so that items following these keywords are not indexed.

RENBR handles as single units alphameric strings which are designated either as a decimal number of characters followed by the letter H and the characters of the string, or else are preceded and followed by apostrophes. It is likely that no harm will result if programs containing other alphameric string designations are renumbered. If some character other than apostrophe is used as the delimiting character in programs to be renumbered, the only situation likely to give trouble will be when IF statements are used to test character data against alphameric strings containing either left or right parentheses. RENBR was used for several years before the ability to handle alphameric strings as single units was even added to it.

Statement numbers appearing in CALL statements can be preceded either by an asterisk, by a dollar sign or by an ampersand. The RENBR program must be changed if some other character is used to indicate such statement numbers in CALL statements.

Format of Statements in the Output File

The text of statements in the renumbered output file begins in column 7 if card format has been selected, or else following the tab which separates the statement number field from the statement text field. Original indentation in the form of additional blanks or tabs is not preserved. Where possible, in a continuation line the original number of blanks separating the statement continuation character from the text of the continued statement is maintained. All other spacings are left unchanged.

Files Provided

RENBR.F4	Source of the FORTRAN renumbering program.
RENBR.DAT	Test data for RENBR. Directions for use are given as comments at start of file.
RENBR.LST	This file.
RENBR.RNO	File which generates this file when processed by the PDP-10 text justifier program RUNOFF.
REFMT.F4	FORTRAN program used with RENBR BLOCK DATA routine and with the DATAST package to reorder the RENBR syntax recognition table. Directions for use are

given as comments at start of file.
#
DATAST.F4 FORTRAN subroutine package which generates as
output the FORTRAN DIMENSION, EQUIVALENCE and DATA
statements which represent an input single
precision singly subscripted integer array,
calling sequence is described in subroutine DATA.

REPORT

The REPORT subsystem is used to interrogate a data base which contains a running commentary on the status of the BBN-TENEX system. The data base includes system Interruption Reports, System Restart Reports and Scheduled Downtime Reports. The program is self-explanatory. It asks the following questions:

```
ENTER DATE FROM WHICH TO LIST STATUS:
```

> (The default reply is carriage return which implies earliest date of entry in data base; otherwise, enter date in form MM/DD/YY, e.g. Ø3/12/74)

```
OUTPUT TO:
```

> (The default reply is carriage return which implies TTY:; otherwise, enter a file specification or LPT:)

```
REVERSE OR FORWARD ORDER (R OR F):
```

> (the default reply is carriage return which implies R, i.e., reverse chronological order for output; otherwise, enter R or F)

RUNFIL

Provides a means for using a file instead of a terminal to supply an input command stream. A file is prepared containing the character stream which would be typed on the terminal to perform some desired set of operations. The stream is initially received by the EXEC but may start any subsystem, and primary input will continue to come from the file. All output is directed to the controlling terminal in the usual manner, and the resultant typescript is indistinguishable from that produced when input is from the keyboard. Optionally, output may be directed to a file other than the terminal. However, such a file may contain "errors" in echoing, that is, the source file "echoes" present in the output file may be out of phase with the output.

```
@RUNFIL

COMMANDS FROM FILE:     name
```

The user supplies the name of the file from which commands are to be taken. If the name is confirmed with carriage return, output will be directed to the terminal and processing will begin. If the name is confirmed by comma, the program will ask:

```
OUTPUT TO FILE:     name
```

and the name of the file to receive primary output should be entered.

The program begins by starting an EXEC in an inferior fork, with primary input and output set as determined by the dialogue above. The command file must contain EXEC commands to start any desired subsystem. The last command should leave control at the command level of the EXEC. When the end of file is reached in the command file and any processing is completed, the inferior fork(s) will be killed and control returned to the top-level EXEC.

Format of Command File

As stated, the command file contains exactly what would be typed on a controlling terminal to perform some desired set of operations. To facilitate certain operations, however, the following actions cause special interpretations by RUNFIL.

Control-^ acts as a warning character and interprets the immediately following character(s) as follows:

1) A-Z, [,\,],^,_,@

Converts the single character in this group to its control equivalent. Eg. ^C (i.e. control -^ followed by C) becomes ^C, etc.

Note: Control-C in an input stream causes an immediate termination of processing as usual. However, for some subsystems, ^C is the only way to return to the EXEC, and the user would normally not type ^C for this purpose until processing had been completed. In a command file, ^^B serves this function, i.e. RUNFIL waits until the inferior program is ready for more input (an indication that processing is completed), and then sends a ^C.

2) Decimal number

The number will be taken as a length of time (in milliseconds) to wait before taking further input from the command file.

The number must be terminated by a non-numeric character
(e.g., space), which will have no other effect in the input
stream. This "pause" facility is furnished to allow orderly
input from the terminal at desired points in the file
stream.
#
# Note that once processing has begun, any characters typed on the
# controlling terminal will be intermixed with the file stream,
# usually producing undesired results. In particular, a ^C typed
# on the controlling terminal will be handled by the <u>lower-level</u>
# EXEC, and so is not a way to abort command file processing. The
character ^B is enabled in RUNFIL for this purpose. (^B is the
# only character which has a different effect when input from the
# terminal.)
#
In most cases ^B from the terminal will cause an immediate
termination of command-file processing and a return to the
top-level EXEC. However, ^B typed during a requested "pause" in
command-file input (^^<decimal number> above) will break the
pause, i.e., RUNFIL will "wake up" and immediately resume command
file input. Thus processing can be caused to continue after
manual terminal input without waiting out the remaining specified
delay. An actual ^B (as opposed to ^^ followed by B in the file
stream is treated as ordinary input and is passed to the
subsystem running under RUNFIL.

RUNOFF

RUNOFF is a TENEX program to facilitate the preparation of typed or printed manuscripts, such as memos, manuals, etc.

The user prepares his material on any regular TENEX terminal, and writes it onto a file using TECO. The user includes not only textual material, but also case and formatting information. RUNOFF then takes the file and reproduces it onto the line printer, teletype, any terminal or other file to produce a final copy or final file image. It performs the formatting and case shifting as directed, and will also perform line justification, page numbering and titling, etc., as desired.

The principal benefit of such a program is that files prepared for use with it may be edited and corrected easily. Small or large amounts of material may be added or deleted, and unchanged material need not be retyped. After a set of changes, the program may be operated to produce a new copy which is properly paged and formatted. Documentation may thus be updated as necessary without requiring extensive retyping.

TENEX Operating Procedure

RUNOFF is a regular TENEX subsystem, and is started by the EXEC command

```
        @RUNOFF_
                  (In this document, _ will be used to mean
                  carriage return.)
```

The program will print its name and version and ask for the input file name:

```
   (1)    INPUT FILE:
```

The user should type the name of the source file (using alt-mode for name recognition) and confirm the name with a carriage return or comma. If a carriage return is typed, RUNOFF will assume standard settings for all modes and immediately begin to produce output to the line printer. Typing a comma allows the user to specify various additional options as follows.

```
   (1)    INPUT FILE: name,
                  (Comma to ask for options)

   (2)    ASCII, DURA OR FLEXO?
                  (Type A, D, or F.  If D or F typed, output
                  will be to paper tape punch with no further
                  questions asked.)

   (3)    OUTPUT FILE:
                  (Confirm (c) with carriage return for no more
                  questions, comma to specify following
                  options.)

   (4)    UNDERSCORE (L, C, B OR N)
                  (Underscore by line, character, backspacing
                  or none.  Use L (normal) for line printer, C
                  for flexo or dura, N for anything which does
                  not have underscoring capability.)

   (5)    SIMULATE FORM FEED (Y OR N)?
                  (Yes means use repeated line feeds to eject
                  to top of each new page.  No means use real
                  form feed character.  Use N (normal) for line
                  printer, flexo and dura.)
```

(6) PAUSE?

(Answer Y or N. Yes means stop program at completion of each page so that paper can be adjusted in on-line output device. Used only for direct output to TTY. Use N (normal) for line printer, flexo or dura.)

When finished with the current input file, RUNOFF will type DONE and await specification of a new input file.

Summary of Operations

1. For output to line printer:

   Type carriage return after input file name.

2. For Dura typewriter paper tape:

   Type comma after input file name,

   Type D to question 2.

3. For Flexowriter paper tape:

   Type comma after input file name,

   Type F to question 2.

SOURCE FILE FORMAT

As stated above, the source file contains the textual material which will appear on the final copy, plus information to specify formatting. Most importantly, upper and lower case information may also be supplied so that copy can be prepared on the teletype or other such device which can normally input only upper case letters. All command information consists of regular ASCII printing characters so that a listing of the source file may be examined if the final copy is not exactly as desired.

All material in the source file is taken to be source text except those lines beginning with a period. A line beginning with a period is assumed to be a command, and must match one of those listed below. The commands provide the formatting information, and control various optional modes of operation.

Usually the text is filled and justified as it is processed. That is, the program FILLS a line by adding successive words from the source text until one more word would cause the right margin to be exceeded. The line is then JUSTIFIED by making the word spacings larger until the last word in the line exactly meets the right margin.

The user may occasionally wish to reproduce the source text exactly which is done by disabling filling and justification. The program may be set to fill but not justify, in which case the output will be normal except that lines will not be justified to the right margin. The program may also be set to justify but not fill, although this would probably produce peculiar results and is not recommended.

When the fill mode is on, spaces and carriage returns occurring in the source text are treated only as word separators. Multiple separators are ignored.

Some of the commands cause a BREAK in the output. A break means that the current line is output without justification, and the next word goes at the beginning of the next line. This occurs at the end of paragraphs.

The program will advance to new pages as necessary, placing the title (if given) and the page number at the top of each page. The user may explicitly call for a page advance where desired, and may inhibit the occurrence of a page advance within specified material.

CASE INFORMATION

Specification of case for files prepared on the teletype is done with two characters, up-arrow (^, shift-N), and back-slash (\, shift-L). The appearance of an up-arrow causes the letter immediately following to be transmitted in upper case. The appearance of a back-slash causes the letter immediately following to be converted to lower case. Any letter not preceded by one of these characters is transmitted in the current mode. The mode is initially lower case, and is changed by the occurrence of two successive case control characters. Two up-arrows (^^) cause the mode to be set to upper case, and two back-slashes (\\) cause the mode to be set to lower case.

The use of the above corresponds to the use of the shift and shift-lock keys on a typewriter. Usually, typing appears in lower case. To type one letter in upper case, the shift key is used. The shift-lock is set to type a series of upper case letters, after which it is released.

The following shows the uses of the case control characters:

```
^HERE IS A ^SAMPLE ^SENTENCE IN ^^UPPER CASE\\ AND LOWER CASE.
```

becomes:

```
Here is a Sample Sentence in UPPER CASE and lower case.
```

Note: Case conversion takes place only on ASCII codes 101 to 132 octal, that is, the upper case letters. Any actual lower case letters (codes 141 to 172 octal) appearing in the source will be transmitted unchanged. If the source is prepared on a device such as a flexowriter or model 37 teletype or any terminal which produces letters of the proper case, the mode should be set to upper case at the beginning of the file and left unchanged for the remainder.

Special Characters

The character ampersand (&, shift-6) is used to specify underscoring. The ampersand will cause the character following it to be underscored, e.g. &f&o&o becomes foo.

Underlining of a string of characters can also be specified, similarly to the use of the shift lock operations described above. An appearance of ampersand preceded by up-arrow (^&) will cause underlining of all following characters except space. An appearance of ampersand preceded by backslash (\&) will disable this mode.

It is occasionally necessary to include spaces in the text which should not be treated as word separators. For this purpose, RUNOFF treats numbersign (#) as a quoted space; i.e. it will print as exactly one space in the output, will never be expanded nor changed to a carriage return.

To allow the appearance of the special characters (ampersand, number-sign, up-arrow, or back-slash) in the output, the character left-arrow (_, shift-O) is used as a quote character. The character immediately following a left-arrow will be transmitted to the output with no formatting effect. The left-arrow itself is just another case requiring the usual quote characteristic. The following five cases occur: _&, _^, _\, __, and _#.

COMMANDS

The following commands will be recognized if they are at the beginning of a line started with a period. Any line in the source file beginning with a period is assumed to be one of these commands. If it is not, an error diagnostic will be typed and the line will be ignored. Some commands take one or more decimal numbers following. These are separated from the command by a space.

Formatting

.BREAK

Causes a break, i.e. the current line will be output with no justification, and the next word of the source text will be placed at the beginning of the next line.

.SKIP n

Causes a break after which n*(line spacing) lines are left blank. If the skip would leave room for less than two printed lines on the page (i.e. if there are less than n+2*(line spacing) lines left), the output is advanced to the top of the next page.

.BLANK n

Exactly like SKIP, except that n (rather than n*(line spacing)) lines are specified. BLANK is used where space is to be left independent of the line spacing; SKIP, where the space should be relative to the size of line space.

.FIGURE n

Like BLANK except that if less than n lines remain on the current page, the page will be advanced, and n blank lines will be left at the top of the new page. Principally used where it is desired to leave room for a figure to be drawn in manually.

.INDENT n

Causes a break and sets the next line to begin n spaces to the right of the left margin. n may be negative to cause the line to begin to the left of the left margin (useful for numbered paragraphs).

.PARAGRAPH n

The number is optional and, if present, sets the number of spaces which paragraphs are to be indented. The initial setting is 5. The command causes a break and leaves (m+1)/2 blank lines, where m is the regular line spacing. The next line will be indented as indicated above.

.PAGE

Causes a break and an advance to a new page. Does nothing if the current page is empty. Titling and numbering as for automatic page advance.

.TEST PAGE n

Causes a break followed by a conditional page advance. If there are n or more lines remaining on the current page, no advance is made and no lines are skipped. Otherwise, the page is advanced as for PAGE. This command should be used to ensure that the following N lines are all output on the same page.

.NUMBER n

Turns on page numbering (normal) and, if _n_ is supplied, sets the current page number to _n_.

.NONUMBER

Turns off page numbering. Pages will continue to be counted, so the normal page number will appear if numbering is re-enabled.

Mode Setting

.JUSTIFY

Causes a break and sets subsequent output lines to be justified. (Initial setting)

.NOJUSTIFY

Causes a break and prevents justification of subsequent output lines.

.FILL

Causes a break and specifies that subsequent output lines be filled. Sets the justification mode to be that specified by the last appearance of JUSTIFY or NOJUSTIFY. (Initial setting)

.NOFILL

Causes a break and prevents filling of subsequent output lines. Also turns off justification.

Note:

1. The nofill-nojustify mode need be used only where there are several lines of material to be copied exactly. A single line example will not require using these commands if there are breaks before and after.

2. Normally FILL and NOFILL are used to turn both filling and justification on and off. It is usually desirable to do both. A subsequent appearance of a justification command will override the fill command however.

3. Because of the action of FILL, a single occurrence of NOJUSTIFY will cause the remainder of the file to be unjustified, with filling as specified. In order to justify but not fill (not recommended), a JUSTIFY command must follow every NOFILL command.

Parameter Settings

.LEFT MARGIN n

Causes a break after which the left margin is set to n. n must be less than the right margin, but not less than 0. The initial setting is 0. The amount of any indent plus the left margin must not be less than 0.

.RIGHT MARGIN n

Causes a break after which the right margin is set to n. n must be greater than the left margin. The initial setting is 60.

The number of characters on a line will be equal to or less than the right margin minus the left margin minus any indenting which may be specified. Even if filling has been disabled, lines will not be extended past the right margin.

.SPACING n

Causes a break after which the line spacing will be set to n. n must be within the range 1 to 5. Single spacing is 1, double spacing is 2, etc. (Initial setting is .SPACING 2)

.PAGE SIZE n

Sets the number of lines per page to n. n must be greater than 10. The initial setting is 58. n includes the top margin of 5 lines. The page number and title appear on the third line.

.TAB STOPS n ... n

Clears all previous tab stops and sets new tab stops as specified. The several n (max 32) must be greater than zero and in increasing order. They are the positions of tab stops independent of the setting of the left margin, although any which are less than the left margin will not be seen. There are no tab stops initially.

Tabs should be used only in lines which will be unjustified and unfilled, i.e. where filling is disabled or a break immediately follows. Clearly, the spaces specified by a tab character should not be expanded to justify the line--this would defeat the effect of tab formatting. The appearance of a tab in the source text will be translated to one or more spaces, the amount necessary to advance to the next tab stop. If a tab appears at a point where no further tab stops have been set on a line, the tab will be treated as though it had been a space.

Miscellaneous

.TITLE tttt ... tttt

This command takes the remaining text on the line as the title. This text will appear at the top of all subsequent pages, at position Ø, on the third line with the page number. The title is initially blank.

.SUBTITLE tttt ... tttt

This command takes the remaining text on the line as the subtitle. This text will appear on the line immediately following the title and page number. The subtitle is initially blank. The subtitle is not indented, but may contain leading spaces to achieve the same effect, if desired.

.CENTER

This command causes a break after which it centers the next line following in the source file. The centering is over position 3Ø, independent of the setting of the left and right margins.

.FOOTNOTE n

Allocates n*(line spacing) lines at the bottom of the current page for a footnote(1). If insufficient room remains on the current page, space will be allocated at the bottom of the following page. The text for the footnote should begin on the line following the command, and it may contain any appropriate commands (e.g. CENTER, SKIP) necessary to format the footnote. The footnote is terminated by a line beginning with an exclamation point (the remainder of which is ignored). The lines delimited by this line and the FOOTNOTE command are put into a buffer to be processed when the output moves to within the stated distance of the bottom of the page. If a page has multiple footnotes, the allocated space is the sum of the allocations for all footnotes assigned to the page. The user must

- - - - - - - - - - -

(1) This is a footnote. This text and the dividing line above were specified by text and commands following a FOOTNOTE 5 command.

include his choice of footnote-designating symbols within the text.

The current values of left and right margin and line spacing are saved and restored after processing of footnotes. Therefore, a footnote may contain commands which change these parameters, and the effect will be limited to the footnote text.

The actual space taken by the footnote may be more or less than that specified by n. The n merely allocates space and should be the user's best guess. If it is considerably off, the footnote lines may overflow the page, or extra space may be left at the bottom. The user may wish to adjust n after examining a first draft printout.

.INDEX tttt ... tttt

This command takes the remaining text on the line as a key word or words and adds it, along with the current page number, to the internal index buffer. The command does not cause a break. It should appear immediately before the item to be indexed. A key word or words may be indexed more than once.

.PRINT INDEX

Causes a break after which it prints the entire contents of the index buffer. Entries are printed in alphabetical order, and are set against the left margin. Regular line spacing is used, except that a blank line is left between entries of different first letters. The number of the first page on which each entry appeared is put on the same line as the entry, beginning at the middle of the line (midway between the left and right margins). Additional page numbers for multiple entries follow, separated by commas. The index buffer is left empty.

INDEX
(Entries entirely in upper case are command names.)

SNDMSG

SNDMSG is used to send messages to users throughout the ARPANET. It prompts for the addresses to which to send the message, then for the subject and text of the message. It automatically fills in the date and who the message is from. Finally it sends the message. If the message can´t be delivered immediately, it is queued and automatically sent later by the background MAILER process.

The following sections describe the various stages of SNDMSG (telling how to enter each kind of information) and the form of the message that is actually sent.

TO SUPPRESS TYPEOUT FROM SNDMSG, TYPE ^O (CONTROL-O) WHILE IT IS TYPING.

A. The Addresses

Addresses are divided into "To" and "cc". SNDMSG prompts separately for each. When SNDMSG prompts for addresses, enter them separated by commas. Carriage return terminates the list and goes on to the next section unless it is preceded by a comma, in which case address input continues on the next line. There must be at least one "to" address, but there needn´t be any "cc". If you don´t enter any "To" addresses, SNDMSG will skip to the subject and message, then will come back to the addresses.

The following kinds of items may appear in the address list separated by commas. (angle brackets are not typed - they are just used in this description to enclose a type of item)

1) <user name> - The mail is addressed to that user at the default host.
   Normally the default host is the local host (i.e. the one on which you are running SNDMSG) but you can change it as desired (see below).

2) @<Host name or octal host number>
   Changes the default host to be the given host. The default remains in effect for the rest of the address list until explicitly changed. To set the default back to local, type just "@" (i.e. omit the host name)

3) <user name>@<host name or octal host number>
   The mail is addressed to the given user at the given host. Putting the host name together with the user name makes it apply only to that user. That is, it does not change the default host, but overrides it just for the user. A blank host name means the local host.

4) <Group name>: - All subsequent addresses will be considered part of the specified group (which can be any name you care to define). This group remains in effect until explicitly changed by another "<group name>:" item. Typing just ":" with no name in front sets the group to none - i.e. subsequent addresses are ungrouped. Group names do not affect the sending of the message (they are not addresses), just what appears in the heading of the transmitted message (see section E).

   (Note: It is not necessary to separate the group name from the next address by a comma - the colon implies a comma.)

5) *<File name> - The message will be sent to the given file. Files are always local and must already exist. Defaults for fields you omit are:

   directory: logged in directory (not connected)
   name: SAVED
   extension: MESSAGES
   version: highest existing

   File name addresses are omitted from the message heading (see section E).

Special characters for editing, etc. are as follows:

1) recognition

   altmode (ESC) - If typed part way through a user or host name, causes recognition of the name if possible. The remainder of the name is typed out. If the name typed so far is ambiguous, a bell is rung.

2) correcting mistakes

   There is no way to edit anything but the current item being typed (i.e. editing won´t go past the preceding comma). The following control characters edit the current item or kill the whole address list.

   ^A - deletes preceding character - will only go back as far as preceding "@", ",", ":", or "*".

   ^H - same as ^A

   ^Q - deletes current word

   ^W - same as ^Q

   ^X - deletes whole address list, starts over rubout - deletes current item (back to "," or ":")

3) Viewing addresses

^R - retypes current word

^S - retypes whole address list

4) Inserting a file

A file containing addresses may be prepared ahead of time and inserted into the SNDMSG address list. Addresses are entered on the file exactly as they would be typed to SNDMSG except that carriage returns from a file will not terminate the address list - thus addresses on a file may appear on separate lines without commas if desired.

To insert the file, type ^B (control-B) during SNDMSG address input. You will be asked for a file name (which you type in the usual TENEX way) and for confirmation (type space or carriage return to confirm). The file is read into the address list exactly as if you had typed its contents. When SNDMSG types EOF (End-of-file) you continue entering addresses as usual.

Note: Although files are often used to hold definitions of groups (i.e. they often start with a "<group name>:"), files and groups are two distinct features. The purpose of a file is to permanently hold some list of addresses. The purpose of a group name is to prevent the component addresses from cluttering up the message heading. Although groups may be defined in files, they may also be typed directly into SNDMSG, and files inserted into SNDMSG need not define groups.

B. The Subject

When SNDMSG prompts for "Subject", type in up to one line, terminated by carriage return. You may type just carriage return to omit having a subject. If a subject is entered, it appears in the heading of the message. The same control characters described below under "Message" (section C) apply to the subject as well, with the following additions and exceptions.

^Z has no special meaning
^B automatically means insert file, never invoke TECO
^X deletes the whole subject and starts over
rubout is same as ^X but echoes differently.

C. The message

When SNDMSG prompts for "Message" type in the text of the message. When you are done composing the message, type ^Z (control-Z) to finalize it and to go on to the next stage.

Various control characters invoke features that aid in composing the message. They are:

1) Correcting typing mistakes

   ^A - deletes previous character
   ^H - same as ^A
   ^Q - deletes current line (if typed at beginning of line, deletes previous line)
   ^W - deletes previous word

2) Viewing the message

   ^R - retypes current line
   ^S - retypes whole message

3) Inserting pre-composed text and editing

   Especially when dealing with large messages, you may wish to prepare your message ahead of time on a file or you may find the local editing characters above insufficient. If you type ^B (control-B) SNDMSG asks whether you want to insert a file or invoke TECO (a general text editor).

   If you answer "T" TECO is started up with your message in its buffer. When you exit TECO (with ";H") the edited message is passed back to SNDMSG and you continue composing the message as usual (you are positioned at the end of the message, as if you had just typed it into SNDMSG).

   If you answer "F" SNDMSG asks for the name of the file to insert. Type the file name in the usual TENEX way (the usual editing characters and recognition apply). You will then be asked to confirm it. If you do (for example, you type space or carriage-return) the file is read in and appended to any text you have already entered, and "EOF" is typed. You then continue message composition as usual. NB: Characters read from the file are treated just like teletype input, so be sure the file doesn´t contain any of the control characters which have special meanings in SNDMSG!

MESSAGE.COPY

When you terminate input (^Z) or invoke TECO, the message is saved on the scratch file MESSAGE.COPY in the connected directory. Thus if you mess up your text in TECO or simply decide not to send the message yet you can quit SNDMSG (^C) and run it later, inserting the copy with ^B. Note that since MESSAGE.COPY is a scratch file, it is deleted when you log out!

CAVEAT - The message must not contain a line consisting of the single character "." if it is to be sent over the network.

D. Sending the message

When all information has been entered (addresses, message, etc.) SNDMSG asks whether to send the mail now or queue it for later delivery (the question is: "Q,S,?,carriage-return"). Type "?" for an explanation of the commands. If you just answer with carriage return, the message is sent immediately. The queueing or sending begins after this command is given -- i.e., when you terminate it with carriage return. (Note: If you have addressed the message to a file using * in the address list, SNDMSG will try to deliver it even if queueing is in effect.) The message is queued for or delivered to each of the addresses. For local addresses (ones in which no host was specified) duplicates are eliminated.

SNDMSG types each address as it starts to process it, then the outcome. There are three possible outcomes:

1. "ok" - the message has been successfully delivered to the address.

2. "queued" - the message was not delivered, but has been queued for later delivery. If SNDMSG was supposed to deliver the message (because of the commands you gave) the reason it had to be queued is typed. The message is placed on the file [--UNSENT-MAIL--].address in the directory under which you are logged in. MAILER, running in the background, will deliver it a soon as possible (see writeup of MAILER). In some cases (SNDMSG will tell you if this is the case) you may have to run MAILER yourself.

3. "can´t" - the message could not be delivered, and SNDMSG did not queue it because the reason for the failure looked permanent. The reason is typed out, and the message is placed on the file /UNDELIVERABLE-MAIL/.address in the directory under which you are logged in. Thus, for example, if the problem was that you misspelled the person´s name,

you can correct it and requeue the message without having to
compose it all over again. (See the writeup of MAILSTAT in
this manual.)
#
# For an explanation of what happens to queued mail see the writeup
# of MAILER.
#
For information on checking the status of queued mail see the
writeup of MAILSTAT.
#
E. What the message looks like
#
The message has the following form:
#
Date: <date and time the message was composed>
From: <name under which you were logged in when you sent it>
Subject: <whatever you typed - omitted if you didn´t give a subject>
To: <all the addresses you send the message "To" except for file
(as opposed to user) addresses - addresses entered as part of a
group are replaced by the group name>
cc: <same as "To" but for the "cc" addresses - omitted if none>
#
<the message>

SNOBOL

The TENEX SNOBOL is identical to the one distributed by DEC with the following exception:

> Vertical Tab and Form Feed characters in a user´s data will be passed to his program like any other character, instead of faking an end-of-line.

SNOBOL users may obtain the following excellent references:

1) Wade, L.P., "SNOBOL4 Version 3.4", Digital Equipment Corporation, DECUS no. 1Ø-1Ø4, Nov. 24, 197Ø.
2) Griswold, Poage, Polonsky, "The SNOBOL4 Programming Language", Prentice - Hall, 1968.

SORT

SORT is a column-oriented text file sorter. It is described in Appendix E of the DEC COBOL manual, and also under the COBOL section of this User Guide.

Example:

To sort a file called SORT.IN which contains the sorting information in columns 5 through 10 of each line, and where each line is no more than 72 characters, type the following:

```
@SORT
*SORT.OUT/A/R72_SORT.IN/K5.6
SORTED 423 RECORDS
comment %
```

SON OF STOPGAP

N O T E   T O   U S E R S

This manual has been updated to reflect changes made at Utah which include Tenex implementation, removal of bugs, and new commands. It has been updated through program revision V1.2 which was the SOS version in existence at Utah from October 1972 up to October 1974. Since then, SOS has been using a Help-command and a helpfile called SOS.SUPPLEMENT to document revisions and enhancements. The reader of this file SOS.MANUAL should be aware therefore that certain aspects of SOS have changed or been extended, and that the SUPPLEMENT or, equivalently, the Help command is the up-to-date reference guide... but only a guide or quick summary.

In particular, for new users, the FILE= syntax herein has been changed to ask for just an input file or a <cr> deferral. All other changes are essentially "upwardly compatible" with this manual, but the user should browse through the SUPPLEMENT (via the Help command) to check recent revisions and new command-options.

ABSTRACT

SOS is a line-number oriented editor for text files. It features two flavors of intraline editing (for Teletypes and displays), string search and substitution, hyphenless text justification, and other glories.

CREDITS

The original STOPGAP text editor was designed and programmed by Bill Weiher. Steve Savitzky added text justification (JU, JC, JR, and JL commands) and more extensive string search features. Dan Swinehart subsequently added display line editor commands (Z and Q) and automatic file saving (SAVE and ISAVE). This manual has been rewritten by Les Earnest, and at Utah by Kevin Kay.

1. INTRODUCTION

SOS provides the ability to insert, delete, modify, and print lines of text. While most commands are line-number oriented, string search and substitution commands are available. Commands are discussed below roughly in order of increasing complexity. It is suggested that you begin by reading Sections 1 and 2, then do some text editing. Successive sections describe more elegant functions and may be consumed one at a time.

A command to the editor consists of one or two characters followed by a list of arguments. The input format is free field i.e., spaces are ignored except that they delimit numbers and identifiers. Tabs are treated as multiple spaces.

1.1 OPERATION

To start SOS, simply type SOS to the Exec's @. Alternately, there are provisions within LISP, RLISP, REDUCE, and SAIL, to link directly into the SOS editor at need.

When SOS is started from Exec-level, it will inquire 'FILE='. You may reply in any of three different ways, depending upon your immediate purpose:

1) If you want to read an existing file, then simply type in the 10X filename, followed by a return; e.g., <SOMEONE>FOO.BAH<cr> .
2) If you want to create a new file, and name it now (rather than later), then type an altmode followed by the desired name, and a return. The name will be remembered at W- or E-time.
3) If you want to create a file, but prefer to defer naming it until later (the first World or End command), then reply with just a return.

When SOS is ready to accept a command, it will type '*'. If FOO isn't found or is illegally specified, SOS will say so, then re-type 'FILE=' and expect you to give another name.

If the file name includes a directory name other than the one you are connected to, the original file will not be changed, but a new version will be created in your area. Similarly, if the file being read is on Dectape, it will not be deleted by SOS (since it can't be undeleted in the event of error).

If you wish to read a file but not modify it, you may type ',R' after the input filename, which puts you in read-only mode. In this case, any attempt to modify the file will give a non-fatal error message.

1.2 FILE HANDLING

SOS currently works by recopying the text of your file with corrections and additions onto a file whose name it invents. The name of this file is of the form $ED$jj.TMP, where jj is your current job number. When the edit is finished, SOS renames this temporary file to the name of the user's original file unless a new name is specified (see the E command).

If the user attempts to reference a line which occurs earlier in the file than the one last referenced, SOS may have to finish the current copy and start copying over again. This process is automatic but does take time, so there may be a delay in executing certain commands. Notice that the form of editing used means that if the user calls the system without saying either "W" or "E", his original text file will be unchanged, so any editing will have been lost.

SOS keeps a portion of the file in core during the editing process so that a certain amount of backup can be done without recopying. SOS will use all the core it is given for this purpose. However, note that running SOS in a large amount of core may result in much poorer response time.

There are certain error messages which may occur at almost any time during an edit and which cause the editor to call exit. Note that this will cause the edit to be lost, but the original file will not be harmed. These errors are as follows:

DEVICE OUTPUT ERROR	Output error. No recovery will be attempted.
DEVICE INPUT ERROR	Input error. No recovery will be attempted.

INTERNAL CONFUSION or ILLEGAL UUO or TENEX CONFUSION
SOS has just discovered a bug in itself.

In addition to the above, there are two messages which are primarily warning messages. The first of these is *LINE TOO LONG*. This means that some line in your text is too long (more than 147 characters).

The line will be shortened to 147 characters and printed. The second message is *OUT OF ORDER*, which indicates that some line of your input has a number lower than the line before it. The line which is out of order will be printed. In both of the above cases, the page number on which the error occurs will also be printed.

The error *ILLEGAL LINE FORMAT* may occur at any time. This probably means that you have a line containing a <return> not followed by a <line feed> or a <line feed> not preceded by a <return>. The best cure is to replace the line completely (R command).

1.3 SPECIFYING LINES AND RANGES

The text file is organized in terms of pages which are subdivided into lines. The pages are numbered sequentially starting at 1 and continuing to the end of the file. The division into pages is determined by the user rather than by the editor. Page marks (which indicate the start of a page) may be inserted and deleted by the user. Page numbers are "floating"; that is if page mark 2 were deleted, the page which was formerly numbered 3 would now be page 2.

In contrast to this each line on a given page has a line number which is "sticky". That is, a given line will retain the same number regardless of insertions or deletions. The lines are usually numbered by 100 or some other increment larger than 1 to allow room to insert new lines. Several pages may have lines of the same number.

Most SOS commands refer to either a single line or a range of successive lines. A single line is specified by giving both the line number and the page number in the form <line number>/<page number> as 100/3 for line 100 on page 3. A range is specified by giving the first and last lines of the range separated by a colon as 100/3:4702/6 for line 100 on page 3 through line 4702 on page 6. Alternatively, a range may be specified by a starting line and a number of lines. For example, 200/4!10 refers to the 10 lines beginning on line 200 of page 4. In either case, this construction is referred to as a range specifier.

Instead of a number the symbol "." may be used. "." means either the current page or the current line depending on whether it appears after or before the /. Thus 100/. means line 100 on the current page and ./3 means the current line on page 3 (the line on page 3 which has the same line number as the current line). The current line and page are determined by the last command which was executed (see the individual commands for further details).

For ease of use, some of the specifications may be omitted. If the page number of the specification is omitted, it is assumed to be the current page (in this case the slash is also omitted). Thus 400/. and 400 specify the same line. If the page number for the end of a range is omitted, it is assumed to be the same as that for the start of the range. Thus 400/7:3120/7 and 400/7:3120 specify the same range. So do 400/.:700/. and 400:700.

If just a page number is given with no line number, it means all the lines on that page. Thus /3 means all the lines on page 3 (except for the delete command). Omitting the first line number of a range means the first line of that page, while omitting the second line number means the last line on that page. Thus /3:/5 specifies all the lines on pages 3 through 5. It is not legal to omit both the line and page number (thus :100/6 is not legal).

Relative page and line numbers may be used instead of absolute ones. Thus 100+3/. .+27/4 and .-5/.+6 are all legal specifications. For pages this has the obvious meaning (if you are on page 5, then /.-4 is page 1). For line numbers however, .+n means the nth line after the current one. Thus if a file has lines numbered 100, 103, 106, 109, 111, 142, and 200 and if the current line is 100, then .+3 is line 109 not line 103. The start and end of the page act as boundaries for relative line numbers as follows: If a page has lines numbered 100, 200, 300, 400, 500, and 600, and if 300 is the current line, then .+3, .+4, etc. are all line 600. Similarly .-2, .-3, etc. are all line 100.

The symbol "*" may be used to specify the last line on a page. "*" may not be used to specify a page (use =BIG to find the last page explicitly). Expressions such as "*-4" are permitted.

If you are not used to line-numbers, and/or would prefer not to see them, and/or are used to TECO or another string editor, you may suppress the appearance of line-numbering by the '_Suppress' switch (see the Set command).

1.4 TRANSLITERATION

SOS is capable of using the full 128 character character set through Teletype keyboards. To do this it uses ? to give each character on the teletype a second meaning. Thus, typing ?2 on the teletype causes ^W to be entered into the file (octal code 027). Similarly a ^W in the file will type out as ?2 on the teletype. To enter a ? one must type ??. To enter lower case characters through an upper-case-only keyboard, precede each letter with a "?", or use the shift commands (_LOWER and _UPPER, see Section 4).

Below is a list of the alternate meanings of the various non-alphabetic characters:

Ascii	?x	^x	Stanford name
1	?!	^A	down-arrow
2	?"	^B	alpha
3	?#	^C	beta
4	?$	^D	and
5	?%	^E	not
6	?&	^F	member,epsilon
7	?'	^G	pi,bell
10	?(	^H	lambda
11		^I	tab

16	?)	^N	infinity
17	?*	^O	del
20	?+	^P	horseshoe-right
21	?,	^Q	horseshoe-left
22	?-	^R	horseshoe-down
23	?.	^S	horseshoe-up
24	?/	^T	for-all
25	?0	^U	there-exists
26	?1	^V	circlex
27	?2	^W	iff
30	?9	^X	underline
31	?6	^Y	right-arrow
32	?4	^Z	tilde
33	?=	^[	not-equals
34	?<	^\	<=
35	?>	^]	>=
36	?7	^^	equivalence
37	?8	^_	or (see App.A)
140	?@		accent breve
173	?[		left-bracket
174	?:		vertical stroke
175	?]		right-bracket

All other characters have the same meaning whether preceded by a question mark or not. SOS will also do this sort of conversion on Teletype output. Vertical tabs and formfeeds are ignored.

If you are on a display, no question-mark conversion will be done, because the displays have the full character set. If you are on a TTY Model 37 or an ARDS display, use the "_M37" command (one of the Set commands) to see lowercase and the curly-brackets.

2. BASIC COMMANDS

Whenever SOS types a '*' in column 1 (except after a 'P,S' command), it is in command-mode, ready for input, and waiting for you to enter a command. You can edit any command with ^A (or rubout), ^R, and ^X. Except in the case of giving a filename, ^X will always abort the current command. When typing in a filename (Copy, End commands, etc.), ^X may only clear the current partial filename and expect another; you can tell by whether a '*' appears.

As at Exec-level, you may preface a command-line with a ';' and the line will be ignored as a comment. This is handy if someone links to you. Similarly, a '?' will inform you of all possible SOS commands, though not the details of individual syntax. You may also type Help, and SOS will interact and try to explain what you don't understand.

The syntax of each command below is shown with optional arguments enclosed in parentheses. Where two or more arguments are shown with "|" between them, it means that any one (but only one) may be used. Thus the I command, below, may take any of the following forms:

```
                  I
                  I,<increment>
                  I<line>
                  I<line>,<increment>
```

Insert--I(<line>)(,<increment>)

The insert command is used to insert new lines into the file. Insert accepts a single line specifier as its argument and begins inserting at that line. Each time you complete a line, SOS will add the current increment to the number of the line just inserted and try to insert that line. The current increment may be set by giving a second argument to the insert command. Thus I100,30 will start inserting at line 100 and set the increment to 30. The increment is set to 100 at the start of editing.

When inserting, SOS will type out the number of the next line to be inserted. The user should then type the desired text of that line followed by a return. SOS will then either print the line number of the next line to be inserted or will return to command mode and print a "*".

To stop inserting, type an ALTMODE. This causes the line in which it appears not to be inserted and returns SOS to command mode. When an altmode is given, the number of that line is remembered and will be used if the next "I" command is given with no arguments.

If the first line to be inserted specifies a page which does not exist, SOS will respond with *NO SUCH PAGE*. While inserting, you may correct typing errors using the system editing commands (i.e.,

^A or RubOut, ^R, and ^X).

Example:

```
*I100
00100  NOW IS THE
00200  TIME FOR ALL
00300  <altmode>
*p100
00100  NOW IS THE
*I
00300  GOOD MEN
00400  ...
```

If the line at which inserting is to start already exists, an insert will be done on line <number> + <current increment> unless there is a line with a number between <number> and <number> + <current increment> in which case the line number will be halfway between <number> and the number of the next line in the file. If, however, any subsequent line which is to be inserted already exists, or if a line with number between two consecutive lines of the insert exists, then the insert will terminate and SOS will return to command mode. As an example, suppose a file has lines numberd 100, 200, 300, 400, and 500. The command I120,40 would allow lines 120 and 160 to be inserted and then would automatically return to command mode. The command I120,20 would allow lines 120, 140, 160, and 180 to be inserted and then would return to command mode.

The current line and page are set to the last line actually inserted (not the one terminated with altmode) on the specified page. If an attempt is made to insert a line containing more than 147 characters (not counting the return at the end) the error *LINE TOO LONG* will be given, the line will not be inserted, and SOS will return to command mode. If the next line to be inserted would have a number greater than 99999, SOS will stop inserting and return to command mode.

Delete--D<range>

The delete command will accept a range specifer as its only argument and will delete all of the lines specified. If there were no lines in the range specified, SOS will respond with *NO SUCH LINE(S)*. There are two exceptions to the normal manner of specifying lines. The delete command will not allow a page specification for the second line number of a range. Thus the command D100/5:200/6 is illegal and will result in the error message *ILLEGAL COMMAND*.

The second exception is that the command D/5 will not delete all of page 5, but will instead delete page mark 5. This may result in the error message *OUT OF ORDER*, indicating that the deletion results in a page on which there are some sequence numbers not in proper order (e.g. 100, 200, 300, 150, 200). Note that the deletion has been made. The correct procedure at this point is to renumber all of the lines on the appropriate page (4 in the above example). See the Number command.

All of the lines on page 5 may be deleted by the command D0/5:99999. An attempt to remove page mark 1 (D/1) or some page mark which does not exist will result in the error message *NO SUCH PAGE*. The current line and page are set to the last line deleted on the specified page. Note that after a deletion, the command P. will give *NO SUCH LINE(S)*.

```
    Print--P(<range>)(,S)
    Output--O(<range>)
```

The print command accepts a range specifier as its argument. The lines specified will be printed on the teletype. The current line and page will be set to the last line actually printed. If the range specified has no lines in it, the error *NO SUCH LINE(S)* will be given. If the range of printing includes the boundary between two pages, "PAGE n" will be printed to indicate the presence of the page mark. The command P<return> is the same as P.!16<return>.

If a second argument of ,S is given, line numbers will be suppressed in the printout. This is useful for clean copies on a TTY model 37. In addition, page numbers will not be printed, and the * for the next command will be suppressed, so that copies so generated will be absolutely clean. The O command is equivalent to P,S .

When the '_Suppress' switch is on, then the sense of 'P,S' and 'O' are reversed: the line-numbering IS shown for the given range.

```
    List--L(<range>)(,S)
```

The List command is not yet applicable at Utah, since we lack an online printer. Use RQLIST.

List is like Print in format and error messages, but the output goes to the line printer instead of the teletype. Page headings will be printed at the top of each line printer page: the name of the file, the time and date of printing, and the page number. Page numbers are given in the form M-N. where M is the actual SOS page number on which this text can be found and N denotes the N-th page of line printer paper required for logical page M. The current line and page are set to the last line printed. If no range specification is given, the entire file will be listed.

The S option for suppressing line numbers and headings also applies to the List command. Unlike the print command, however, the * for the next command will print, as there is no reason for it not to.

End--E(<file name>)(,S)

This command is used to terminate the edit. If no arguments are given, the old copy of the file being edited will be deleted and the new copy will be renamed to the old name, with a higher version number. If an argument is given it will be taken as a file name (it should be of the form foo. or foo.baz) and the new copy of the file will be given this name.

If the original source file (assuming you're not creating) had a non-standard protection code, this will be propagated to the new output.

You may, if pertinent, give the old filename with a ';Ø' suffix. In this case, the old file is deleted (irredeemably by 1ØX) and the new one is substituted.

If you wish the file to be pristine, without line-numbers, then add the ',S' tail. The SOS file will be saved and then stripped, just as if by the CUSP Pippy (see Appendix A -- Data Conversion).

SOS will return to the monitor when it has finished.

To abort the E or W command, use rubout (and ^X after a comma).

No spaces are permitted between the E and the filename or comma;
E. is not correct.
Efilename (when the input had an extension) will produce a file with that extension --- as is the Tenex usual convention. To avoid this, use a period after the filename.

3. INTERMEDIATE COMMANDS

save World--W(<file name>)

The W command is the same as the E command except that it leaves you editing in the same place. This command is useful for saving the current version of the file in case the system should die. File names given to W commands are "sticky", and need not be given twice.

In case the system dies and you have not done a W recently, try the following procedure: type the system command 'DIR *.TMP'; a temporary file with your previous job number may exist. If one does, it should be the one SOS was using when the system died. Edit the file to see if it is. If it is not, delete it and try again. If this fails, you are authorized to tear your hair (and ask yourself why you didn't do a W).

Every now and then, say "DEL *.TMP" to the system to get rid of accumulated garbage temporary files, which are caused by system crashes or by saying ^C during editing.

The SAVE and ISAVE parameters may be set to cause automatic "W" operations at regular intervals. See the Section 4 for parameter-setting details.

Go--G(<file name>)

This command is the same as the E command, at Utah. It ignores the ',S' if any, however.

Mark--M<line>

This command is used to insert a page mark into the text. It accepts a single line specifier as argument and places the page mark immediately before the line specified. Note that this will increase the page numbers of all following pages by 1. The current line is set to Ø and the current page is set to the new page. Thus if the command M472Ø/5 is given the current page will be set to 6.

If the line specified does not exist, the page mark will be inserted immediately before the line of next higher number on that page. If the page specified does not exist, the error *NO SUCH PAGE* will be given and no page mark inserted.

Hint: When inserting new text, to insert a page mark at the end of the current page use M99999.

Number--N(<increment>(,<range>(,<starting number>)))

This command is used to alter the numbers of currently existing lines. It takes Ø, 1, 2 or 3 arguments. The command "N" with no arguments causes the entire file to be renumbered with an increment of 1ØØ.

The first argument is the increment to use in the renumbering. The first line renumbered will be given this number (unless there is a third argument) and each succeeding line will be given a number which is the sum of this argument and the number given the last line. If the renumbering crosses a page boundary, the first line on the new page will be given this number again. The current line and page will be set to the new number of the last line renumbered. Thus if page 3 has numbers 1Ø7, 254, 5ØØ and page 4 has numbers 27, 39, 1Ø8, and the command N2Ø,/3:/4 is given, the new numbers on page 3 will be 2Ø, 4Ø, 6Ø and on page 4 will be 2Ø, 4Ø, 6Ø and the "current position" will be line 6Ø on page 4.

If there is no second argument, the entire file will be renumbered. If the second argument specifies only a single line, only that line will be renumbered. If there are no lines in the range specified, the error *NO SUCH LINE(S)* will be given.

Note that if only portions of a page are renumbered, a situation can be created in which sequence numbers are out of order. If this happens the error message *OUT OF ORDER* will be given, however the renumbering has already been done. The best way to correct this situation is to renumber the entire page on which the error occurs.

If the third argument is present, it is used as the number for the first line renumbered. Thus if page 3 has lines 400, 700, 905, 1233 and the command N100,/3,47 is given, page 3 will now have numbers 47, 147, 247, 347. This feature is useful in renumbering a page before deleting a page mark in order to avoid an *OUT OF ORDER* error.

If the renumber increment is 0, the error *ILLEGAL COMMAND* will be given. If the increment is too large, i.e. some of the line would have numbers greater than 99999, the high order digits of the large numbers will be lost and the error *WRAP AROUND* along with the page on which the error occurs will be printed. Note that this leaves the page with line numbers out of order so it should be renumbered with a smaller increment.

Display Alter-- Z(<range>)

This command is valid only if you are using a Stanford display console or an Imlac. Each line in the range is sent back to the time-sharing system for alteration. Initially, the text-editing cursor will be set at the first character in the line. All editing features which are operative while inserting text (using the "I" command) are available for altering the line.

Alter--A<range>

This command is used to make changes within a line without having to retype the entire line. It accepts a range specifier indicating the lines to be altered. For each of the lines in the range, it prints the line number and then enters a special intra-line editing mode which has its own commands. These commands are not echoed on the teletype, so that the line shown on the teletype at the end of the intra-line edit is nearly the same as the line which will appear in the text.

The intra-line editor maintains a pointer within the line being changed. This pointer points to the character which the next command will effect. The pointer is initially placed pointing to the first character of the line. In general, any command in this mode may be preceded by a number which will cause it to be repeated that number of times. For example, the command 10D causes 10 characters to be deleted.

next--<space>

This command causes the character pointed to to be printed on the teletype and the pointer to be moved right one character. If the pointer is already at the extreme right of the line, the command is ignored.

back--<rubout or ^A>

Moves the pointer to the left one character. If the pointer is already at the extreme left of the line, a <return><line feed> is done and the number of the line is printed again. The pointer then points to the first character of the line. The characters moved over are printed surrounded by \'s.

Thus, if the line being edited is "How now ..." and the intra-line commands 7<space>, 3<rubout>, <space> are given, the printed line will say "How now\won\n" and the pointer will be on the "o".

Change--C

This command causes a character to be accepted from the teletype. This character is printed and replaces the character pointed to. If the pointer is at the extreme right of the line, the command will be ignored. <rubout> from the teletype will be ignored, but ^A, ^R, or ^X will be used as legitimate replacements. <line feed>,<return>, or <altmode> will cause the remainder of the C command to be aborted. This is useful if a number was used which proves to be too large.

Delete--D

This command deletes the character pointed to. The deleted characters will be printed surrounded by \\'s. If the pointer is at the extreme right of the line, the command is ignored. After the command, the pointer will be pointing to the character to the right of the one deleted. If this command is preceded by a number, only the last 3 characters deleted will be printed, surrounded by \\'s.

Insert--I

This command causes characters from the teletype to be printed and inserted into the line just ahead of the pointer until an <altmode> is seen. The pointer is left pointing at the character to the right of those inserted.

<rubout or ^A> causes the character to the left of the pointer to be deleted. This character will usually be the last character inserted, but it is possible to delete more characters than were inserted. Typing <rubout> when at the left end of the line has no effect. The

characters deleted are printed surrounded by \'s.

^R will perform as if '<altmode>PI' were done.
^X will abort the line alteration as if '<altmode>Q' were done.

If enough characters are inserted to make the total length of the line more than 147, the error *LINE TOO LONG* is given and SOS returns to command mode without having made any changes in the line being altered.

If a <return> or <line feed> is seen, a <return><line feed> will be inserted at that point. This will create a new line whose contents are that part of the line to the right of the present pointer position. The number of this new line will be determined as follows: If the I command was preceded by a number, this number will be added to the current line number to produce the "provisional line number". If there was no number preceding the I command, the "provisional line number" will be created by adding the current insert increment to the current line number. If the "provisional line number" can be used without producing an order or wrap around error, it will be used. If not, the new line will be given a number which is halfway between that of the current line and the number of the next line. If the number of the next line is only one more than the current line number, an *OUT OF ORDER* error will be given, SOS will retype the line number and the contents of the line to the left of the pointer. SOS will then be ready to accept more characters to be inserted.

If no errors occur the pointer will be left pointing at the first character of the new line and the current line will be set to the new line created.

finish--<line feed> or <return>

Causes the part of the line to the right of the pointer to be printed and the intra-line edit to be finished. If any lines remain in the range specified for the alter command, the intra-line edit of the next line is started, otherwise SOS returns to command mode.

Quit--Q (or ^X)

Causes intra-line editing to be terminated without having made any changes in the line being altered. SOS returns to command mode. This command is useful if you discover that a mistake is being made since it restores the line being altered to its original state.

Start over--<control U>

This command causes SOS to start the intra-line edit of this line over. The line is restored to its original state and intra-line editing is re-started. It is equivalent to typing <altmode> and then giving the alter command again.

Skip--S

This command accepts one subsequent character from the teletype without echoing it (note that on a teletype ?1 is counted as a single character) and moves the pointer to the right until it points to the next occurrence of that character. For example to go to the next M type SM; to get to the third m type 3Sm. It will print all of the characters that it passes over. The character currently pointed to is printed but not compared. If there are no further occurrences of the specified character the pointer will be moved to the extreme right end of the line. ^A, ^R, and ^X are legitimate.

Kill--K

This command is the same as S except that it deletes all of the characters it passes over instead of printing them. If there are no further occurrences of the specified character on the line, the command will be ignored instead of deleting the remainder of the line. As with the D command, the last 3 characters deleted will be printed, surrounded by \\'s.

Replace--R

<number>R is exactly equivalent to <number>DI.

Line--L

This command prints the remainder of the line to the right of the pointer, then returns and prints the line number and leaves the pointer on the first character of the line in Alter mode.

Print--P (or ^R)

This command prints the remainder of the line to the right of the pointer, then returns and prints the line up to the position it was in when the command was given.

Justify--J

This command inserts a <return> <line feed> at the place the pointer is currently pointing and then concatenates the portion of the line to the right of the pointer onto the start of the next line. The pointer is left positioned at the start of the next line. The current line will be set to the new line number.

This command is intended to be useful principally in hand justifying text. The error *LINE TOO LONG* will be given if the new line created is longer than 147 characters. The error *NO NEXT LINE* will be given if this line is the last one on this page. Either of these errors will cause the J command to be ignored and the line number and portion of the line to the left of the pointer to be typed out.

Any other commands to intra-line edit mode will be ignored.

4. OTHER USEFUL COMMANDS

Replace--R<range>(,<increment>)

This command is the same as a delete command followed by an insert command. It accepts a range specifier and an optional second argument (separated from the first by a comma) which if present will be used to set the increment. It performs a D command using the range specifier given and then an I command with the first line specified by the range specifier.

There are some slight differences between the R and D commands. Whereas D/3 will delete page mark 3, R/3 will replace all of the lines on page 3. In addition, the R command will never give *NO SUCH LINE(S)* error messages.

Beginning--B

This command simply repositions you to the start of your file. It is chiefly useful in avoiding a range specifier to Find or Substitute commands, i.e., always saying 1/1:....

next line--<line feed>

This command causes the next line of the file to be printed. If the current line is the last of the file, the error NO SUCH LINE(S) will be given. If the current line is the last one on the current page, PAGE n will be printed where n is the number of the next page. If there are several blank pages the following typeout may result:

```
PAGE 10
PAGE 11
PAGE 12
PAGE 13
00100   This is the first line on page 13.
```

previous line--<altmode>

This command prints the line before the current line. If the current line is the first line of the file, the error NO SUCH LINE(S) will be given. Page numbers may be printed as in the <line feed> command.

Copy--C<dest>(_<file>(,S)),<source range>(,<incl>)(,<inc2>)

The Copy command will insert a copy of a given piece of text in a given location. The source for the text may be on the file being edited or on some other file. The basic form of the Copy command is:

```
        C<destination>,<source range >,<increment>
```

The Copy command acts as if an I<destination>,<increment> had been done and then all of the lines specified by the <source range > had been typed in. The current line is set to the last line entered.

If the <increment> is large enough that it would cause an *ORDER* or a *WRAP AROUND* error, the Copy command will pick a smaller increment. The message "INC1=<number>" will be printed to show what increment was chosen. If it is impossible to choose a small enough increment, either "INC1=ORDER" or "INC1=WAR" messages will appear and the given increment will be used.

Since all of the text to be copied must be contained in core at one time, copying huge blocks of text may result in the error message *INSUFFICIENT CORE AVAILABLE*. This should never happen, of course. The only possible solution is to copy several smaller blocks.

If the source lines contain page marks, the renumbering of lines will cease when the first page mark is reached. Lines between the first and last page marks will be inserted with their original numbers. Lines after the last page mark will be inserted with their original line numbers unless a second increment is given. This increment should appear immediately after the first increment and be separated from it by a comma. If the second increment is so large that *ORDER* or *WRAP AROUND* errors would occur, or if no second increment is given and an *ORDER* error would occur if no renumbering were done, SOS will choose an increment to use. SOS will print "INC2=<number>" to indicate the increment chosen. If there is no suitable increment, SOS will print "INC2=ORDER" and use the specified increment or the original line numbers if no second increment is given.

If the source lines are to be on some file other than the one being edited, the Copy command is given as:

```
        C<destination>_<source file name>,<source range specifier>...
```

This command may give the error *FILE NOT FOUND*.

A special form of the Copy command is available for copying lines from another file when the line numbers are not known. The command:

```
        C<destination>_<source file name>,S
```

will cause SOS to respond with a "*". The file indicated is now being edited in read-only mode. The P,L, and F commands may be used to find the desired lines. After the lines have been found, say "E". SOS will respond with "SOURCE LINES=". At this point, type the remainder of the C command string (<source range specifier>...).

```
    Transfer--T<dest>,<source>(,<incl>)(,<inc2>)
```

The Transfer command moves a set of lines from one place on a file to another. It acts like a Copy command followed by a Delete command. It has only two differences from the Copy command. The first difference is that the source lines must be on the file being edited. The second difference is that the error message *ILLEGAL TRANSFER DESTINATION* may be given. This error will occur when the destination is inside the source range. (i.e. T400/5,/2:/9)

If the deletion would produce an order error because of the removal of a page mark, SOS will reinsert one page mark and type

```
        PAGE MARK INSERTED TO PREVENT ORDER ERROR
```

If the destination specified is on a page which does not exist, SOS will insert the text at the end of the file and type

```
        TEXT INSERTED AT END OF FILE
```

eXtend--X<range>

This command is like the Alter command except that on each line it automatically puts the pointer at the right end of the line in insert mode. It is useful for adding comments to lines.

display extend-- Q(<range>)

Ignore this command at Utah, unless you are an Imlac.
This command is like the eXtend command, except that the system line editor is used. The pointer is positioned to the end of the line; you are therefore in a mode ready to append characters.

set--_<parameter>(=<number)

The set commands enable you to change certain modes and parameters that control the operation of SOS. For the various mode commands below, the first one given represents the initial condition of the editor, while the parameter commands show the initial values.

case shift-- _UPPER   _LOWER

When operating on a Model 33 Teletype, it is often convenient to invert alphabetic case shifts. This makes it easier to edit text which is largely lower case. The command _LOWER will cause the editor to enter the characters A through Z as their lower case counterparts and ?A through ?Z as upper case. The command _UPPER reverses this so that A through Z are taken as upper case letters and ?A through ?Z are taken as the lower case.

Note that this may result in confusion if you are using a model 37 where it is possible to type in lower case letters directly. If this is the case, saying _LOWER will cause the state of shift to be inverted for letters. This case inversion also happens in a similar fashion on output.

character set-- _DPY (for displays)
_M33 (for teletypes) _M37

If you are running on a Model 37 Teletype, you should say _M37, which will cause both upper and lower case letters as well as "|", "{", and "~" to be printed directly rather than in the form ?A, ?:, etc. To set to "not a 37" use _M33.

The _DPY lets you get back into the right mode when running on a display, even though you may have been so foolish as to get into one of the other modes.

transliteration-- _C128 _C64

If you are operating on a teletype and would like to be able to type a single ? for ? instead of ??, a mode is provided which turns off the special properties of ? on input. Note that ? will still print as ?? unless you have said _DPY. To enter this mode type _C64. To leave it use _C128.

messages-- _NOVICE _EXPERT

After you have used SOS for some time, you may want to have messages printed in a shorter form. Saying _EXPERT will cause all error messages to be abbreviated to three characters (thus *ILC* for *ILLEGAL COMMAND*). To go back to full printout of error messages say _NOVICE.

line numbers-- _SHOW _SUPPRESS

The initial mode of SOS is to always show line-numbers on Teletype output. This mode may be reversed by _SUPPRESS', which also reverses the sense of the commands 'P,S' and 'O'. The line-numbers will still appear in the output file unless you choose to do an 'E,S'.

justification-- _PMAR=1 _LMAR=1 _RMAR=69 _MAXLN=99999

You may change the parameters used by the justification commands (JU, JL, JR, JC). PMAR is the beginning column for the first line of each paragraph, LMAR is the left margin for all other lines, RMAR is the right margin, and MAXLN is the maximum line number. Thus, to indent the first line of each paragraph 4 spaces (to column 5), say _PMAR=5.

line spacing-- _INC=100

The line increment used by the I command and others may be set directly.

file saving-- _SAVE=34359738367 _ISAVE=34359738367

You can cause the W command to be executed automatically at certain intervals, thus causing your edit file to be saved. ISAVE controls the number of lines of text which will be inserted (using the "I" or "R" command) between "W" operations. When SOS decides to save your file, it types "SAVING" on a new line. When the "W" operation is complete, SOS proceeds with the I or R operation. You may wait quietly while all this is happening, or continue to type -- SOS will catch up with you when it is through saving your file.

SAVE indicates the number of file-altering commands (A, X, D, I, R, etc) SOS will accept between automatic "W" operations. The actual saving operation is similar to that described for ISAVE, except that the editor is in command mode, not insert mode, after the operation. Note that the initial values of SAVE and ISAVE, given above, will cause automatic file saving to occur rather infrequently.

give information--=<parameter>

This command is used to determine any of several pieces of information which are not otherwise available, such as the modes set by the _ command. The following variants of this command are available:

==	Prints a summary of this list of things you may inquire about.
=_	Prints a short list of the parameters you may set with _.
=.	Prints the current line and current page (e.g. "5ØØ/3").
=BIG	Print the page number of the largest page in the file.
=CASE	Prints out either UPPER or LOWER to indicate the mode set by the _ command. If the user has done a _M37 command, SOS will print either MODEL 37 UPPER or MODEL 37 LOWER. If the user is on a display, SOS will print DPY in front of the UPPER or LOWER. If a _C64 has been done, C64 will also be printed.
=PMAR	Prints the current left margin for the first line of a paragraph not begun with a tab, as used by JU.
=LMAR	Prints the current left margin used by JU, JL, and JC.
=RMAR	Prints the current right margin used by JU, JR, and JC.
=MAXLN	Prints the maximum line number currently allowed by JU.
=INC	Prints the current increment (for I or R commands).

=ISAVE Gives the current ISAVE figure.

=SAVE Gives the current SAVE figure.

=ERROR Prints out the last error message given. Printout will always be done in full regardless of any _EXPERT commands which may have been given.

=FILE Prints the current name and version of the file you're editing, or the fact that you deferred the naming.

=STRING Prints out the three strings used by the Find and Substitute commands. (See next section for further details). The strings are printed with the titles SEARCH, SUBSTITUTE, and FOR. The titles are indented while the strings start at the left margin.

5. ADVANCED COMMANDS

Join--J<line>

The join command is used to join two successive lines into one. Its argument is the first of the pair of lines to be joined. The new line formed will be given the number of the first of the pair. The error message *LINE TOO LONG* may be given, in which case, the lines will be unchanged. If the line given is the last line on a page, the error message *NO NEXT LINE* will be given. The current line will be set to the line created if there are no errors.

JUstify--JU<range>

This command takes a range as its argument. Note that there are two letters in the command. It justifies the text in the range by ignoring all line numbers, carriage returns and line feeds in the range, and inserting its own in such a way that adding an extra word to a line would cause its length to exceed RMAR(right margin) - LMAR(left margin)+1 characters. (See the "_" command, above, for a list of initial values for parameters.) A word is taken to be anything between blanks. The end of a line is considered a blank. If a word ends in ".", "?", or "!", two blanks are permitted after it. Otherwise, only one blank is permitted, and others are ignored.

Next, extra blanks are inserted between words, starting from the left and right on alternate lines, to make the length of the line exactly RMAR-LMAR+1. Then LMAR-1 blanks are inserted in front of the new line, and it is given a line number which is the same as if the new, justified text had been numbered with a N<INC>,<range> command. A pagemark is automatically generated if the line number for the next line would exceed MAXLN. You will be told if this occurs.

There are exceptions to the above proceedure, all having to do with paragraphing. Any of the following conditions are treated as the end of a paragraph: TAB in first column (note that tabs in other places do not start paragraphs), BLANK LINE, PAGE MARK, BEGINNING or END of RANGE.

When one of the above conditions is encountered (except of course beginning of range), the immediately preceding line is not expanded. It is, however, moved out to LMAR by the insertion of leading blanks, if necessary.

If the new paragraph begins with a TAB, the tab is merely inserted into the text. If not, the first line of the paragraph is made to start at PMAR (paragraph margin) rather than LMAR.

Justify Left--JL<range>

Lines in the range are left justified by removal of leading blanks, and the insertion of LMAR-1 leading blanks to move them out to the left margin. Paragraphs have their first lines treated as in the JU command. No chopping, filling, renumbering, etc. is done.

Justify Right--JR<range>

Like JL, only enough blanks are inserted to move the line out to the right margin. The required blanks are inserted to the right of the rightmost tab in the line.

Justify Center--JC<range>

Like JR, only half as many blanks are inserted, so that the line ends up centered between LMAR and RMAR.

For a summary of the parameters used in the Justify family of commands, their initial values, meanings, and how to change them, see the Set(_) and Give(=) commands.

Find--F(<string>)<altmode>(<range>)(,A|,N)(,E)(,<number>)

The Find command is used for locating occurrences of given strings of text. The basic form of the Find command is:

```
        F<string><altmode><range>
```

The first occurrence of the specified string within the specified range will be found and the line containing that string will be printed. If the range includes more than one page and the line found is not on the first page of the range, PAGE n will be printed where n is the number of the page on which the line occurs. For example, to find the first occurrence of the string "FOOBAR" on page 5, use the command:

```
        FFOOBAR$/5
```

where $ is used to indicate an altmode. To use the same string used by the last F command, simply omit the string (but not the <altmode>). For example, after the above search, to find the first occurrence of the string "FOOBAR" on page 14, use the command:

```
        F$/14
```

Note that it is possible to determine what string will be used in such a case by using the =STRING command. If no previous F command was done, the error message *NO STRING GIVEN* will be printed if the string is omitted. Upper and lower case letters will be considered the same inside <string>. Thus the strings "FOO" and "?F?O?O" will find the same lines. The use of <return>, <line feed>, ^^, ^E, ^N, ?:, and ?/ in search strings should be avoided until you learn how to use them. (See the section below on Special Characters).

If the range is omitted (e.g. FFOOBAR$) then the range searched will be from the line after the present one (essentially .+1) to the end of the file. To search from the present position to some location, give only the second half of the range. Thus to search from .+1 to the end of page 1Ø, use:

```
        FFOOBAR$:/1Ø
```

Giving the command F<return> will cause the search to continue from the present point. This differs from the command F$<return>, which searches to the end of the range specified in the previous search command instead of continuing to the end of the file.

If no occurrence of the string is found in the given range, SOS will simply print a * and wait for the next command. The current line will be set to the last line found. If no line is found, the value of . will be unchanged.

Multiple Strings

To search for more than one string at the same time, separate the strings by a <return>. Thus to find the first occurrence of either "FOO" or "BAZ" on page 5 use the command:

```
FFOO
BAZ$/5
```

If too many strings are specified in this manner, the error message *TOO MANY STRINGS* will be given. The current limit on number of strings is 6. The error message *STRING TOO LONG* will be given if the total length of all strings being searched for is greater than the table space available (currently 2ØØ characters).

Alter switch (,A)

It is possible to cause SOS to enter intra-line edit mode automatically when a string is found. To do this, append ",A" after the range when giving the F command as:

```
        FFOOBAR$/5,A
```

or   FBAZ$,A   (range omitted)

When a match is found while using this feature, SOS will enter intra-line edit mode (A command) and move the intra-line edit pointer to point to the first character of the string found (using the <space> command). If the F<return> command is used after editing of that line is finished, the ",A" will remain in effect. Thus F<return> is really a continue command even as far as special modes are concerned. This effect of the F<return> command also holds for the N and E modes explained below.

Line Numbers only (,N)

Occasionally it is sufficient to know just the line numbers on which a given string occurs. This is especially true on teletypes where printing takes a great deal of time. For this reason, SOS allows ",N" to be added to an F command immediately after the range. This will cause only the line number to be printed when a line is found.

Exact compare (,E)

If it is undesirable to have upper and lower case letters treated as being identical, a ",E" may be included in the command string. It should occur immediately after the ",A" or ",N" if either is present, or after the range if both are absent.

Number of occurrences

It is possible to find more than just the first occurrence of a string. This may be done by ending the command string with ",<number>" where <number> is the number of strings to be found (99999 will almost certainly find all of the strings). This has the effect of giving the F command and then a series of F<return> commands until either the count is exhausted or the end of the given range is reached.

Special characters

Certain special characters may be included in the string to be searched for. Instead of being matched by themselves, they indicate a class of characters which may occur at that point in the string. These characters are as follows (in the form for Tenex teletypists):

?: Will be matched by any "separator". A separator is any character which is not a number, a letter, a ., a %, or a $. (i.e. a character which cannot be part of a symbol in MACRO or FAIL.)

?/ Will be matched by any character.

^E Will cause the character following it to be matched by any character which it would not normally be matched by. Thus the string F^EAB will be matched by FBB, FCB, FDB, F$B, etc., but not by FAB or FaB. ^E?: will be matched by any letter, number, etc. ^E?/ will be matched only by the begining or end of a line. Thus ^E?/FOO will find only those occurrences of foo at the start of a line.

^^ Is used to quote the next character. Thus ^^^E is used to search for the character ^E and ^^^^ to search for ^^. Note that ^E^^^E (or equivalently, ?%?7?%) will match any character but ^E (?%).

^N Is used to mean "any number of" whatever follows it. Thus the string A^NBC will be matched by AC, ABC, ABBC, etc. In case of ambiguity, the shortest such string will be found. Thus the string ^NAB will find B rather than AB. Strings of the form ^N^E^^^E are perfectly legal. The example will be matched by any number of characters which are not ^E's.

Certain strings which can be formed with the above characters are considered illegal and may give an *ILLEGAL SEARCH STRING* message. The strings are not checked before use, so the message will only be given when an attempt is made to check for a match with that particular part of the string. The illegal conditions are ^E, ^N, or ^^ when not followed by another character and the construct ^E^N... Due to the fact that some of these special searches involve recursion and others require the use of table space inside the editor, it is possible to get the error message *SEARCH STRING TOO COMPLEX*. If this happens, try a simpler string.

```
    Substitute--S((<ostring><altmode><nstring>)<altmode>
                                  (<range>)(,D|,N)(,E)(,<number>))
```

This command is used to substitute one string for all occurrences of another string. The basic form of the Substitute command is:

```
        S<ostring><altmode><nstring><altmode><range>
```

<nstring> will be substituted for all occurrences of <ostring> in the given range. Note that while the F command finds the first occurrence, the S command substitutes for all occurrences. The Substitute command will print all lines on which substitutions have been made. The line will be printed after all substitutions on that line have been made. As with the F command, "PAGE n" will be printed if the first line printed is not on the first page of the given range or if a subsequent line is not on the same page as a previous line.

For example, to change all occurrences of FOO to BAZ on page 17, use the command:

```
        SFOO$BAZ$/17
```

To use the same strings as were used by the last S command, simply omit both strings and one of the <altmode>'s. Thus if it were now desired to change all FOO's to BAZ's on page 33, one could use the command:

```
        S$/33
```

Note that as with F, it is possible to determine which strings will be used by using the =STRING command. If the strings are omitted and no previous S command has been given, the error message *NO STRING GIVEN* will be printed. Again as with F, upper and lower case characters will be considered the same in the first of the two strings (but not in <nstring>).

The effect of omitting the range or of specifying only the last half of the range is the same as for the F command. S<return> is a continue in the same manner as F<return> but is rarely needed since the S command affects all lines in the given range. The current line is set to the last line changed. If no substitutions are made, the value of "." is unchanged.

Multiple Substitution

As with the F command it is possible to do several substitutions at the same time. Several strings to be searched for, separated by <return>'s, may be given for <ostring> followed by an <altmode>, then several strings to replace them, again separated by <return>'s, are given for <nstring> followed by another <altmode>. The first string given for <nstring> will be substituted for the first given for <ostring>, the second for the second, etc. If more <ostring>'s than <nstring>'s are given, the last <nstring> will be used to substitute for the excess <ostring>'s. Thus to simultaneously substitute ALPHA for BETA and DELTA for GAMMA on page 5 through page 7 use the command:

```
SBETA
GAMMA$ALPHA
DELTA$/5:/7
```

The errors *TOO MANY STRINGS* and *STRING TOO LONG* will occur under the same circumstances as for F.

Decide switch (,D)

A special mode of the S command is provided in which the user has a chance to look at each line after substitutions have been made in it and to decide whether he wants the new line or the old one. To use S in this mode, put ",D" after the range in the command string. For each line in which a substitution is made, the line will be printed after all substitutions in it have been made. SOS will then wait for a single character to be typed on the user's console. If this character is <rubout>(or <BS>), the indicated substitutions will not be made and the old copy of the line retained. SOS will then proceed to look for the next line and repeat the process. If the character is E (or e), SOS will immediately return to command mode without having made the substitution. Any other character will cause the line as printed to become the new line and substitution to continue.

Numbers only (,N)

If the user is very sure of himself, he may suppress printing of those lines in which a substitution has been made. To do this put ",N" after the range in the command string.

Exact compare (,E)

As with F, ",E" will cause upper and lower case letters to be treated separately in the first string. This should come after the ",D" or ",N" if present and otherwise after the range.

Special Characters

All of the special characters permitted in the string of an F command (^N, ?/, ?:, ^E, and ^^) may be used in the first string of the S command. This may create a problem, however. Suppose it were desired to change all occurrences of FOO to BAZ but there were strings present containing FOO such as AFOO and FOOBAR. This can be circumvented by giving "?:FOO?:" as <ostring> but leaves the problem of replacing the separators found by themselves. All strings which match one of these special constructs are called partially specified strings. If the construct ?*<number>?* occurs in <nstring>, it is replaced by the <number>th partially specified string found by <ostring>. Thus the above problem can be solved by the command:

```
        S?:FOO?:$?*1?*BAZ?*2?*$<range>
```

To insert a ?* or an ^^, preceed it by an ^^. If a ?* is not followed by a number followed by a ?*, or if an ^^ is not followed by another character, the error *ILLEGAL REPLACEMENT STRING* will be printed. This same message will be given if a partially specified string which does not exist is specified as ?*8?* when there are only 2 partially specified strings.

BREAK-OUT--<control B>

There eventually comes a time when you make a mistake in the range given for a substitution or justification, and you want to recover from it immediately with as little actually changed as possible. Or you have started a long Find or Print command, but you decide to abort and want the current line/page pointer to be close to where you started (so the file doesn't have to recopy itself).

Rather than panic and pound ^C madly, you may simply hit ^B. ^B is always ready and able to stop cleanly any command which (explicitly or implicitly) involves a range specificier. ^B will tell you at what point SOS gracefully screeched to a halt, whereas ^O lets a command run to completion (though hiding the output: a binary switch).

If you ^B a substitution, all changes to that point have been made in the text buffer, but only to that point. If you reacted too late, you may need to go back to the source file (do a FILSTAT to see what file you're reading from). To help as much as possible, ^B does not turn off Teletype output...so that you can see the extent of the substitutions made, or whatever. You may do a ^O concurrently.

If you are in the process of copying from a second file, the breakout is deferred until things are stable again.

6. LINE NUMBERS REVISITED

It is possible to address lines by content rather than by number. This is done by replacing a <line-number> field of any command by

```
  <altmode><string><altmode><carriage-return>
```

The rest of the command string is continued on the next line. Thus, to list everything between a line containing "FOO" and a line containing "BAZ", suppressing line numbers, use the command

```
  L$FOO$
  :$BAZ$
  ,S
```

where $ denotes an altmode. Note that if the ",S" had not been included, two carriage returns would be required after the ":$BAZ$"; one to terminate the search, and one to terminate the L command.

If the string between altmodes is null, then the string last used in this context will be used again. Note that this is not the same as the string used in the F command.

The search will be from "." to the end of the file, unless otherwise specified. It is possible to otherwise specify: the full construction looks much like an F command, with altmode replacing the F:

```
  <altmode>(<string>)<altmode>(<range>)(,D|,N)(,E)(,<number>)<return>
```

The <number> (call it N) specifies that the Nth occurrence of the string is to be used as the designated line. The E option, as in the F command, specifies that upper- and lower-case letters are not the same things.

The D or N options will, upon finding a line, cause the line or its number respectively to be printed. If the next character typed is a rubout or backspace, another line will be looked for. Anything else will cause the line just found to be used as the line you were looking for. this process will be repeated at most N times. You cannot reject the last line (the Nth line), and so N should be other than 1. (99999 will suffice in most cases, and is the largest number SOS can understand on input.)

Also note that the range used in this construction can be another search-type construction. This sort of thing can be nested to a depth of 3. Anyone who thinks he needs more depth is invited to consult a psychiatrist.

APPENDIX A

CONVERSION

A few words about copying files and converting formats.

1. To copy files from disk to disk, disk to dectape, or dectape to disk, use the system COPY command.

2. To convert SOS files to TECO files, use 'E,S' or <SUBSYS>PIPPY. Either method removes line-numbers and null padding from the file, but leaves formfeeds in. E,S will delete the intermediate SOS copy of the file, iff the strip is successfully completed; otherwise, it remains the current file.

3. To convert TECO files or other files without line numbers to SOS files, simply read them with SOS. SOS will number them by 100 and insert a page mark if there are more then 999 lines on a page. Any form feed which is the first character on a line will be converted into a page mark. "Bare" <return>'s are deleted and "bare" <line feed>'s are changed to <return> <line feed>. An Ascii 37 in the file is presumed to be a TENEX end-of-line signal, and is converted without comment into <return><linefeed>.

4. For those users sending files to UCLA's 360-91, a program here is available which converts SOS files to respectively-numbered card-image files. The program is <CCN-KAY>A2CWP.SAV "Ascii-to-Card with Pagemarks", and converts SOS numbering as follows:

```
        Columns 73-75 == the SOS page, and
                76-80 == the SOS line.
```

If the text of the source SOS-line is > 72 characters in length, the line is split at char 72 onto a second line, with card-columns 73-80 having asterisks.

There is another program, <CCN-KAY>UPDCOM "update-compare", which is a modified srccom. It takes two SOS files and generates a new file with just the updates that were made, plus embedded insert-delete-replace-number commands ... this file is then sent to UCLA rather than sending the complete source each time. A program at UCLA takes the commands and generates an updated source there, as shown in <HEARN>JCL.

APPENDIX B

SUMMARY OF ERROR MESSAGES

Fatal:

DDE	DEVICE OUTPUT ERROR
DIE	DEVICE INPUT ERROR
DNA	DISK NOT AVAILABLE
FNF	FILE NOT FOUND
ICN	INTERNAL CONFUSION
TNX	TENEX CONFUSION
ILUUO	ILLEGAL UUO
NEC	INSUFFICIENT CORE AVAILABLE

Non-fatal:

ILC	ILLEGAL COMMAND
ILFMT	ILLEGAL LINE FORMAT
ILR	ILLEGAL REPLACEMENT
IRS	ILLEGAL REPLACEMENT STRING
ISS	ILLEGAL SEARCH STRING
ITD	ILLEGAL TRANSFER DESTINATION
LTL	LINE TOO LONG
NLN	NO SUCH LINE(S)
NNN	NO NEXT LINE
NSG	NO STRING GIVEN
NSP	NO SUCH PAGE
ORDER	OUT OF ORDER
STC	STRING TOO COMPLEX
STL	STRING TOO LONG
UNA	DEVICE NOT AVAILABLE
TMS	TOO MANY STRINGS
WAR	WRAP AROUND
BKO	BREAK OUT
XRO	NOT DURING READ-ONLY

APPENDIX C

SUMMARY OF COMMANDS

The following is a brief summary of SOS commands. Those arguments enclosed in () may be omitted. Where several arguments appear, seperated by |, it means that any one of these (but only one) may be used. [] are used for grouping.

```
Alter    A<range>
            <space>  next character
           <rubout>  last character -- also ^A
                  C   Change
                  D   Delete
                  I   Insert
           <return>  end alter mode
                  Q   Quit             -- also ^X
       <control U>  start over
                  S   Skip
                  K   Kill
                  R   Replace
                  L   Line
                  P   Print            -- also ^R
                  J   Justify

Beginning
         B

Break-out
         <control B>

Comment
         ;

Copy     C<dest>(_<file>(,S)),<source>(,<incl>)(,<inc2>)

Delete   D<range>

End      E (<file name>)(,S)

Find     F((<string>)<altmode>(<range>)(,A|,N)(,E)(,<number>))

Go       G (<file name>)

Help     H
Insert   I<line>(,<increment>)

Join     J<line>

Justify Center
         JC<range>
```

```
Justify Left
        JL<range>


Justify Right
        JR<range>

JUstify
        JU<range>

List    L(<range>)(,S)

Mark    M<line>

Number  N<increment>,<range>(,<starting number>)

display extend
        Q<range>

Print   P(<range>)(,S)

Replace R<range>(,<increment>)

Substitute
        S((<ostrng><altmd><nstrng>)<altmd>(<rng>)(,D|,N)(,E)(,<number>))

Transfer
        T<dest>,<source>(,<incl>)(,<inc2>)

save World
        W(<file name>)

eXtend  X<range>

display alter
        Z<range>

give information
        = =|_|FILE
        = .|BIG|CASE|INC|ERROR|STRING|PMAR|LMAR|RMAR|MAXLN|SAVE|ISAVE

set
        _UPPER|LOWER
        _DPY|M33|M37
        _C128|C64
        _NOVICE|EXPERT
        _SHOW|SUPPRESS
        _[PMAR|LMAR|RMAR|MAXLN]=<number>
        _INC=<number>
        _[SAVE|ISAVE]=<number>
```

APPENDIX D
STANDARD TEXT FORMAT

The format of a standard text file is defined to be as follows:

A line number is a string of 5 ascii digits, left justified in a single word, and having bit 35 a 1. A line consists of a line number followed by the text of the line in ascii characters, left justified in words with bit 35 a Ø. The first character of the text is a tab and the last two characters are <return> and <line feed>. The characters null (Ø) and delete (177) may not appear in the text. The last word of the line is filled with nulls to make a complete word if necessary.

A page mark is a word containing 5 ascii spaces (4Ø), left justified in a word with bit 35 a 1, followed by a word containing the characters <return>, <return>, <form feed>, <null>, <null> and with bit 35 a Ø.

Lines are placed into records starting with the first word. A line is never broken across 1Ø/5Ø block boundaries. No Ø words appear in a record except after the last line of that record. All unused words in a record are Ø.

SPELL

SPELL is a program that checks text files for correctness of word spelling. In addition to the spelling check, SPELL provides a facility for correcting words that it thinks are misspelled. The program was originally written by Ralph E. Gorin at Stanford University. It has been adapted to TENEX and TOPS-20 and considerably augmented at BBN.

SPELL reads through the input text file, checking each word against its dictionary. All words that it finds in the dictionary or that it can find after stripping off common prefixes and suffixes are copied through to an output file. When it comes upon a word it can't recognize, it tries to guess a correction and then asks you how to treat this word. You can have SPELL accept the word as correct, add the word to its dictionary, or correct the word in several ways. In this way, the output file becomes a corrected version of the input file.

<u>NOTE</u>: SPELL has no knowledge of syntax, so it will readily accept things like "JOHn red the book", even though "red" is incorrect usage. SPELL can only check each word against its dictionary to see if it corresponds to any correct spelling. Note also that SPELL takes no special note of "JOHn", since it does not consider case to be significant. You shouldn't come to believe that SPELL will catch all your typographical and spelling errors; it <u>will</u> detect many errors that a human reader could easily pass over, but it is no substitute for proofreading!

SPELL's internal dictionary contains over 40,000 words (including RUNOFF and MRUNOFF commands). Because SPELL's built-in dictionary cannot possibly contain all the words and proper nouns in an individual user's vocabulary, SPELL contains provision for the user to maintain a private dictionary file (or files) to augment SPELL's built-in vocabulary. For all but casual users <u>this practice is highly recommended</u>.

NORMAL USE OF SPELL

SPELL's commands and switches are all single letters. A few commands take an optional dictionary number following the letter. All typeins to SPELL must be terminated by carriage return. The editing of typed input can be done in the usual way, using the conventions of either TENEX or TOPS-20.

The editing characters are:

Char-delete	^A, Rubout, or ^H
Word-delete	^W
Line-delete	^Q, ^U, or ^X
Retype line	^R
Quote character	^V

(1) When SPELL is started, the first thing it does is to see if you have a default private dictionary file; if there is a file in your login-directory named DICT.SPELL, this is assumed to be a dictionary file and loaded as an "incremental" addition to SPELL's built-in dictionary. (When you subsequently exit from SPELL with the "E" command (see paragraph (5) below), SPELL will automatically update this file if necessary.) Proceed to paragraph (2) below.

If no DICT.SPELL file was found, SPELL will ask if you want to load a private dictionary file by asking:

    Do you want to augment the dictionary?

If you do not want to load a dictionary file (or have none to load), type "N" (or just <cr>) and skip to paragraph (2) below. (If you forget to load a private dictionary at this point, you'll have other chances later on.) If you answer "Y", SPELL will type:

    Dictionary file name:

After you type in the name of the dictionary file, SPELL will type:

    Type "I" to mark these as incremental insertions:

The usual practice at this step is to type the "I", for that means that the new words will be marked as "incremental" additions, so that they would be included in an incremental copy of the dictionary at the end of the session (if you make one - see (5) below); not typing "I" means that the new words become merged with SPELL's internal dictionary.

After the dictionary file is loaded, SPELL will again ask if you want to augment the dictionary, thus permitting more than one dictionary file to be loaded. Eventually you'll give a negative answer and thus go on.

(2) SPELL will next ask if you want to set any optional mode switches:

```
     Mode switches (zero or more of P,U,M,N,A,T,Q, or ?):
```

The usual response is just a carriage return to specify no switches. The other options are:

P Pickup mode. You will be asked to specify a page and line number at which to "pick up" the spelling checking. When you have a partially corrected file, this mode enables you to skip over the initial portion that has already been corrected. The input file is copied to the output file without checking until the page and line specified, at which point spelling checking begins.

U Upper case mode. In this mode, each new word that is entirely upper case will not cause SPELL to ask you about it, but will be inserted in a special dictionary. You will be asked to specify a dictionary number for the upper case words (see HOW TO USE MULTIPLE DICTIONARIES below). This mode can be useful if your file contains many special "words" that are written in upper case, such as logic symbols, opcodes, or program variable names, and you don't want SPELL to query you about each and every one. Note, however, that SPELL does NOT check to insure that subsequent occurrences of these words are also entirely upper case.

M Misspellings. Put misspellings (and their corrections) in a special dictionary. You will be asked to specify the dictionary number. Normally, when you correct a misspelling found by SPELL, the misspelling-correction pair is put in SPELL's main dictionary so that subsequent occurrences can be automatically corrected, but these misspelling-corrections can't be saved for future use. This mode lets you specify a dictionary for these misspelling-corrections; this dictionary can be the default incremental dictionary (#1) or a special one. This mode can be useful if you misspell more frequently than you mistype, and you'd like to accumulate instances of the words you misspell for use in checking other text files.

N No suffix-stripping.

A No prefix-stripping. The suffix- and prefix-stripping algorithms are useful and clever, but far from foolproof. You can use the N and A switches if you prefer to use SPELL without them.

T Training mode. SPELL will treat the input file as a training set rather than a file to be corrected. See the discussion of TRAINING MODES below.

Q Q-training mode. Similar to T: see the discussion of TRAINING MODES below.

(3) You will then be asked to specify the names of the file to be checked, the corrected output file, and an exception file:

```
Name of the file to check and correct:
File name for corrected output:
File for exceptions:
```

If you specify the input file as INFILE.EXT, then typing <esc> to the "corrected output" prompt defaults the output file name to INFILE.EXT; similarly, typing <esc> to the "exceptions" prompt defaults the exception file name to INFILE.EXCEPTIONS. (See THE EXCEPTION FILE below for a description of that.) If you type <cr> to either of those prompts, then the respective file will not be created at all.

(4) After you have specified all the files, SPELL will respond with "Working..." and start checking the input file for spelling errors. While it is doing so, there are three kinds of occurrences that will cause SPELL to type out on your terminal: (a) unknown words that SPELL needs to ask you about, (b) known misspellings, and (c) words that SPELL has matched by means of affix-stripping rules. Only the first of these requires you to type anything in reply.

(4a) Each time SPELL encounters a word that it doesn't recognize, it will ask you about it by typing the page number, the line number, and the line in which the word occurs, followed by the word itself, and whether it has any guesses for correcting it. (Note that the page number typed by SPELL refers to the number of page-feeds in the input file; it generally won't correspond to the page number of the finished document.) If SPELL has one to three guesses, it will type them out also. For example:

```
Page 1:34
reads through the imput text file, checking each
IMPUT  I guess: (1) INPUT or (2) IMPUTE
*
```

The asterisk prompts you to tell SPELL what to do with the excepted word. At this point, you have a choice of several commands, some of which (S, C) are dependent on the number of guesses. (Typing "?" will get you a summary of the allowable commands.)

A Accept this word, this one time.

I Insert. Accept this word and insert it in the dictionary so that subsequent occurrences of this word will be recognized and accepted. Words that are inserted this way are marked as incremental insertions, and may be copied out to form an auxiliary dictionary file.

C(n) Correct this word with SPELL's (n-th) guess. If SPELL has volunteered just one guess, then type "C" to use it as the correction. If SPELL has shown 2 or 3 guesses, or if you have typed out many guesses with the "S" command, then type "Cn" to use SPELL's n-th guess as the correction.

S Show all of SPELL's guesses. If SPELL has more than 3 guesses for the unknown word, it won't type them out unless you request it with this command. After SPELL has done this, you may select one of the guesses with the "Cn" command described above.

R Replace this word. SPELL will ask you to type the replacement word. If the replacement word is not already in the dictionary, SPELL will give you an opportunity to insert it.

If the misspelling is due to an omitted space between words, use the "R" command to retype the words with the space.

L Load a private dictionary file to augment SPELL's built-in dictionary, then reconsider this word. You will be asked for the dictionary file name.

X Accept this word and finish. The word will be accepted. Then the remainder of the input file will be copied without checking to the output file. This is useful if you are only part way through a file and you wish to stop without losing the corrections already made. (The next time you use SPELL, you can use "Pickup" mode to resume checking this file at the same page and line number.)

If the X is followed by a number n, it means something slightly different. It means to suspend spelling checking for the next n lines (including the current one). This can be useful for skipping a portion of text containing nonword character strings.

W Save the incremental insertions. After you type "W", you will be asked for a file name. Then an incremental copy of the dictionary will be written into the file. After the copying is complete, you may decide what to do with the excepted word.

D Display the line and offending word again. The line that is displayed will not have any corrections shown in it.

(4b) SPELL knows about many common misspellings, and as you process the input file, it also remembers corrections you make to misspellings it finds. If SPELL encounters a word that it already knows is wrong, it automatically performs the correction, and it informs you of this by typing (for example):

```
Page 1:179
as seperate but equal branches of government.
seperate ==> separate
```

(4c) If SPELL cannot find a word in its dictionary, it applies various prefix- and suffix-stripping rules, to see if it can recognize the stem after these affixes have been removed. For each such word it finds, SPELL will:

- type it on your terminal, as for example:
```
UNSTEADINESS   = UN+STEADY-Y+I+NESS
```

- note it in the exception file, and

- enter it into the dictionary (specifically dictionary #31)

Since each affix-match is entered in the dictionary, if the same word is encountered again, it will be accepted with no further action. Thus such an affix-match type-out signifies only that the file contains <u>at least</u> one occurrence of it. Although the affix-stripping rules are quite effective, they are far from foolproof (e.g., CHOSES would be accepted as CHOSE+S). Therefore SPELL types out these affix-matches so that you can monitor their validity.

(5) When the input file is exhausted, all files are closed, SPELL types "Finished processing the input file." and enters an exit-command sequence:

```
Type E,I,N,C,A, or ?:
```

Again, you have several choices:

E Exit from SPELL to the EXEC. If SPELL had automatically loaded a DICT.SPELL dictionary file at startup and words were incrementally added to that dictionary, an updated DICT.SPELL file is automatically written.

I Incremental copy. Make an incremental copy of the dictionary. All words that you inserted while running SPELL (and loaded from private dictionary files) are copied to a file. SPELL asks for the file name. The words in this file will not normally be in alphabetical order, but rather sorted only on the first two letters and the length. (If you type IS (S for "sorted"), the file will be sorted into alphabetical order, but BEWARE! this consumes quite a lot of CPU time.)

N Numbers. Type out statistics about how many words were processed, how many spellings and misspellings are in the dictionaries, and the time used by SPELL.

C Correct. Go back and correct another file.

A Augment the dictionary (load another dictionary file), set new mode switches, and correct another file.

SPELL DICTIONARY FILES

Dictionary files for use with SPELL (and as copied out by SPELL) are ordinary text files, which may be edited and listed. They contain two types of entries: correct spellings and misspelling-correction pairs. The format is one dictionary entry per line. Each entry must be composed of alphabetic characters (apostrophes are permitted inside a word) less than 40 letters long. The entries need not be in alphabetic order. Upper and lower case letters are not distinguished.

Misspelling-correction pairs are put on a single line in the format "misspelling>correction" (e.g., ARGUEMENT>ARGUMENT).

When you load a private dictionary file, any words in the file that are already in SPELL's main dictionary are not duplicated. Hence, if your words are marked as incremental additions, then in a subsequent incremental dictionary copy, these duplicate words will not be copied out.

SPELL comes with a large built-in dictionary, but even a very large dictionary cannot encompass all the jargon words and proper nouns in an individual user's working vocabulary. SPELL lets you make incremental additions to the dictionary as you encounter new words, but it cannot remember them from one session to another. Therefore it is wise for you to maintain a private "incremental" dictionary file in which you save such words after each use and which you load into SPELL each time you use it.

SPELL's "default private dictionary" feature makes this particularly easy to do. When you start up SPELL, if there's a file in your login-directory named DICT.SPELL, it will be automatically loaded (into dictionary #1). When you exit SPELL via the "E" command, if you have added words to the incremental dictionary, then SPELL automatically writes an updated DICT.SPELL file before stopping. If you don't have a private dictionary file to start with, simply create an empty file named DICT.SPELL, or the first time you use SPELL, create a dictionary file named DICT.SPELL with the "I" command before exiting.

Of course, you can name your dictionary file (or files) anything else. In that case, you will have to load and update it yourself, using the commands described above in NORMAL USE OF SPELL.

HOW TO USE MULTIPLE DICTIONARIES

SPELL has a set of features whereby you can cause the creation of several disjoint incremental dictionaries. In this way, you may collect several dictionaries of special vocabulary classes.

The section above called NORMAL USE OF SPELL talked only of SPELL's "main" dictionary and "incremental" additions to it. In fact, SPELL contains 32 dictionaries, numbered Ø to 31. Dictionary Ø is the main dictionary; words can be added to this one only by reading in a private dictionary file at SPELL-startup time. Words that are inserted "incrementally" are marked as being in dictionary 1, unless you specify otherwise. Dictionary 31 is reserved for holding affix-matching words. You may specify a dictionary number at several places during the running of SPELL by appending a dictionary number to some commands; if no number is specified, the dictionary number will default to 1.

The following places are where you can specify which dictionary to add words to:

1. When loading a dictionary file at SPELL startup time, you are asked to "Type "I" to mark these as incremental insertions." Responding with "In" (where n is a decimal integer between 1 and 3Ø) means to add the words to dictionary number n.

2. While the input file is being checked, after SPELL has asked you about an unrecognized word, type "In" to insert that word in dictionary n. If, instead, you load a dictionary file with the "L" command, to have it go into dictionary n, type "In" to the prompt that says "Type "I" to mark ...".

3. After replacing a word (with the R command), if SPELL asks you whether you want to enter the word in dictionary, then type "In" to insert the replacement into dictionary n.

When requesting an incremental copy of a dictionary to a file, you may specify the particular dictionary to be copied (1-31). This is appropriate in two cases:

1. After some word has been asked about, the command "Wn" will cause dictionary number n to be copied.

2. During the exit sequence, the command "In" will cause dictionary number n to be copied. (If you want dictionary number n to be sorted alphabetically, type "InS".)

HINT: In the course of correcting a file, it is likely that you will be asked about words that you wish to have accepted during this file, but which you don't wish to have saved in your incremental dictionary(s). In these cases, simply insert them in a "throwaway" incremental dictionary (such as dictionary 9), which you don't bother to copy to a file when you're finished.

THE EXCEPTION FILE

In addition to the corrected output file, SPELL may produce an "exception file," in which are noted those places in the input file where SPELL encountered a word not found in its dictionary. Each word that SPELL asked you about and each automatic misspelling correction is noted along with the line in which it occurs. Also noted is (the first occurrence of) each word SPELL recognized by dint of stripping off prefixes and suffixes.

The exception file also has a special use in Q-training mode, as described below.

TRAINING MODES

SPELL has two "training" modes, in which it treats the input file as a training set rather than a file to be corrected. All words in the file that are unfamiliar to SPELL are entered in the dictionary as incremental insertions. Afterwards, you may do an incremental copy of the dictionary to a file, examine and edit it to remove misspellings and other inappropriate entries, and subsequently use it as an auxiliary dictionary file. This feature is provided for the purpose of creating specialized dictionaries of jargon or technical words from existing text files.

The training modes are selected by typing "Q" or "T" in response to the "Mode switches" request when SPELL is started. In both cases, you are asked to specify a dictionary number in which the unfamiliar words are entered; if none is typed, 1 is assumed. There is no output file.

T Training mode. Operates as explained above.

Q Q-Training mode. Identical to T, but with this additional feature. If any of the unfamiliar words is "close to" a word that SPELL does know, it is also output in the exception file. In this way, SPELL calls to your attention the fact that these words may be misspellings. The exception file contains only such words.

MISCELLANEOUS FEATURES OF SPELL

A text file may contain portions (such as quoted dialect or program excerpts) on which it is undesirable that SPELL perform its checking and correcting. You can tell SPELL to turn off its spelling checking by including in the file a line containing only ".<any-char>NOSPELL". Spelling checking is turned on again by a line containing ".<any-char>SPELL". (Case is not significant in NOSPELL/SPELL.) Most text formatters permit comment lines of the form ".*Comment" or ".;Comment", so these SPELL-control lines will be transparent to them. For example, in a MRUNOFF file:

```
This part of the text will be checked by SPELL.
.*nospell
Any "misteaks" in this part won't be seen by SPELL because
of the .*nospell comment line above.
.*spell
```

SPELL will read either SOS or TECO format files for the file to be corrected; the output file will be written in the same mode, with the same SOS line numbers, if present, that the original file had. Dictionary files may be either SOS or TECO format.

When a word is corrected, the output file will be rewritten with either upper case, lower case, or mixed (first letter upper, the remainder in lower), depending on the cases of the first two letters in the original word. Note that this will be incorrect in a small number of cases (such as McCarthy); SPELL will type a warning in these cases.

When SPELL updates the default dictionary file (DICT.SPELL), it sets the file retention count to 1. On TOPS-20, this has the effect of deleting all but the current version of that file. (On TENEX, this has no effect.)

Before SPELL asks you about a word it doesn't recognize, it tries several spelling correction heuristics in an attempt to "guess" the correct word, on the assumption that the excepted word is a misspelling and not a novel word. For short words, this will usually result in a large number of guesses, but for longer words, this will frequently result in only one or two guesses. The kinds of errors checked for cover a wide variety of sins of misspelling and typographical error:

1. one wrong letter
2. one missing letter
3. one extra letter
4. two transposed letters
5. an omitted space between words (sometimes)

USE OF SPELL UNDER HERMES

HERMES, BBN's electronic mail handling system, permits SPELL to be used to process text fields. When SPELL is invoked from HERMES, excepted words are handled as explained in paragraph (4) above in NORMAL USE OF SPELL, but the preliminary and final procedures (paragraphs (1-3, 5)) are virtually eliminated.

1. The "default private dictionary" works as described above. You are not specifically asked if you want to augment the dictionary, but the "L" command at excepted-word time gives you an opportunity to do this if desired.

2. No mode switches may be set.

3. When SPELL is finished, the default dictionary file is updated. If no default dictionary file exists, but words have been added to the incremental dictionary (#1), you are asked if you want to save it (create a dictionary file). There is no opportunity to save any other dictionary.

## SRCCOM

A Source Compare Program

The program SRCCOM on TENEX is an adaptation of the verify option of UTILITY on the XDS 940 with several improvements and additional features. The name is the same as DEC´s source compare program, but for our purposes SRCCOM has been TENEXized and is now 50% faster.

SRCCOM provides a facility for comparing two symbolic files and discovering the differences between them. Two forms of output are generated by SRCCOM: a marked listing and differences. The marked listing is a listing of the first (usually newer) file with 2 asterisks in the right margin after each line which is different from that of the second (usually older) file. An option exists to list only pages which have been changed. Changed pages include those which are renumbered due to inserting or deleting a page. The first page is also always listed. Marked listings with giant pages are not possible.

# SRCCOM provides a facility for automatically maintaining a set of
# listings. A file named "RECORDOFLISTINGS" contains a record of
# the last listed versions of a set of files. SRCCOM uses this
# record to determine which files have newer versions and generates
# a listing of changed pages and differences for the new version
# against the old. When the listing of all such changes is
# completed, the record is updated to reflect the newly listed
# files. The file "RECORDOFLISTINGS" can be initialized by using
# the EXEC´S directory command and putting the output in a file
# (the first two lines of heading information must be deleted with
# TECO).
#
# When SRCCOM starts, if the file "RECORDOFLISTINGS.;1" is present
# in the connected directory, SRCCOM will ask the question
# "SPECIAL?". A reply of "Y" will cause the file
# "RECORDOFLISTINGS" to be ignored. A reply of "N" will cause the
# record to be used. A reply of "F" will perform the same
# functions as "N" except that any errors in the "RECORDOFLISTINGS"
# file will be paused for giving the user a chance to correct the
# error. (This will occur when SRCCOM looks for an explicit
# version of a file and doesn´t find it). The rest of the
# initialization dialogue is self explanatory.

TAINT

This writeup describes a very early version of a program called TAINT. TAINT´s function is to read files produced by the TENEX MINI-DUMPER program, and to either produce a directory of the tape or to write the files on a DECsystem1Ø disk system (or both).

The raison d´etre of TAINT is to allow installations running a DECsystem1Ø monitor to read TENEX dump tapes from co-operating TENEX sites or in preparation for installing a TENEX system of their own. Since the testing of this program has been done under TENEX´s 1Ø/5Ø compatibility system, reports of problems under a real 1Ø/5Ø system are not impossible and will be appreciated.

TENEX directory names are alphanumeric strings. Therefore, a set of translations to DECsystem1Ø project - programmer numbers is required, and TAINT sets this up before reading any files (see below). In addition, TENEX file names and extensions are character strings of up to 39 characters, and an 18-bit version number is a part of the file name (not an attribute). When these files are copied to the DECsystem1Ø disk, the file name must be six or fewer characters, and the extension three or fewer. If this results in truncation of a name, a comment will be typed mentioning the (full) file name. The DECsystem1Ø version number is not a part of the file name and it is (at present) lost. In particular, if more than one version of the file exists on the TENEX tape, then the one appearing first on the tape will be written and (unless the "SAVE EXISTING FILES" option is used (q.v.)), a version appearing later on the tape will over-write it. Since MINI-DUMPER tapes normally have the highest version first, this would be undesirable, since the lower version number (older) file would remain on the disk.

Program Operation:

When TAINT is started, it carries on the following dialogue with the user.

1. TAINT asks: "LIST TAPE´S DIRECTORY? (Y OR N)" and the user replies with Y or N. If Y, TAINT asks for a file name to write the directory on (defaults being DSK: TENEX.LST). Examples are "LPT:" or "TAPE.DIR".

2. TAINT asks: "READ ANY FILES?"
The user replies Y or N. If Y, TAINT asks for the TENEX DIRECTORY NAME to PROJECT, PROGRAMMER NUMBER correspondence. The simplest response is "carriage return" abbreviated in this memo as "CR". This response implies "Read all TENEX directories on the tape into my directory (the one I am logged in as)". The next simplest response is, for example,

```
SMITH"CR"
JONES"CR"
"CR"
```

which means to load all of the files from SMITH and JONES into "my" directory.

Another possibility is:

```
SMITH,10,7"CR"
"CR"
```

which loads SMITH´s files into [10,7] (assuming, of course, that you have write access to the [10,7] directory). The full case is:

```
SMITH"CR"
JONES,10,7"CR"
D-DUCK"CR"
M-MOUSE,10,6"CR"
GOOFY"CR"
"CR"
```

which loads SMITH into "my" directory, JONES and D-DUCK into [10,7] and M-MOUSE and GOOFY into [10,6].

In all of the above, note that the end of the input is determined by typing a blank line.

3. TAINT next asks "SAVE EXISTING DUPLICATE FILES?"
A Y answer implies that no existing files with identical file names (after any truncation) will be overwritten. Therefore, the first (usually newest) file of a given name and extension on the tape will not be written over by a subsequent one with a different version number. Unfortunately, an already existing file on the disk (from an earlier magtape, for example) will not be overwritten either.

An N answer implies that all files will be overwritten, possibly resulting in an older version being the last one written as described above.

To get the latest version from tape, write only into empty (but existing) directories, and answer Y to this question.

Files do not (yet) get their correct creation date from the tape. They just get the date that the tape is written onto the disk, i.e., the current time and date.

4. TAINT asks "TAPE DRIVE NUMBER". The answer to this is a digit from Ø thru 7, for MTAØ: thru MTA7:. (Don´t type this digit until the tape is mounted and ready.)

5. TAINT then processes the tape, mentioning on the user´s TTY the tape identification, each user name for which files exist (and where they are copied to if they are copied), truncated file names (if any), and any errors encountered. All output files (at present) are a multiple of 1ØØØ words octal in length (a TENEX page).

TAPCNV

TAPCNV is a subsystem which reads a card image file previously processed by MTACPY. It then does the appropriate conversion to ASCII from BCD (Ø26 keypunch code), or eight-bit EBCDIC, or six-bit EBCDIC (Ø29 keypunch code) or from BCL. (Burroughs' external code).

When a file is copied from magtape to disc using MTACPY, an additional auxiliary file is created with extension .RECSIZ which contains the size of each record. TAPCNV uses this auxiliary file to correctly "parse" the data image file into card images for code conversion to ASCII.

On startup, TAPCNV asks:

```
     TAPE IMAGE FILE =
```

and the correct response is a file previously moved from magtape to disc by MTACPY. Then the program attempts to open the corresponding .RECSIZ file. If it fails to find this file, it asks for it:

```
     RECORD SIZE FILE =
```

If this file does not exist, the file should be re-acquired from magtape using MTACPY to create this auxiliary file.

TAPCNV then asks,

```
     OUTPUT FILE =
```

and the correct response is any writeable, ASCII file. If just an altmode is typed for name recognition, the program creates a default file with the same device, directory, and name as the input file, but the extension is .ASCII.

Next, the program asks,

```
     BCL or BCD or EBCDIC?:
```

conversion begins immediately if the response is BCL or EBCDIC. If the response is BCD, the program asks:

```
     Ø26 or Ø29 CODE?:
```

and conversion begins once the user responds.

Trailing blanks are suppressed during conversion, a carriage-return, line feed is supplied for each card image, and each new record is assumed to begin a new card image.

```
@TAPCNV
TAPE IMAGE FILE = FILE1
OUTPUT FILE = <altmode>FILE1.ASCII [NEW FILE]
BCL OR BCD OR EBCDIC?:BCD
Ø26 OR Ø29 CODE?: Ø26
DONE.
@
```

TENEX TECO

This tutorial is designed for a single fast reading. Its purpose is to convey the details of TECO to a reader who is acquainted with the TENEX System and who has some idea of what to expect from a text editor.

The first section describes a subset of TECO commands which are sufficient to do any ordinary editing job. The second section describes commands which are essential for editing large files, and includes some of the most useful commands in TECO. The final section gives brief descriptions of the commands of TECO not already described, and introduces the reader to groups of specialized commands for which he may later have a need.

For more information see the TENEX TECO Manual.

1. BASIC EDITING

The text which TECO edits is a pure character string, without restrictions or exceptions. A sequence of characters of almost unlimited length can be accommodated, and any one of the 128 ASCII characters can appear at any position in this sequence. Even printer-control operations such as carriage return or tab are represented by ASCII character codes, and they require neither a special form of data to store them nor special commands to manipulate them.

The Character Set

The complete ASCII Character set is described in Appendix B of the TECO Manual. The characters can be divided into three groups, as follows:

- -- Each of the graphic characters (95 of them) prints a single letter, digit, or punctuation mark and then advances the printing element one position. The Space character is included in this group.

- -- Each of the format control characters (7 of them) causes the output device to take a special action, such as starting a new line, spacing to a tab stop, or ringing the bell.

- -- Each of the command control characters (26 of them) can be used by TENEX software for a one-character, special-purpose command.

```
# Names for Control Characters
#
# Since a control character does not  type  a  particular  graphic,
# such  as "a" or "+", it must be given an artificial name.  In the
# text of the TECO Manual, a name is used which indicates the  mode
# of  production  of the character on a terminal.  For example, the
# character produced by holding down the CTRL key and striking  the
# "D"  key  is  called  Control-D.   In the listings of interactive
# dialog between the user and TECO, a shorter name is  used  for  a
# control  character, and this name is set off from the rest of the
# dialog by an enclosing pair of parentheses.  For example,  "(^D)"
# is used for Control-D.
#
#      The following table  gives  the  short  names  of  the  most
# important  control  characters  and suggests the way in which the
# characters are used in TECO:
#
#      (%)       A Carriage Return was transmitted at this
#                point  in the dialog.  It was used to end
#                a line.
#
#      (HT)      A (Horizontal) Tab was  transmitted.   It
#                was  produced by Control-I and caused the
#                terminal to space to the next tab stop.
#
#      (^C)      A   Control-C   was   transmitted.     It
#                interrupted TECO.
#
#      (^D)      A   Control-D   was   transmitted.     It
#                terminated  the character string argument
#                of a TECO command.
#
#      (ESC)     An  Escape  was  transmitted.    It   was
#                produced by a key with "ESC" or "ALTMODE"
#                or "PREFIX" written on it and caused TECO
#                to  execute  a string of previously typed
#                commands.
#
#      (DEL)     A  Delete  was   transmitted.    It   was
#                produced  by a key with "DEL" or "RUBOUT"
#                written on it and caused  TECO  to  abort
#                the command it was executing.
#
# The functional descriptions just given are introductory sketches.
# Subsequent  sections will describe the use of these characters in
# detail.
#
```

Integers

The main activity of TECO is to manipulate character string data, but TECO can also operate on integers. The commands make use of integers for three special purposes: to count off the characters or lines in a string, to specify the number of times a command loop is to be executed, and to represent an ASCII character code.

The operators of elementary arithmetic, "+", "-", "*", and "/", are available for making up expressions. The division operator forms the quotient and then drops the fractional part without rounding. It is rare that the TECO user needs an expression with more than one operator in it; if such a case arises, however, the user should use parentheses to indicate the order of evaluation. For example, write "3+(4*5)", not "3+4*5.

RUNNING TECO

The purpose of TECO is to enable a user to enter a program or a document into the TENEX System. The essential steps of this process are (1) start up TECO, (2) type in the text, and (3) file the result; indeed, if a user could avoid mistakes and changes, this much would suffice.

A Complete Session with TECO

The dialog which follows is a sketch of the overall control of TECO by the user:

```
@TECO(%)         The user (who is at the TENEX
(%)              EXECutive Level) types "TECO" and a
                 Carriage Return.
                 TENEX loads and starts TECO.
*xxx(ESC)$(%)    TECO types "*" (go ahead sign);
                 the user types one or more TECO
                 commands, xxx;
                 then types Escape (go ahead, TECO);
                 and
                 TECO echoes with "$" and a Carriage
                 Return and then executes the command
                 string.
*xxx(ESC)$(%)    The cycle is repeated until the user
                 has completed his work and filed the
                 results.
...              ...
*;H(ESC)$(%)     Finally, the user gives the Halt
                 Command (;H) to exit from TECO, and
@ ...            TECO returns the user to the
                 EXECutive Level.
```

In the dialog just given, xxx represents a command string.

# Several commands can be typed in as a single command string, and
# no separation is required between the commands. When the user
# has finished typing a string of one or more commands, he types
# the Escape character. Only then does the execution of the
# commands begin, proceeding from left to right.
#
The examples in this document make use of both upper and
lower case letters (although some terminals are limited to upper
case). TECO pays attention to the case of a letter only when the
letter is part of the text being edited. When a lower case
letter is typed as part of a command name TECO treats it as if it
were upper case. Thus the Halt Command illustrated above can be
typed in as either ";H" or ";h".
#
An Error
#
If TECO encounters a command which it cannot execute, it responds
as in the following dialog:

```
#       *8C;ADD(ESC)$(%)      A string of four commands has
#                             been typed. TECO executes the
#                             first command, 8C, but rejects
#                             the next command ";A".
#       ?42(%)                TECO types an error number
#       8C;A(%)               and types the command string
#                             through the illegal command
#                             (thus pointing out the error).
#                             TECO then discards the
#                             subsequent commands, "DD".
#                             Finally,
#       *                     TECO calls for a new and better
#                             command string.
```

The Error Messages Appendix of the TECO Manual lists the error
numbers used by TECO to report errors. In that appendix, the
interpretation of "42" is "An undefined command character has
been given". Therefore, the reason ";A" was rejected in the
dialog above is that there is no such command in TECO.
#
The Illustrations
#
This writeup uses both isolated examples and dialogs to
illustrate TECO. An isolated example shows the command string
which a user types but does not show the response from the
computer system.
#
A dialog is a longer and more complete way to illustrate
TECO; it shows a sequence of interactions between the user and
the computer system. In the dialogs the user type-in is
underlined and each use of a control character is explicitly
shown. These conventions make clear who has done what.

During an actual session with TECO, the underlines and the control-character names are not typed out. For the dialog given above (TECO´s handling of an error), the actual listing would therefore be as follows:

```
*8C;ADD$
?42
8C;A
*
```

TYPING IN THE TEXT

The main storage of TECO is the buffer, and it holds the character string which is being edited. The buffer is initially empty, but it can expand to a capacity of over a million characters as the user enters his text. Since a typed page typically contains 2000 characters, the buffer can accommodate about 500 pages of text. Thus it is almost always the case that the entire program or document being edited fits in the buffer and can be manipulated as a single character string.

There is a pointer associated with the buffer. It specifies the position in the buffer at which the main editing activity is going on, and it is moved by commands which are discussed in a later section. Many TECO commands, including those described in this section, operate relative to this pointer.

Typing in the Text

The Insert-String Command is the command normally used to type text into the buffer. The command is entered by typing an "I", followed by a character string, s, of any length, followed by a Control-D or ESC. The command inserts the character string, s, into the buffer just before the pointer. The inserted string may contain any letter, digit, punctuation mark (including space), or formatting character. The only characters excluded are certain control characters which rarely appear in text.

Two examples of the Insert-String Command follow:

```
     Ia(^D)            Insert the letter  "a"  just  before
                       the pointer.

     IThis is(%)       Inserts two lines
     a test.(%)        just before the
     (^D)              pointer.
```

A full dialog will now be given to illustrate the Insert-String Command. In this dialog the user enters the upper-case alphabet into the buffer, five characters per line.

```
     *IABCDE(%)        The user starts typing
     FGHIJ(%)          the alphabet...
     KLM(ESC)$(%)      Exhausted, and fearful of losing
     (%)               his work, he terminates the command.
                       TECO enters 14 characters (including
                       2 End-of-Line characters) into the
                       buffer.

     *INO(%)           After a pause, the user
     PQRST(%)          completes the typing of
     UVWXY(%)          the alphabet.
     Z(%)              Note that input resumed
     (ESC)$(%)         just where it left off, in
     (%)               the midst of the line which was left
                       unfinished in the first
                       Insert-String Command.
```

The purpose of the (^D) at the end of a character string argument is to end the argument. When the argument is already at the end of a command string, the (^D) is not necessary. This useful exception is applied to both Insert-String Commands in the dialog just given, where (ESC) is used to end both the command string and the character string.

A Dangerous Error

Suppose the user, intent on the job of getting some text into the buffer, starts typing the text without preceding it with an "I". When the user types Escape, the text is not entered into the buffer; instead, TECO tries to interpret it as a command string. If the user is unlucky, TECO will be able to proceed through several "commands" before being stopped by an illegal command.

When this error occurs, the user needs to determine whether the text already in the buffer has been affected by the execution of the false commands. He can check this in one of three ways:

-- He can trace, command by command, the action of TECO in its runaway interpretation of the text. Usually this is easy; occasionally it is very difficult.

-- He can type out and read the entire contents of the buffer.

-- He can start over; that is, he can go back to the previous version of the file being edited.

Usually, TECO stops before the buffer has been modified; but some damage to the user's text is always a possibility when a random sequence of commands is executed.

Recovering Lost Type-in

TECO has a backup register in which it saves the most recent command string which was over 15 characters long. A command string is saved just before execution begins, so the whole string is saved even if it contains an illegal command. This backup register is of great interest to the user who has just typed a long insertion without supplying the initial "I", as described in the previous paragraphs.

The ;Get-Commands Command (;G) makes a copy of the command string in the backup register and inserts the copy into the buffer just before the pointer. The following dialog illustrates the use of this command:

```
*Daffodils ...          The user is typing a
...                     novel as a single
... snow fell.(%)       Insert-String Command...
(ESC)$(%)               But he leaves the "I" off!
?32(%)                  TECO tries to interpret the
D(%)                    text as a command string and
                        (fortunately) fails before
                        executing a single command.

*;G(ESC)$(%)            The user recovers with a
(%)                     ;Get-Commands Command, and the
                        novel is copied from the backup
                        register into the buffer.
```

The Carriage Return

The type-in of a Carriage Return, indicated by "(%)" in the dialog, requires a special explanation. Strictly speaking, it is the function of the Carriage Return key only to move the printing element back to the left margin. A second key, Line Feed, is provided to advance the paper TENEX intervenes as follows:

-- A Carriage Return character (decimal code 13) coming in from the user's terminal is echoed as a Carriage Return followed by a Line Feed and is then entered into storage as an End-of-Line character (decimal code 31).

-- An End-of-Line character going out to the user's terminal is converted into a Carriage Return followed by a Line Feed.

This arrangement has substantial advantages. When the user wants to end a line, he needs only a single keystroke (Carriage Return). Further, the separation between lines in a stored character string is represented by a single character used only for that purpose (End-of-Line). Of course, the user must use a special technique to enter a true Carriage Return character into storage, but the need for this character is rare.

Typing Tricky Text

The list of allowed characters given in the description of the Insert-String Command excluded some ASCII characters. The fine points of this matter are discussed in the Character Set Appendix. For the present, it is sufficient to introduce a command which can be used to insert any ASCII character into the buffer. The Insert-Code Command (nI) inserts a single character into the buffer just before the pointer. It inserts the character whose decimal code is given by the integer value, n, which precedes "I" in the command. This command offers full generality but is not, of course, as convenient as the Insert-String Command.

In the following dialog, the user wants to type out a line of slashed zeroes by using (1) a line of "O" characters, (2) a true Carriage Return (without a Line Feed), and (3) a line of slashes. He must use the Insert-Code Command to get a Carriage Return (decimal code 13) into the buffer since a typed Carriage Return would be transformed by TENEX into an End-of-Line character.

```
*IOOO(^D)$13II///(ESC)$(%)
(%)
```

FILING THE TEXT

TECO communicates with the TENEX file system on behalf of the user; that is, certain TECO commands are used to transmit text between a TENEX file and the editing buffer of TECO. Since the editing buffer has such a large capacity, the user can almost always read his entire file into the buffer rather than process it piece by piece. Under these conditions, input/output is

# simple and only three commands are required, as follows:
#
-- The ;Yank-File Command (;Y) is the input command.
It places a copy of the designated file after
whatever is already in the editing buffer and
leaves the file unchanged.
#
-- The ;Unget-File Command (;U) is the output command.
It replaces any previous contents of the designated
file with a copy of the entire contents of the
buffer and clears the buffer.
#
-- The ;Save-File Command (;S) saves the entire buffer
on a new version of the file. The buffer is not
cleared.
#
File Designators
#
Each of these commands requires a file designator in order to
continue. A full description of the way in which TENEX files are
designated is outside the scope of an introduction to TECO. This
discussion will assume the simple case that the desired file is
on the main system storage device and is in the user's own
directory. In that case, a file is uniquely designated by
#
-- the name of the file followed by a ".",
#
-- the extension followed by a ";", and
#
-- the version number.
#
The name and the extension are each a sequence of letters and
digits; the version number is an unsigned integer. For example,
"HENRY.IV;2" is a file designator.
#
The Designator Dialog
#
A ;Yank-File or ;Unget-File Command obtains its argument in an
unusual way: it asks for it. the dialog proceeds as follows:
#
-- The user types the command (";Y" or ";U" or ";S")
followed by an Escape.
#
-- TECO types "INPUT FILE:" or "OUTPUT FILE:", as
appropriate.
#
-- The user types all or part of a file designator,
followed by an Escape.

# -- TECO types a brief message acknowledging the
# correctness of the file designator and asking for
# confirmation.
#
# -- The user types a Carriage Return to confirm.
#
# -- TECO performs the input or output operation and
# then types "*" when it is ready for further
# commands.
#
# When things do not go smoothly, one of the following cases
# applies:
#
# -- If TECO finds that a file designator is illegal,
# TECO types "?" and prompts the user to try again.
# This occurs when a file requested for input does
# not exist or when a file designator is ill-formed.
#
# -- If the user decides to abort the command before
# typing the confirming Carriage Return, he must type
# two Delete characters. TECO will then go to the
# await-commands state without performing any
# input/output.
#
# -- If the user decides to abort the command after
# typing the confirming carriage return, it is too
# late. He should let the operation run to
# completion and then take whatever action is
# appropriate. Otherwise, he must cope with a
# partially completed input/output operation, and
# this requires study of the section called
# "Complicated Input/Output".
#
#
# Designator Recognition
#
# When the user is inputting a file, he can type just enough of the
# name and extension to uniquely specify the file in his directory
# and then type the Escape character. TECO will then ascertain and
# type out the remainder of the designator. Occasionally, the
# designator assumed by TECO will not be what the user wanted, and
# he can use two Delete characters to get out of the designator
# dialog and start a new input/output command. On other occasions,
# TECO will decide that the file designator thus far typed is
# inadequate to uniquely specify a file and will ring the bell at
# the user's terminal to ask for more characters of the designator.
#
# As a special case which is very useful, the user can reply
# to the request for a file designator by typing Escape
# immediately, without giving any part of the required file
# designator. TECO will assume (and type out) the designator for

# the file last used in a ;Yank Command in the current TECO
session.
#
It is a good rule to let TENEX fill in the version number of
a file. That is, even if the user does not make general use of
designator recognition, he should not type the version number.
When the user types Escape, TENEX will fill in the appropriate
version number. For an output file, TENEX will supply "1" for a
new file or a version number one greater than the highest version
# of an existing file. For an input file, TENEX will supply the
highest version number of an existing file.
#
If the rule just mentioned is followed, a new version number
will be created every time a file is edited, and no previous
version will be overwritten. This convention makes it easy to
keep the previous version of a file until the integrity of a new
version has been established. When an old version is no longer
wanted, it can be deleted by the TENEX Executive Command DELETE.
Thus the preparation of text and the housekeeping of the file
directory are separate activities.
#
Preparing a Long File
#
It is prudent to pause from time to time in preparing a large
file and save a copy of the file as it currently stands. This
limits the loss which can result from a system crash, the break
down of the communication line, or a serious user error. In the
following dialog, the user saves two intermediate versions of his
file before outputting the third and final version.
#
#

```
#       @TECO(%)          The user starts TECO,
#       * ...             types in a lot,
#                         then pauses to save a copy.
#       *;S(ESC)$(%)
#       (%)
#       OUTPUT FILE: HENRY.IV;(ESC)1 [New File](%)
#       (%)
```

```
    * ...               The user continues to type in his
                        file.  The next time he saves a
                        copy, TECO types the designator.
    *;S(ESC)$(%)
    (%)
    OUTPUT FILE: (ESC)HENRY.IV;2 [New Version](%)
    (%)
    * ...               The user types the remainder of his
                        file and outputs the complete copy.
    *;U(ESC)$(%)
    (%)
    OUTPUT FILE: (ESC)HENRY.IV;3 [New Version](%)
    (%)
```

In a complicated project and especially in a project in which files are shared among many users, it is useful to maintain a record of the time and source of each new version of a file. The ;Date-and-Unget-File Command (;D) provides a way to keep this record within the file itself. This command is equivalent to the ;Unget-File Command (;U) except that it adds a date line at the beginning of the buffer just before outputting the buffer. The date line consists of

-- a comment string (usually just a single semicolon), followed by

-- the complete file designator, followed by

-- the date and time at which the new version is being written out, followed by

-- the name of the user who has produced the version.

If this command is used to perform output of every new version of a file, a log of all modifications to the file will be accumulated at the beginning of a file. Since each date line begins with a semicolon, it is ignored by the many TENEX subsystems which treat such a line as a comment. Since a date line is just an ordinary line of text, it can be deleted when it becomes superfluous.

POSITIONING THE POINTER

The pointer is always located between two characters or, as a special case, at the beginning or end of the buffer. The pointer is not a character itself, and does not take up any space in the buffer. TECO editing is built up around the moving of this pointer back and forth through the text, and many of the TECO commands operate relative to the current position of the pointer.

TECO maintains several registers which contain integer values with special significance. The register named "." (period) always contains the number of characters in the buffer before the current position of the pointer. The register named "Z" always contains the number of characters in the entire buffer. "B" contains Ø, which is the beginning of the text buffer. ";B" is the character address at the top of the current page, while ";Z" is the location of the bottom of the current page. If ;B is preceded by a number, its value is the location of the top of that page. The user can refer to these registers by name wherever an integer value is required in a TECO command.

The next two commands to be considered are the simplest in TECO. Each specifies the positioning of the pointer by an integer argument. As is the case with many TECO commands, this integer argument can be omitted and the TECO interpreter will fill in a well-defined default value.

<u>The Jump</u>

The <u>J</u>ump Command (nJ) places the pointer after the <u>n</u>-th character of the buffer.

28J	Places pointer after the first 28 characters of the buffer.
ØJ	Places the pointer at the beginning of the buffer (after Ø characters).
J	Means ØJ.
ZJ	Places the pointer at the end of the buffer (after all Z characters).
Z-1J	Places the pointer just before the last character of the buffer.
.+3J	Advances the pointer across 3 characters.
;BJ	Jump to the top of this page.
;ZJ	Jump to the bottom of this page.
24;BJ	Jump to the top of page 24.

<u>The Character Skip</u>

The <u>C</u>haracter-Skip Command (nC) advances the pointer <u>n</u> characters through the buffer.

3C	Advances the pointer across 3 characters.
C	Means 1C.
-29C	Moves the pointer 29 characters backward (toward the beginning of the buffer).
-C	Means -1C, moves pointer backward one character.

The Line-Skip

Certain commands in TECO are *line-oriented*. They consider each End-of-Line character in the buffer to be the last character of a "line" of characters, and consider any character after the last End-of-Line character in the buffer to be a line. They consider the line which contains the character just after the pointer to be the "current" line of the buffer. The line-oriented commands operate on the buffer in terms of these imaginary divisions.

The first of the line-oriented commands is the *Line-Skip Command* (nL). This command advances the pointer to the beginning of the *n*-th line after the current line. The argument to this command does not have to be positive. When *n* is zero, the phrase "the *n*-th line after the current line" means "the *current* line" and when *n* is -5 (for example), the phrase means "the fifth line *before* the current line".

3L	Advances pointer to the beginning of the third line after the current line.
L	Advances pointer to the beginning of the next line (1L is assumed).
ØL	Moves pointer back to the beginning of the current line.
-L	Moves pointer back to the beginning of the previous line (-1L is assumed).
-12L	Moves pointer back 12 lines.
:L	Move to the end of the current line.
-:L	Move to the end of the previous line.
2:L	Move to the end of the next line.

Combinations

Since an individual TECO command can be just one or two characters long, it is convenient to run a few commands together when they perform an operation which, from the user´s point of view, is unitary. The first of the following examples is particularly useful.

L-C	Places the pointer just after the text of the current line (and just before the End-of-Line character).
J3L	Places the pointer at the beginning of the fourth line of the buffer.
ZJØL	Places the pointer at the beginning of the last line of the buffer.
ZJØLC	Places the pointer after the first character of the last line of the buffer.

The last example of a command string could give an error message. If the last character of the buffer is an End-of-Line character then the last line of the buffer is the empty string of characters between the End-of-Line and the end of the buffer. In this case, "ØL" leaves the pointer at the end of the buffer. This situation arises often in actual practice.

TYPING OUT TEXT AND INTEGERS

The user examines the contents of the buffer by means of the Type-Out Commands. The simplest of these is the Type-String Command (m,nT). It types everything from just after the m-th character of the buffer to just after the n-th character.

TECO has a convenient abbreviation, "H", for the argument pair, Ø,Z which designates the contents of the whole buffer. This abbreviation can be used with the Type-String Command.

HT	Types the whole buffer.
Z-1Ø,ZT	Types the last 1Ø characters in the buffer.
Ø,.T	Types the buffer up to the pointer.
;B,;ZT	Types the current page.

Line-Oriented Type-Out

A more convenient Type-Out Command is the Type-Lines Command (nT), which types the characters between the pointer and the beginning of the n-th line after the current line. The argument, n, can be zero or negative, as with the Line-Skip Command.

3T	Types characters from the pointer up to the beginning of the third line after the current line.
T	Types the part of the current line which is *after* the pointer (1T is assumed).
ØT	Types the part of the current line which is *before* the pointer.
-12T	Types the previous 12 lines and the part of the current line before the pointer.

The Type-String and Type-Lines Commands have the same code name, "T", but they are distinguished by having a pair of arguments and one argument, respectively.

A second line-oriented type-out command is the View Command (m,nV), which types m-1 lines before the current line, then types the current line, then types n-1 lines after the current line. When the two arguments, m and n would be equal, the command can be used with a single argument (nV).

1Ø,2V	Starts with the ninth line before the current line and ends with the first line after.
2V	Types the preceding line, the current line, and the following line ("2,2V" is assumed).
V	Types the current line ("1,1V" is assumed).

Three Type-Out Commands have been described here although, in theory, one would be sufficient. Each has its particular application. The Type-String Command, with its fussy generality, is seldom useful except for the special form, "HT", which types out the whole buffer. The Type-Lines Command, with its sensitivity to the position of the pointer, is used to check the pointer position just before the insertion or deletion of text. Finally, the View Command, with its convenient simplicity, is used to look at the general context in which the pointer appears.

```
# Examples of Type-Out
#
# For this dialog, the buffer contains the alphabet as it was typed
# in earlier, in the examples of the Insert-String Command.
#
#        *HT(ESC)$(%)            This command types out
#        ABCDE(%)                the whole buffer and, as
#        FGHIJ(%)                promised, reveals the
#        KLMNO(%)                alphabet, stored 5
#        PQRST(%)                letters per line.
#        UVWXY(%)
#        Z(%)
#        (%)
#
#        *1Ø,15T(ESC)$(%)        What does this do?
#        J(%)                    (Everybody who expected
#        KLM(%)                  "KLMNO" raise their hand.)
#
#        *15J(ESC)$(%)           Puts the pointer before "N".
#        (%)
#        *2T(ESC)$(%)            Types the remainder of
#        NO(%)                   the current line and
#        PQRST(%)                all of the next line.
#        (%)
#        *ØT(ESC)$(%)            Types the current line
#        KLM(%)                  up to the pointer.
#
#        *V(ESC)$(%)             Types the entire
#        KLMNO(%)                current line.
#        (%)
#        *2V(ESC)$(%)            Types the current
#        FGHIJ(%)                line and one neighboring
#        KLMNO(%)                line in each
#        PQRST(%)                direction.
#        (%)
#
# TECO follows the successful completion of any command  by  typing
# out  an  End-of-Line.   It  follows that a blank line will appear
# after a type-out if and only if the text  being  typed  out  ends
# with  an  End-of-Line.  For example, the alphabet typed out above
# ends with an  End-of-Line  and  is  followed  by  a  blank  line,
# indicated by the solitary "(%)".
#
# Abbreviated Commands
#
# A line feed character simulates the command string  "LT(ESC)"  if
# it  appears  as  the  first  character  in  a  command.  Thus, it
# advances the pointer to the beginning of the next line  and  then
# types that line.
#
# Backspace (^H) types the preceding line by simulating the command
```

# "-LT(ESC)". As with linefeed, backspace operates in this fashion
# only if it is the first character in the command.
#
Typing Out Integers
#
The Type-Integer Command (n=) types out the value of the integer
expression, n. The command is useful in obtaining the value of
"." (the number of characters before the pointer) or "Z" (the
number of characters in the buffer), but it is not limited to
this purpose.
#
The Access-Code (1A) Function of TECO has as its value the
ASCII code for the character which immediately follows the
pointer. This function can be used wherever an integer value is
allowed. In particular, it can be used as the argument to the
Type-Integer Command just described.
#
Under certain circumstances, the Access-Code Function can be
essential. The situation is analogous to the problem of
inserting control characters into the buffer, which was discussed
earlier when the Insert-Code Command was introduced. Just as
there are certain characters which can be inserted only by means
# of their integer codes, so there are certain characters which can
only be examined in this way. Some of the characters do not
print out anything (not even a space) when the string of which
they are included is typed out; for example, Control-A. Other
characters are modified by TENEX as they are typed out; for
example, a lower-case letter is capitalized when it is
transmitted to a terminal which does not have lower case.
#
The following example assumes the buffer contains the
alphabet, as before.
#
```
#       *15J(ESC)$(%)   Puts the pointer after the
#       (%)             15-th character.
#       *.=(ESC)$(%)    Shows that the pointer is
#       15(%)           after the 15-th character.
#       (%)
#       *Z=(ESC)$(%)    Shows that there are 32
#       32(%)           characters in the buffer.
#       (%)
#       *1A=(ESC)$(%)   Shows that the character
#       78(%)           after the pointer is "N"
#       (%)             (decimal code 78) and not "n"
#                       (decimal code 1Ø9).
```
#
DELETING CHARACTERS
#
The user can delete a substring from the buffer by any of three
commands. The simplest of the three is the Kill-String Command
(m,nK). It deletes everything from just after the m-th character

of the buffer to just after the n-th character and then moves the pointer to the position of the deleted characters. The special form "HK" deletes the contents of the whole buffer.

A given Kill-String Command deletes exactly what a Type-String Command with identical arguments types out. This makes it easy to "simulate" a deletion (by typing out the string to be deleted) before performing the actual deletion.

The Kill-Lines Command (nK) is the line-oriented Deletion Command of TECO. It deletes the characters between the pointer and the beginning of the n-th line after the current line. Again, in parallel with the Type-Out Command, this command deletes exactly what the Type-Lines types out.

K	Kill the (remainder of) current line.
:K	Kill the (remainder of) current line, but not the end-of-line character.
3K	Kills the (remainder of) current line and the two following it.
3:K	Same as 3K but the final end-of-line is left.
;B,;ZK	Kills this entire page.
.,;ZK	Kills the part of the page after the pointer.
-K	Kill the preceding line and the initial portion (before ".") of this line.
ØK	Kills the part of this line to the left of the pointer.
-:K	Same as -K except an additional end-of-line character is also killed.

The Delete-Characters Command (nD) operates relative to the pointer. It deletes the n characters just after the pointer. There is no Type-Out Command which is directly analogous to this command; but this command is normally used to delete just a few characters and is used more casually.

8D	Deletes the 8 characters just after the pointer.

D	Deletes the character just after the pointer (1D is assumed).
-D	Deletes the character just before the character (-1D is assumed).

The End of the Alphabet

In the following dialog, the alphabet (as entered by the Insert-String Command many pages ago) makes its farewell appearance.

```
*JCDDV(ESC)$(%)       Starts at the beginning, skips
ADE(%)                a letter, deletes two, checks.
(%)
*2,7T(ESC)$(%)        Simulates deletion of
E(%)                  characters
FGH(%)                3 through 7.
*2,7K(ESC)$(%)        Performs deletion.
(%)
*C6DC4DC(ESC)$(%)     Does more deletions.
(%)
*3T(ESC)$(%)          Simulates deletion
T(%)                  from pointer
UVWXY(%)              through next
Z(%)                  2 lines.
(%)
*3KHT(ESC)$(%)        Performs deletion.
ADIOS(%)              and checks result.
(%)
```

CONTROLLING TECO

At the beginning of this section, we observed that the ordinary commands of TECO are executed only when an Escape is typed. In addition to these ordinary commands, however, TECO has certain <u>control commands</u> which are single characters and which are <u>executed immediately</u>. These commands are used to control TECO itself rather than to edit the text in the buffer.

Interrupt TECO

The user sometimes needs to access the TENEX EXECutive. For example, he may want to use the DIRECTORY command to check existing file names before choosing a name for a new file he has prepared; he may LINK to another TENEX user without wrapping up his TECO session; or he may wish to eliminate an unwanted file by means of the DELETE command. To get back to the TENEX EXECutive he uses the TECO Control Command Control-C (^C) and ends with the EXECutive command CONTINUE, as in the following dialog:

```
     *IA(HT)     B(%)     The user inserts a line
     (^D)-LT(ESC)$(%)     which includes a tab
     A       B(%)         and types it out
     (%)                  with standard tab stops.
     *(^C)^C(%)           Interrupts TECO.
     @STOPS 3(%)          Uses Exec to set stop.
     @CONTINUE(%)         Returns to TECO.
     T(ESC)$(%)           Types out the line
     A  B(%)              with new tab stops.
     (%)
```

The Control-C Command does not always result in an immediate interrupt. If TECO is performing input/output, the interrupt will be delayed until the buffers have been properly emptied. To obtain an immediate interrupt, the user can type a <u>second</u> Control-C; when he does so, however, data in the buffers <u>may be</u> lost. Usually this is acceptable only during a type-out at the user's terminal.

Abort a Command

A different kind of interrupt is produced by the Delete Command (DEL), also called RUBOUT. Two cases must be considered:

- -- If Delete is typed during <u>execution</u> of a command, that command is <u>immediately aborted</u> and TECO types "*" and enters its await commands state.

- -- If Delete is typed during <u>type-in</u> of a command (when TECO is <u>not executing a command</u>), TECO rings the bell on the terminal and waits to see what the user does next.

  - -- If the user types a second Delete, TECO discards the command string thus far typed in, types "*", and enters the await commands state; however,

-- If the user types anything but a Delete, TECO assumes the first Delete was a mistake and forgets it.

The cautious handling of this case is appropriate because the user could type many lines of text as an unfinished Insert-String Command and then accidentally type a Delete. Rather than wiping out the type-in, TECO rings the bell to warn the user not to type Delete again.

In the following dialog, the user starts an input command, gives the wrong file designator, and deliberately aborts the command instead of confirming it. A second try causes the desired file to be read in. The user instructs TECO to type the whole file, reads the first few lines, decides it looks right, and aborts further type-out.

```
     *;Y(ESC)$(%)
     (%)
     INPUT FILE: HENRY.VI(ESC);1 [Confirm](DEL)(BEL)(DEL)
                         (%)
     *;Y(ESC)$(%)
     (%)
     INPUT FILE: HENRY.IV(ESC);3 [Confirm](%)
     21982 Chars(%)
     (%)
     *HT(ESC)$(%)
     So shaken are we,(%)
     so wan with care,(%)
     Find we (DEL)(%)
     (%)
     (%)
     *...
```

Checking up on TECO

An interrupt of a special kind is produced by the Control-T Command. This command can be used at any time at all and will promptly report on the status of the System, giving the user the status of TECO (waiting for input or running), the TENEX load average, and the CPU and console time used. Since the command never disturbs the command being executed by TECO, it is a safe and convenient way to check up on TECO when TECO seems to be taking a long while to execute a command.

The Control-T Command can be used to determine whether a delay in the response of TECO is due to a heavy load on the TENEX System, a minor breakdown in TENEX, or an infinite loop entered by the user (this facility will be explained later). But an especially interesting example is the following recovery procedure.

When there has been a failure in the TENEX System, the user may find himself in a somewhat uncertain position in TECO. Either (1) there has been a crash of TENEX TECO and the system has been restored without the loss of its previous state or (2) the communication link between the user and TENEX has been broken and the user has used the EXECutive ATTACH command to re-establish the connection and resume at the point of interruption. The uncertainty results from certain random processes which may occur during the breakdown. The user should explore the situation as follows:

```
     (^T)IO WAIT AT 4262(%)
     LOAD AV. =   Ø.Ø7,  USED Ø:Ø2:Ø8.6 IN Ø:48:Ø6(%)
                         First,the user determines the status
                         of TECO.
                         Apparently TECO is waiting for a
                         command.
                         Next, the user looks at the current
                         command string:
     (^R)(%)
     BTRFSK#!(DEL)(BEL)(DEL)(%)
                         The command string looks like line
                         noise and the user erases it.
     * ...               The user proceeds, if necessary, to
                         check the state of TECO in other
                         ways appropriate to the situation.
```

Erasing Typing Errors

A command string is not executed as it is typed in. Instead, it is accumulated in a special command register and is executed only when an Escape is typed. The three commands given here are specialized commands designed to assist in correcting a command string as it awaits execution in the command register.

Each erasing command is a single control character, and acts immediately when it is entered. The Control-A causes the last character in the command register to be erased. The Control-Q Command causes the last non-empty line in the command register to be erased. The Control-R (for "Retype") Command causes the last non-empty line in the command register to be typed out.

In the following dialog, the user inserts three words and types out the buffer. Along the way, he illustrates the use of single and multiple erasures and of the retyping of a line to "clean it up" after it has been cluttered by erasures.

```
*HASTE(%)                         The user starts
MAKES(%)                          an insertion.
(^Q)  (%)                         He notices the "I"
(^Q)(%)                           is missing, erases
*IHASTE(%)                        the command string
MAKE(%)                           and starts over.
WA(^A)\A(^A)\W(^A)\(%)            He erases "A", "W",
(^R)(%)                           and an End-of-Line
MAKES(%)                          and Retypes the line.
WASTE.(%)
(ESC)$(%)
(%)
*HK(^A)\KT(ESC)$(%)               The user almost clears
HASTE(%)                          the buffer, but
MAKES(%)                          changes "K" to "T".
WASTE.(%)
(%)
```

Erasing typing errors using Backspace Key

Some terminals are able to perform the backspace function. Hardcopy devices do this by moving the print head; CRT displays can move the cursor. To take advantage of this, Control-H or Backspace deletes characters just as Control-A does but indicates what has been deleted by using the mechanical backspace mechanism. In order to activate this function, the user must tell TECO what style terminal he is using. This is done by commands such as 3^H$. (Four characters: 3, ^, H, and ESCape.) Instead of 3 the user should select one of the following:

Code	Terminal
Ø	No mechanical backspaces (Model 33)
1	Mechanical backspace but no eraser. (TI, 2741)
2	Scope that uses ^H to backspace the cursor.
3	Bendix scope, Backspace sequence is ESCape D.
4	Terminal, VTØ6 scope. Backspace character is ^Y.
5	Beehive. Backspace character is ^D.
6	Infoton, Backspace character is ^Z.

2. LARGE-SCALE EDITING

The previous section described a collection of commands. Those commands can be used to select any position in the text being edited and then insert or delete any characters at that position. Additional commands are required when the file being edited is large and the modifications being performed are complicated.

When a file is more than a few dozen lines long, it is not efficient to locate a position within the file by counting lines or characters; instead, commands are required which can locate a particular phrase or identifier or number within the buffer. When a passage of text which is more than a few words long must be moved, it is not efficient to delete the passage and then retype it elsewhere; instead, commands are required which can extract, move, and insert the passage without retyping. When a particular modification must be made over and over (as in the case of a consistently misspelled word), it is not efficient to type the necessary command string over and over; instead, some kind of loop command is required.

The commands described in this section fill the requirements just mentioned. Although the commands described are introduced here by the problems of large-scale editing, they are useful for all editing jobs. The commands of the previous section are the framework of TECO; the commands in this section supply the power.

SEARCHING THROUGH THE TEXT

The Search Command is used to search the buffer for an occurrence of a particular substring. It is entered by typing "S" followed by a character-string argument. The character-string argument is a sequence of characters, the *citation*, terminated by typing Control-D. The Control-D can be omitted when a Search Command occurs at the end of a command string.

Ordinary searches look *after the pointer* for a character sequence which matches the citation. If the repetition count (v.i.) is negative, a reverse search has been specified and the search will proceed *backwards from the pointer*. In either case a match is found, the pointer is moved to the position just *after* the matched character sequence; otherwise, the pointer is not moved from its original position and TECO types the error message "?SEARCH?35".

The Search Command can be preceded by an integer value, *n*, and will thereupon be repeated *n* times. The effect is to search for the *n*-th occurrence of the citation after the pointer.

Sa(^D)	Finds the first occurrence of "a" after the pointer and moves the pointer to just after that occurrence.
-Sa(^D)	Finds the first occurrence of "a" before the pointer.
Sit(^D)	Finds "it" in any context, either as an independent word or within another word.
S(%) Here (^D)	Finds an occurrence of the word "Here " at the beginning of a line.
3S;(^D)	Finds the third occurrence of a semicolon after the pointer and moves the pointer to just after that occurrence.
-3S;(^D)	Finds the third occurrence of semicolon before the pointer. The pointer is left after the find.

A successful search of the buffer always moves the pointer. This is taken for granted in the explanation of some of the examples.

The Search Command can be deceptive. It is quite natural for a user to give a search command for a particular pattern, get an error message in response from TECO, and conclude that the specified substring is not present in the buffer. However, this conclusion is unwarranted if the user forgot to set the pointer to the beginning of the buffer before the search.

The Replace Command

A natural extension of the Search Command is the Replace Command, which not only searches the buffer but modifies it. The command is entered by typing "R" followed by two character string arguments, the *citation* and the *replacement*. Each character string argument is terminated by typing Control-D. The command does exactly what the Search Command does and one thing more: if the substring specified by the pattern is found, it is deleted and a copy of the replacement is inserted. Replace commands also may take a repetition count which can be negative to specify a reverse replace.

Ra(^D)b(^D)	Finds the first occurrence of an "a" after the pointer, moves the pointer to just after that occurrence, and replaces it with "b".

R(%) Here (^D)(%) There (^D)	Replaces the next "Here " which begins a line with a "There ".
-3R@#!(^D)(^D)	Replaces the past three "@#!" strings with nothing; that is, deletes them.
R(%) (^D)(^D)	Deletes the next End-of-Line.
Ra page(^D)a(%) page(^D)	Starts a new line between the words "a" and "page".

The Replace Command is perhaps the most frequently used command in TECO. When a person is making the changes indicated in a marked-up listing of a file, he can often proceed from beginning to end with one Replace Command after another. When there is danger that a Replace Command may apply in the wrong place, the user can simply include a little more context in the pattern. Consider, for example, the following ways of making "big plans" into "big plane".

Rs(^D)e(^D)	This works if the site of the change is just a few characters after the pointer and there is no intervening "s".
Rplans(^D)plane(^D)	This is more selective and will be right unless there is another "plans" along the way to the site of the change.
Rbig plans(^D)big plane(^D)	This is still more selective.
Sbig plans(^D)-DIe(^D)	This saves a few keystrokes but it is more complicated and error prone.

It is good practice to follow a Replace Command with a View Command. The type-out verifies that the citation and replacement were correct and that the modification was applied at the right place in the buffer. Further, when the user is in the habit of typing out each change, he can risk small errors, such as a misplaced replacement, in order to work faster.

Something Extra

Three match control characters are provided for use in the pattern of a Search Command or Replace Command. They are:

(^X)	Matches any character which appears at the corresponding position in the buffer.
(^S)	Matches a Separator; that is, any character except a letter, digit, ".", "$", or "%". (These are the characters which are commonly used in identifiers.)
(^N)	Matches any character except the character which immediately follows the (^N) in the citation.

The match control characters can be used in a citation with other characters in any combination or sequence. However, a Control-N and the character which follows it act as a pair to match (or not match) a single character of the buffer.

S[(^X)(^X)](^D)	Finds any two characters enclosed in brackets.
S(^S)it(^S)(^D)	Finds an "it" which is not a part of a longer word,
S(^S)(^N)(^S)a(^D)	Finds a word whose second letter is "a". Note that "(^N)(^S)" are used in combination to match anything except a Separator character.

An Application

The Search Commands cannot be convincingly applied to the example we have been using (the alphabet). This is precisely because they are designed to look at the content of the buffer. Accordingly, we assume conventional text has been typed into the buffer, errors found, and a decision to correct them.

```
     *Sfield(^D)$V(ESC)$(%)     The user finds a "field",
     a great battlefield(%)     checks, and sees that
     (%)                        it is the wrong one.
     *Sfield(^D)$V(ESC)$(%)     The user finds the next
                                "field",
     field as a(%)              and makes sure it is
     (%)                        the right one.
     *I,(^D)$V(ESC)$(%)         He inserts a comma,
     field, as a(%)             and checks his work.
     (%)

     *Rbefoer(^D)$before(^D)$V(ESC)$(%)
     remaining before us,(%)    The user corrects
     (%)                        a typing error.

     *JSbefoer(ESC)$(%)         The user checks for
                                another
     ?SEARCH?35(%)              instance of this error,
     JSbefoer$(%)               but there is none.
```

The last Search Command in the dialog produced an error message; nevertheless, it illustrates a useful and legitimate application of the Search Command. The user wanted to know if there was an instance of "befoer" in the buffer, and the error message supplied the answer "no". A Search Command which fails does not move the pointer.

MOVING TEXT

It is often necessary to transport a string of text from one place in the editing buffer to another. This operation is required when multiple copies of a given string must be made or when a string must be moved which is too long to be conveniently deleted and retyped. The Q-Registers and their associated commands are used for this operation.

There are 37 Q-Registers, and each of them can hold a character string of virtually unlimited length. Each Q-Register has one of the 26 letters or the 10 digits or @ as its name, and all the Q-Register Commands (except one) end with the name of a Q-Register. The upper and lower case forms of a letter are equivalent as a Q-Register name, just as they are when used in a command name.

Setting a Q-Register

There are two commands for moving a substring of the buffer into the Q-Register. The Extract-String Command (m,nXq) extracts from the buffer everything from just after the m-th character to just after the n-th character. The Extract-Lines Command extracts everything between the pointer and the beginning of the n-th line after the current line. Each command removes the selected string from the buffer and places it in the Q-Register q. The pointer is left where the extracted string was, and the previous contents of the Q-Register are lost.

HXF	Extracts the contents of the entire buffer (leaving the buffer empty) and puts it in Q-Register F.
Ø,23XA	Extracts the first 23 characters and places the sequence in Q-Register A.
5XX	Extracts a substring and puts it in Q-Register X. The substring extends from the pointer to the beginning of the 5-th line after the current line.
-4X@	Extract the preceding four lines and put them into Q-Register @.
:Xl	Extract the remainder of the current line except the end-of-line.

The two Extract Commands move exactly that substring of the buffer which the corresponding Type-String or Type-Lines Command types out. Thus the user can use one of these Type-Out Commands to simulate an extraction.

Using a Q-Register

The Get-QR Command (Gq) inserts a copy of the character string in Q-Register q into the buffer just before the pointer. The contents of the Q-Register is not changed. When an Extract Command which refers to Q-Register q is immediately followed by "Gq", the total effect is to copy a substring of the buffer into a Q-Register without deleting it from the buffer.

The ;Type-QR Command (Qq;T) types out the complete character string contained in Q-Register q. It can be used to check on the successful execution of an Extract Command. This is the one Q-Register Command which does not have the Q-Register name, q, at the end of the command.

GA	Inserts a copy of the contents of Q-Register A into the buffer just before the pointer.
5,62X3G3	Extracts characters 6 through 62, puts them in Q-Register 3, and copies Q-Register 3 into the buffer. Leaves the buffer unchanged.

```
     QZ;T      Types-out the character string in
               Q-Register Z.
```

Merging Files

The commands just described can be used for very general and extensive manipulations of files -- merging, interleaving, and so on. For example, to merge parts of File A into File B proceed as follows:

-- On listings of Files A and B, mark the parts of File A with the names of Q-Registers, and use these Q-Register names to indicate where the parts are to go in File B.

-- Read File A into the buffer, extract the parts into the appropriate Q-Registers, and delete the remainder of the file.

-- Read File B into the buffer and, for each insertion, position the pointer in the buffer and insert the contents of the appropriate Q-Register.

This is a good procedure for making large-scale patches to any program or document.

An Application

The following dialog places at the beginning of the buffer an introductory sentence which cites the opening words of the text which is already in the buffer:

```
     *JSago(ESC)$(%)           Finds end of the opening.
     (%)
     *Ø,.XAGA(ESC)$(%)         Extracts and restores
     (%)                       the opening.
     *QA;T(ESC)$(%)            Types out the contents
     Fourscore and seven(%)    of Q-Register A
     years ago(%)

     *J(ESC)$(%)               Moves pointer to the
     (%)                       top of the buffer.

     *IThe Address begins(%)   Makes up the sentence.
     "(^D)$GAI"(%)
     The entire text is:(%)
     (ESC)$(%)
     (%)
```

```
#       *HT(ESC)$(%)              Types the result.
#       The Address begins(%)
#       "Fourscore and seven(%)
#       years ago"(%)
#       The entire Text is:(%)
#       Fourscore an(DEL)(BEL)(%)
#       (%)
#       (%)
#       *
#
```

# STORING INTEGERS
#
# A Q-Register can also be used to hold an integer value. This
# value is almost always used to save the position of the pointer
# in the buffer, but it is not restricted to that purpose. If the
# user wants to do some integer calculations, he can use
# Q-Registers as his variables.
#
# Only two commands are required and they are very simple.
# The Update-QR Command (nUq) loads the integer value n into
# Q-Register q, destroying the previous contents. The Q-Value
# Command (Qq) is actually a function. Its value is the contents
# of Q-Register q, and it can be used wherever an integer value is
# accepted.
#

```
#       23UA       Puts the integer 23 into Q-Register A.
#
#       .U5        Puts the current pointer position (the
#                  number of characters before the pointer)
#                  in Q-Register 5.
#
#       Q5J        Puts the pointer at the position
#                  specified by the integer in Q-Register 5.
#
#       QA+2UA     Increases Q-Register A by 2.
#
#       QA=        Types out the integer in Q-Register A.
#
```

# The Q-Register Commands for character strings and for integers
# are not mutually compatible. For example, if a Q-Register is
# loaded with the character string "398" by an Extract-String
# Command and an attempt is then made to get its value with the
# Q-value Function, TECO will object and type an error message.
# All of the Q-Registers initially contain the integer value Ø (as
if a ØUq had been executed).
#
# An Application
#
# In the preceding dialog based on the Gettysburg Address, the
# first sentence of the Address was extracted. That was relatively
# easy because the beginning of the sentence had a known position,

Ø, in the buffer. In the dialog which follows, a sentence is extracted from the middle of the Address and both ends must be located. Q-Register Ø is used to save the position of the beginning of the sentence while the end is found.

```
     *Sdid here.(ESC)$(%)      Finds previous period,
     *V(ESC)$(%)               types end of previous
     they did here.(%)         sentence.
     *LT(ESC)$(%)              Gets to beginning
      It is for(%)             of desired sentence and
     *.UØ(ESC)$(%)             saves pointer.
     *S.(^D)$T(ESC)$(%)        Finds end of sentence
      It(%)                    and checks.
     *QØ,.XC(ESC)$(%)          Picks up sentence.
     *QC;T(ESC)$(%)            Types out sentence.
     It is for(%)
     us the living,(%)
     rather, to be(%)
     ...
```

LOOPS

Although TECO has general facilities for both conditional and unconditional transfer of control, it is the simple and specialized Iteration Command which is most useful.

In order to repeat any command string n times, (1) place the command string in angle brackets, "<" and ">", and (2) put n in front of the bracketted string. If n is omitted, an approximation to infinity (2^35) is assumed, and the loop will continue until something in the command string stops it.

```
     3<R;(^D),(^D);  V>  Starts at the current  position
                         of the pointer and replaces the
                         next 3 semicolons with  commas.
                         Types each modified line.

     5<L18<I.(^D)>>      Puts   18   periods   at   the
                         beginning  of  each of the next
                         five lines.

     J<Ryclepped(^D)yclept(^D);  V>
                         Corrects  all  misspellings  of
                         "yclept" in the buffer, however
                         many there are.
     J<S(^S)command(^S)(^D);  V>
                         Types out every line  in  which
                         the word "command" appears.
```

S and R commands inside iteration brackets cause the iteration to terminate if the search fails. Thus, if the buffer contains exactly one occurrence of the string "abc" the command J<Sabc(^D)V> will print the line containing the "abc" twice -- once when the search succeeds and again when it fails. This is because the V command is seen before the > which will cause the iteration to stop.

A Q-Register can be used to count the number of times a loop is executed. A special function, %q, is provided which increases the integer in Q-Register q by one and then assumes the resulting integer value. This function can be used as a free-standing command if the user types Control-D after it to "absorb" its integer value.

```
     96UA26<%AI>         Inserts a complete lower-case
                         alphabet into the buffer.

     ØUAJ<S;(^D)%A(^D)>  Counts the semicolons  which
                         appear in the buffer and leaves
                         the result in Q-Register A.
```

<u>A Final Example</u>

The following loop searches the buffer for the occurrences of the word "command". For each occurrence of the word, the loop types the line in which the word appears, stops to let the user type in a single character, and then capitalizes the words if the user typed "Y" (for "yes").

```
     J<Scommand(^D);  V^T-^^Y"E-7CRc(^D)C(^D)´>
```

This command string uses some commands which have not been described yet and, in any case, needs some explanation. An annotated listing follows:

J	Puts pointer at beginning.
<	Starts loop.
Scommand(^D)	Searches for "command".
; V	Skips to end of loop if search fails. Types out the line containing "command".
^T-^^Y	"^T" is a function which assumes the value of the code for the next character the user types (TECO waits for the user). "^^Y" is the code for "Y".
"E	If the preceding expressions is Equal to zero, the following commands are executed; otherwise, they are skipped up to the apostrophe.
-7C	Back up to just before "command".
Rc(^D)C(^D)	Capitalize "c".
'	End conditional expression skip.
>	End loop.

This example is important. It represents TECO at its best, namely in a short, interactive loop in which the user makes the difficult judgements and TECO carries out the editing details.

3. SPECIAL EDITING

All the TECO commands which have not been described in the previous sections are mentioned in this section. Each division of this section describes a group of commands designed for a particular purpose. The descriptions are brief because the commands are not of universal interest. The purpose of this section is to inform the reader of the existence of specialized commands of TECO. A complete description of each command appears in the TENEX TECO Handbook and can be found by looking up the command name in the command index.

Automatic Indentation

An effective technique for organizing information on a page is to use the "outline" form; that is, to indent lines by varying amounts to indicate the grouping and relative importance of the information. This formatting technique is often applied to improve the readability of programs in block-structured languages, such as LISP, Algol, and PL/I.

The simplest means of indenting a line is to type in the appropriate number of leading spaces. For a long program with many different indentations, this process is tedious and error prone. TECO has a set of four special commands which can be used to supply these leading blanks automatically. The commands are:

(^W)	Records the number of spaces in a new indentation.
(^U)	Inserts spaces required for the indentation which is currently in use.
(^B)	Reverts to the indentation which was recorded just before the current one and inserts the required spaces.
(^Y)	Reverts to the indentation which was recorded just after the current one and inserts the appropriate spaces.

The first time a particular indentation is required, the user types in the necessary spaces himself and uses the Control-W command to record the indentation. Indentations are recorded in a list which is a special part of TECO storage.

Logical Operators

An integer may be represented as a sequence of binary digits (as the reader may know). TECO has operators which convert an integer into a bit string, apply a logical operation to the strings, and convert the result back to an integer. The operators are:

m#*n*	the *i*-th bit of the result is the logical "or" of the *i*-th bits of *m* and *n*
m&*n*	the *i*-th bit of the result is the logical "and" of the *i*-th bits of *m* and *n*.

Conversion of Integers

In some cases it is useful to convert a sequence of digits which occurs in the buffer into an integer argument for a command. Conversely, it may be useful to convert an integer argument into a character-string representation of the integer. The necessary commands are:

n;N	This function interprets the digit string which follows the pointer and assumes that integer value. The argument *n* specifies the base to be used in interpreting the digits (for example, n=8 for octal digits).

n\	This command expresses its argument, n, as a (possibly signed) sequence of digits of base 1Ø and inserts the sequence into the buffer just before the pointer.

Flow of Control

TECO has a rather complete set of commands for flow of control. These commands are supplied because even a short command string occasionally needs a conditional transfer or a custom-made loop to make it go. The following commands provide conditional execution of an arbitrary command string:

n"Ec'	Executes the command string c if n Equals Ø; otherwise, skips over c.
n"Nc'	Executes c if n Not-equals Ø.
n"Lc'	Executes c if n Less-than Ø
n"Gc'	Executes c if n Greater-than Ø
n"Cc'	Executes c if n is the character code of a letter, digit, ".", "$", or "%".

The following commands provide an unconditional transfer of control:

!s!	This is a label.
Os(^D)	This is a transfer to the label "!s!".

Sometimes it is not appropriate to have a Search Command produce an error message when the search fails. This is the case, for example, when a search is part of a stored program. The following alternative is provided:

:Ss(^D)	The ":" before a Search Command causes the whole command to assume an integer value of -1 or Ø according as the command succeeds or fails. Thus the command can be used wherever an integer argument is accepted. Note: Search commands and Replace commands inside iteration brackets (< ... >) act as if they have the : modifier on.
n;(SP)	The ";(SP)" causes a skip out of the enclosing loop for non-negative values of n and otherwise is ignored. (The semicolon must be followed by a Space character.) If a ":"-Search Command is used as the argument to

this command, the command will skip out of the enclosing loop when the search fails.

Stored Programs

TECO has important commands which make it possible to create a TECO program, store it, and later execute it. Specifically, the command string is typed into the buffer like any other passage of text, is placed in a Q-Register by an extract Command, and is then executed by the Macro Command, as follows:

Mq	This command causes TECO to execute as a command string the contents of Q-Register q.

Since the character string in the Q-Register q may itself contain a Macro Command, this command provides for subroutines which can be called through one another as well as directly by the user.

A minor difficulty arises in preparing programs. An ordinary Insert Command cannot be used to enter into the buffer an Insert or Search Command because the Control-D which is a part of the stored command will prematurely terminate the overall insertion. The following commands solve this problem:

@Itst	This command inserts the character string s, which may include (^D). Any character which does not appear in s can be used as the character t which delimits the character string.
@Stst	This command searches for the character string s, which may include (^D).
@Rts1ts2t	Replace string s1 by s2, both of which may contain the terminator t.

TECO has a few additional commands whose principal use is in stored programs. They are:

^T	This is a function whose value is the integer code for the next character typed in by the user. (TECO waits until the character is typed.)
;Ts(^D)	Types out s and is used to output a message from a running program.

[q Pushes down into a special pushdown stack the current value of Q-Register q.

]q Pops up the pushdown stack into Q-Register q.

? Causes TECO to "trace" its execution; that is, to type out commands as they are executed. The second "?" turns the trace off, the third turns it on again, and so on.

The basic input/output commands of TECO, as described in Section 1, obtain a file designator not from the command string but from a special dialog with the user. This operation is not appropriate for use in a stored program, and the following command sequences can be used instead:

;Rf(^D);Y
Reads in all of file f and adds it to whatever is in the buffer.

;Wf(^D);U
Writes out the entire buffer onto file f and clears the buffer.

These commands are further described in the discussion of paged input/output which follows this section.

The stored program commands and the rather complete set of flow of control commands combine to make possible some relatively complicated symbol manipulation. However, the proper use for the programming facility is to bridge the gap between simple editing and full scale symbol manipulation. TECO should not be used in competition with complete programming systems like LISP or SNOBOL. TECO does not have the debugging facilities, the data structures, or the efficiency to support such an activity.

TECO Paging

The computer on which TECO was originally implemented did not have virtual memory and its actual memory was relatively small. It was therefore necessary to devise a means by which a file could be edited piece by piece. Toward this end, the Form Feed character (Control-L) was selected as an "End-of-Page" character and each substring of a file which ended with a Form Feed was called a TECO page. (There is no relation between this "TECO page", which is purely a software notion, and the "page" which is the unit of the TENEX virtual memory.)

With the introduction of virtual storage and the consequent very great increase in the capacity of the buffer, the need for paging declined. For TENEX TECO, the possibility of the division

```
# of a file into TECO pages arises under the following conditions:
#
#   -- When   a   file   exceeds   the   (approximately)
#      one-million-character  capacity  of  the buffer, it
#      must be broken into parts.
#
#   -- When a file is to be merged with  another  file  or
#      rearranged  in  some  way,  it  must be broken into
#      parts.
#
#   -- When a file exceeds 65,000 characters (a very rough
#      estimate)   and  will  be  subjected  to  extensive
#      editing (that is, many insertions  and  deletions),
#      efficiency considerations suggest that it should be
#      broken into parts.
#
# However,  a  decision  to  break  a  file  into  parts  does  not
# necessarily  lead  to  paging  the  file.  It will often be more
# appropriate to maintain each part  of  the  file  as  a  file  in
# itself.  Thus the cases in which paging commands are essential is
# rare.
#
# For LISTing or TYPEing it is useful to have  ^Ls  on  at  logical
# places to make page boundaries simply for human use.  Also can be
# handy in moving through file using n;BJ.
#
#      The commands for handling paged files will now be  described
# briefly.  They fall into several subgroups according to the steps
# of the input/output process.
#
#      The first step is to open files for input and/or output,  as
# follows:
#
#    ;Rf(^D) Selects the file whose designator  is  f  and
#            opens it for input.
#
#    ;Wf(^D) Selects the file whose designator  is  f  and
#            opens it for output.
#
# The following commands read a page from the file  which  is  open
# for input.
#
#    Y       Deletes the contents of the buffer and  reads
#            the  next  page  of  the  input file into the
#            buffer.
#
#    A       Adds the next page of the input file  to  the
#            end of the current contents of the buffer.
#
# The following commands output a portion of the  buffer  which  is
# specified  by  its  arguments.  Two, one, or no arguments can be
```

used, and their significance is described in the Handbook section of the TECO Manual.

PW	Does output without modifying the buffer.
W	Does output and deletes from the buffer the portion which is output.

The following commands perform a combination of input and output. The ;Y and ;U Commands which appear below were discussed in Section 1; but here they have a different interpretation because they are used when an input or output file is open.

P	Outputs the current contents of the buffer and then reads in the next page of the input file. In effect, the command "turns a page" of the file being processed.
;Y	Reads in the remainder (however many pages) of the input file and adds it to the end of the buffer.
;U	Repeatedly executes P (Page-Turn) Commands until the remainder of the input file has been copied into the output file and then closes the output file.

The following command concludes the output process:

;C	Closes the output file. (For certain technical reasons, the output file is not permanently saved until it is closed.)

Two commands are provided to search an entire paged file in a single operation. Each command starts at the position of the pointer (as usual) but does not stop at the end of the buffer; instead, subsequent pages are read from the input file and scanned until a match is found or the end of the file is reached. The commands are:

<u>n</u>F<u>s</u>(^D)
Searches page after page. After a page has been searched unsuccessfully, it is written on the output file so that nothing is lost. This command is used when a file is being modified.

```
#    n;Fs(^D)
#               Also  searches  page  after  page.  However,
#               after  a  page is searched unsuccessfully, it
#               is discarded.  This command is used when  the
#               file is being examined but not modified.
#
# In contrast to the Search commands just  mentioned,  the  indices
# described  here  are  designed  for use with a multiple-page file
# which is entirely in the buffer;  no  automatic  input/output  is
# involved.
#
#    ;B         Supplies the  number  of  characters  in  the
#               buffer  before the current page.  The current
#               page is the page which contains the character
#               just before the pointer.
#
#    ;Z         Supplies the  number  of  characters  in  the
#               buffer through the end of the current page.
#
#    n;B        Supplies the  number  of  characters  in  the
#               buffer  before the beginning of the n-th page
#               in the buffer.
#
#    n;BJ       Jumps to start of n-th page.
#
# Quoting Control Characters
#
# Certain control characters can be used literally in the character
# string  argument  of  an  Insert  or Search Command when they are
# properly quoted.  The commands for this purpose are:
#
#    (^V)       In most cases, when this  character  is  used
#               immediately   before   a   command   control
#               character it will cause that character to  be
#               entered  into the command register literally.
#
#    (^V)(^Q) When this character pair is used before  one
#               of  the  three  match  control characters (as
#               used in a search command), the match  control
#               character is interpreted literally.
#
# Complete  instructions   for   quoting   characters   on   input,
# recognizing  them  on output, and specifying them in a search are
# given in the Character Set Appendix.
#
# Abbreviations
#
# There are a number of commands in TECO which can be described  as
# abbreviations  for  commonly  used  command strings.  Since TENEX
# TECO is very concise in any case, abbreviations do  not  play  an
# important  role  and  most of them are of limited interest.  They
# are:
#
#    EDIT f  This is an Executive command which is an
#            alternative to the "TECO" command.  It enters
#            TECO and then causes the file f to be read
#            into the buffer.  For a subsequent editing of
#            the file in the same TENEX session, the user
#            can just type "EDIT".
#
#    EG      This is the exit from TECO which is used when
#            TECO has been called automatically by some
#            other subsystem of TENEX.
#
#    (LF)    This Command (the Line Feed character) moves
#            the pointer to the beginning of the next line
#            and types that line.  The command must be the
#            first character in a command string.  (The
#            command was imported from DDT).  Note:  (LF)
#            and (^H) are done immediately (i.e.  (ALT) is
#            not needed).
#
#    (^H)    This command moves the pointer to the
#            beginning of the previous line and types that
#            line.  The command must be the first
#            character in a command string.  (The command
#            was imported from DDT).
#
#    ^^c     This is a function whose value is the integer
#            code of the character c.
#
#    ;P      This is a function whose value is the integer
#            code of the current character and which moves
#            the pointer one character to the right.  It
#            thus combines a "1A" Function with a "1C"
#            Command.
#
#    B       This is an index which is always Ø.  It is
#            useful because "B" is slightly easier to type
#            than "Ø".
#
#    (SP)    This operator (the Space Character) is
#            equivalent to the "+" operator.  It is easier
#            to type than "+".
#
#    (HT)s(^D)
#            This command can be used to insert a string
#            which begins with a Tab (HT) character.  In
#            effect, TECO supplies an "I" at the beginning
#            and makes this into an Insert-String Command.
#
# This completes the list of useful TECO commands.  The Command
```

# Index contains some other commands; they are either obsolete
# names which have modern synonyms or are characters such as
End-of-Line or "$" which have no effect when they appear as TECO
commands. See TENEX TECO Manual for further information.

## TTYTRB

TTYTRB is a teletype trouble report form for use within BBN
Cambridge.
#
# TTYTRB asks you for two things: the location of your terminal
# and a description of the problem.
#
When it prompts for "LOCATION OF TERMINAL:" type up to one line
# of information, terminated by carriage return. For details on
typing this information, see "Subject" under SNDMSG in this
manual.
#
When it prompts for "DESCRIBE TROUBLE:" type as much as you need
to, terminated by control-Z. For details on typing this
information, see "Message" under SNDMSG in this manual.

TTYTST

A teletype-testing program developed at BBN.

The program cycles through a series of tests which are described below. Any test may be prematurely aborted by hitting a single rubout. This will cause the next test to begin.

TEST 1 OUTPUT TEST

A series of lines of output are generated. All of the characters of the alphabet and all of the numerals are typed as well as all of the special characters. If your teletype is consistently misprinting on output, it will probably show up with this test.

TEST 2 CARRIAGE RETURN TEST

Line of somewhat random length repeating the same string of characters are typed. If these characters double up at or near the left margin, your teletype probably needs a dashpot adjustment.

TEST 3 ORDINARY TEST

Some lines of ordinary text are typed out. If your teletype is only occasionally misprinting on output, observe the output of Test 3 carefully, the problem may show up in it.

TEST 4 INPUT TEST

At this point you may type anything you like into a buffer in the program. The program will repeatedly type out the contents of this buffer after you type control D. If you make a number of mistakes use a number of control A´s to erase them. If you hit a rubout during this last test, the program will give you a chance to type in more input.

TYPBIN

TYPBIN is a subsystem which does an octal dump of a packed file. A packed file is a file where every bit is considered to be an information bit of a continuous bit string. A packed file of 8-bit bytes, for example, will contain exactly nine bytes per each two words. In contrast, a standard format file would only contain eight of these bytes per two words, left-justified in PDP-10 ILDB/IDPB format.

When TYPBIN is started, it asks:

```
BINARY FILE INTERPRETER.
INPUT FILE=
```

The correct response is any readable file name. The program asks:

```
BYTE LENGTH (DECIMAL BITS) =
```

Any number from 1 to 36 is acceptable. The program then asks:

```
OUTPUT FILE =
```

Any writeable ASCII file is acceptable. The program then reads each successive "BYTE SIZE" of bits from the input file, writing out the octal value in ASCII to the output file. This output is columnated into easily readable format for 72-column wide paper.

On completion, the program reports the total number of bytes converted and exits. Example:

```
@TYPBIN
BINARY FILE INTERPRETER.
INPUT FILE = ABC [CONFIRM]
BYTE LENGTH (DECIMAL BITS) = 18
OUTPUT FILE = ABC.DUMP [NEW FILE]
DONE.
TOTAL BYTES (DEC) = 18
EXIT.
^C
@
```

At this point, file ABC.DUMP contains:

```
777772 000137 200740 000313 402000 000362 402000 000363 561040 000314
104000 000076 205040 120003 254000 000140
```

In the past, TYPBIN has proved invaluable in deciphering the unknown formats of data read from imported magtapes using the MTACPY subsystem.

TYPREL

Typeout of .REL Files

TYPREL analyzes the contents of .REL files created by MACRO, FAIL, and FORTRAN (F40).

TYPREL first asks for an output file where analysis results are to be written. The question is "OUTPUT FILE =", and an appropriate response is any file capable of receiving ASCII output. (TTY:, LPT:, FOO, etc.).

Next, the program requests the name of the .REL file of interest: "REL FILE=".

The results written on the output file are a tabulation of each block in the REL file, giving block type, the number of data words, (excluding relocation words), and the total number of words in that block. (See DECsystem10 Assembly Language Handbook, for explanation of block types.) For blocks declaring a program name, that name is also typed out. For blocks defining entry points, the names of the first 4 entry points are also typed out.

If a block is encountered which is entirely zeroes, it is noted as a block of type "ERR", and its length is counted but it is considered to contain no data words.

On completion of the file the total (decimal words) length of the file is typed out.

TYPREL provides a good doublecheck of FUDGE2 manipulations.

Example:

```
@TYPREL

OUTPUT FILE= TTY:

REL FILE= ANOM.REL;2

TYPE    DATA    TOTAL


4       Ø       2       ENTRY
6       1       3       NAME    ANOMAL
1       22      24      PROG
1       22      24      PROG
1       14      16      PROG
1       17      21      PROG
1       22      24      PROG
1       7       11      PROG
2       22      24      SYM
2       22      24      SYM
2       20      22      SYM
7       1       3       START   ´Ø
5       2       4       LAST    ´135
Ø       Ø       126     ERR

TOTAL FILE LENGTH (DECIMAL WORDS) = 256

DONE.
@
```

WATCH

WATCH is a program that makes continuous on-line measurements of system activity.

WATCH requests an output file where it will put its measurements.

At a specifiable interval, it measures and outputs the following information in the form of readable ASCII text:

1. The percentage of the last minute spent

    a. In user computations
    b. Idle (no requests for service)
    c. In an I-O wait for a page to be read from the disc or drum.
    d. Running the core manager (a,b,c, and d should always sum to 100%)
    e. In page faults (this time is charged to users)
    f. In each of JOBS 0 to 15

2. The number of jobs in the balance set (runnable and loaded into core) averaged over the last minute.

3. The number of the following events in the last minute

    a. Drum pages read
    b. Drum pages written
    c. Disc pages read
    d. Disc pages written
    e. Terminal wakeups (break characters typed)
    f. Terminal interrupts (^C)

WATCH is terminated by typing ^C.

EXAMPLE:

```
@WATCH
OUTPUT TO: TTY: [OK]
INTERVAL IN SECONDS: 30      (any reasonable # in seconds)

second group values? y          (either answer Y or N)
JOB BREAKDOWN? Y                (either answer Y or N)

USED IDLE IOWT CORE NCOR PURG NREM TRAP NRUN NBAL BSWT DSKW DRMW
DMRD DMWR DKRD DKWR TTIN TTOU TTBK TT^C

JOB0 JOB1 JOB2 JOB3 JOB4 JOB5 JOB6 JOB7 JOB8 JOB9
JB10 JB11 ...
 ..
 ..

62.2  0.0 33.8  3.8   8;  0.3   16  8.5  7.9  3.1  1.9  2.3 97.7
1688  620   16   50  391 1523  353    2
30.2 SEC, 21 JOBS.
11.1  0.7  0.0  0.0  8.6  2.2  0.0  0.0  4.0  0.0
 1.3  1.3  0.2  0.0  0.7  0.0 12.3  2.0  0.0  0.0
```

(on the next example, we just gave a yes on job breakdown)

```
OUTPUT TO: TTY: [OK]
INTERVAL IN SECONDS: 30

SECOND GROUP VALUES? N
JOB BREAKDOWN? Y

USED IDLE IOWT CORE NCOR PURG NREM TRAP NRUN NBAL BSWT DSKW DRMW

JOB0 JOB1 JOB2 JOB3 JOB4 JOB5 JOB6 JOB7 JOB8 JOB9
JB10 JB11 ...
 ..
 ..

60.0  0.0 33.0  3.8  109  0.3   23 10.610.9  4.5  2.4  3.2 95.7
30.3 SEC, 22 JOBS
12.0  3.0  5.4  0.0  0.2  2.6  0.0  0.0  0.0  3.0
 0.8  0.0 23.4  0.2  0.0  2.5  0.0  0.3  1.7  4.3
 0.8  0.7
^C
```

```
@
```

(on the next example, we just gave a yes on second group values)

```
OUTPUT TO: TTY: [OK]
INTERVAL IN SECONDS: 3Ø

SECOND GROUP VALUES? Y
JOB BREAKDOWN? N

USED IDLE IOWT CORE NCOR PURG NREM TRAP NRUN NBAL BSWT DSKW DRMW
DMRD DMWR DKRD DKWR TTIN TTOU TTBK TT^C

74.4  Ø.Ø 21.3  3.9   95  Ø.7   53  8.2 12.6  6.9  2.4  2.6 96.7
1844  661   23   57  531 1693  274   17
3Ø.5 SEC, 22 JOBS.
```

The following game programs in the directory are normally available to in house BBN people only; contact Tony Calleva with requests for exceptions. (e.g. ARPANET demos). See example below for using them.

```
@RUN <HACKS>CHESS

@RUN <HACKS>DOCTOR

@PUN <HACKS>JOTTO

@RUN <HACKS>LIFE

@RUN <HACKS>MAXIM
```

CHESS

CHESS is the chess-playing program developed by Richard Greenblatt, Donald Eastlake, and Stephen Crocker at M.I.T. It was described in "The Greenblatt Chess Program", (authors above), P8Ø1 - 81Ø of 1967 Fall Joint Computer Conference. The program is an honorary member of the United States Chess Federation and the Massachusetts Chess Association, under the name Mac Hack Six. In the April 1967 amateur tournament the program won the class D trophy; it wins about 8Ø% of its games against non-tournament players.

During play, the program understands moves typed in using standard chess notation, some examples of which are given below.

P-KN3	Pawn to king´s knight 3
B*P	Bishop captures pawn
O-O	Castle kingside
QR-Q1	Queen´s rook to queen 1
R/K2-Q2	Rook on king 2 to queen 2
P-R8	Promote pawn (to queen assumed)
Q*P/Q6	Queen captures pawn on queen 6
O-O-O	Castle queenside

Other commands are available for control and information:

BD	Type out board
PW	Play white
PB	Play black
PN	Play neither
PS	Play self (both sides)
M	Make next move
U	Undo last move
DRAW	Request machine to acknowledge draw
PG	Print game (history)
LIST	List commands
RESET	Overturn board (for bad sports)

EXAMPLE:

```
@CHESS
 PB

P-KB3
 B P/K2-K4
BD
WR WN WB WK WQ WB WN WR
WP WP ** WP WP WP WP WP
-- ** WP ** -- ** -- **
** -- ** -- ** -- ** --
-- ** -- BP -- ** -- **
** -- ** -- ** -- ** --
BP BP BP ** BP BP BP BP
BR BN BB BK BQ BB BN BR

P-KN4
 B Q/Q1-KR5 CHECKMATE
PG

1  P/KB2-KB3     P/K2-K4
2  P/KN2-KN4     Q/Q1-KR5
```

DOCTOR

Simulated Psychiatrist

DOCTOR is a LISP program written by Joseph Weizenbaum and described in ´ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine´, Comm. ACM, Vol. 9, #1. Jan. 1966.

DOCTOR simulates a psychiatric interview with a Rogerian psychotherapist. The ´subject´ types in some statement in natural language using normal punctuation and sentence structure. Two editing characters are provided: Control A deletes the previous character typed in (repeatable back to the previous space or comma), and # deletes the entire line typed in.

Each input statement is terminated by a period or question mark. The session is terminated by typing GOODBYE.

EXAMPLE:

```
@DOCTOR

BBN LISP-1Ø  Ø5-27-72 ...

GOOD AFTERNOON.

TELL ME YOUR PROBLEMS. PLEASE TERMINATE INPUT WITH A PERIOD OR A
QUESTION MARK.

*I HAVE A BIG PROBLEM.

I AM NOT SURE I UNDERSTAND YOU FULLY.

*MY GRANDMOTHER HAS A GRAND GZORENPLATZ.

IS IT IMPORTANT TO YOU THAT YOUR GRANDMOTHER HAS A GRAND
 GZORENPLATZ?

*IT WAS THIS MORNING?

PLEASE GO ON.

*HAVEN´T GOT THE TIME.

LETS DISCUSS FURTHER WHY YOUR GRANDMOTHER HAS A GRAND GZORENPLATZ.

*GOODBYE.

IT´S BEEN MY PLEASURE, THAT´S $9.72 PLEASE.
NIL
```

LIFE

LIFE is the Mathematical Game described in that section of Scientific American, Vol. 223, #4, October 1970. It was originated by the mathematician John Conway at Cambridge.

LIFE simulates a colony of organisms living on a 72x72 rectangular grid. Each point except for those on the edges, has 8 neighboring points, 4 horizontally and vertically and 4 diagonally. The rules of LIFE are:

1. Birth
   A new organism is created on an empty grid point if exactly 3 neighbors are adjacent to the grid point.

2. Death
   An organism dies of overcrowding if it has 4 or more neighbors.

   An organism dies of isolation if it has fewer than 2 neighbors.

Deaths and births happen simultaneously.

The program requests an initial colony pattern from the user. This is input by typing for instance,

```
 * * *
*     *
 * * *
```

using asterisks also spaces and carriage returns. Control-A will delete the previous character, Control-X deletes the line, and Control-R retypes the line. The pattern is terminated with an altmode.

Each successive generation will be typed out until one of three things happens:

1. The colony dies
2. A stable pattern is established
3. Any teletype key is pressed

At that point, the program requests another initial pattern.

FTP

Introduction

FTP (File Transfer Protocol) provides facilities for file transfer between HOSTs on the ARPA Computer Network (ARPANET). The primary function of FTP is to transfer files efficiently and reliably among HOSTs and to allow the convenient use of remote file storage capabilities. The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among HOSTs, and 4) to transfer data reliably and efficiently.

FTP Command Interpreter

Instructions to the FTP program are given via the FTP Command Interpreter. Characters typed on the user´s terminal are read by the FTP Command Interpreter and decoded as commands to perform various actions by FTP.

Typing a "?" to the FTP Command Interpreter will yield a message to use the "HELP" command to type a summary of the FTP commands.

The FTP Command Interpreter provides command completion whenever a terminator is typed (full-duplex terminals only) and an exact match is achieved with some command or a unique initial substring is typed. Terminators are space, comma, alt-mode, and carriage return. Terminators are often not distinguished and are thus equivalent. Where necessary, comma is used to separate list items, space terminates a command or option and signals the desire to specify more options, carriage return ends a command unless more information is necessary. Altmode is the same as space except that it will cause command completion in those modes where it is normally suppressed.

Escaping Back to EXEC Mode

At any time, typing a Control-C (^C) will cause FTP to stop whatever it is doing and return to the EXEC mode.

Data Transfer Functions

Data and files are transferred only via the data connection. The transfer of data is governed by FTP data transfer commands received on the FTP connections. The data transfer functions include establishing the data connection to the specified HOST (using the specified byte size) transmitting and receiving data in the specified representation type and transfer mode, handling EOR and EOF conditions, and error recovery (where applicable).

Making a Connection

There are two ways of making a connection. Typing "CONNECT host-name" or "CONNECT octal-number" will cause a connection attempt to be made. If successful, the connection will be said to be complete. If unsuccessful, the connection will be said to be incomplete with a reason given.

Disconnecting

There are three disconnect commands. "DISCONNECT" disconnects the user from the remote host without returning to the EXEC, "BYE" is the same as "DISCONNECT", and "QUIT" returns the user to EXEC without closing the connection. Thus, to close the connection and return to the EXEC, the user should type the "DISCONNECT" command followed by the "QUIT" command.

In the event that the network connections are severed by a network failure, the user will receive a message that the network has been severed and/or that the data transfer is incomplete.

FTP Command Summary

Access Control Commands

CONNECT host-name _or_ octal-number

Performs ICP to connect to the indicated host. This must be the first command issued.

LOGIN username optional-password optional-account

User identification that is required by the server for access to its file system.

username
: The argument field is an ASCII string identifying the user. Special characters may be quoted by ^V.

optional-password

The argument field is an ASCII string identifying the user´s password and may be optional. This field must be immediately preceded by the username field. The typeout of this field will either be "masked" or suppressed. Special characters may be quoted by ^V.

optional-account

This argument field is optional and is a number or ASCII string identifying the user´s account.

CWD anothername

CWD is used to change the working directory to anothername. Using it requires the "QUOTE" command at present. (Example: *QUOTE CWD anothername)

ACCOUNT number *or* string

The argument field is a number or ASCII string identifying the user´s account.

DISCONNECT

Disconnects user from remote host *without* returning to EXEC.

BYE

Same as DISCONNECT.

QUIT

Returns user to EXEC *without* closing connection.

Transfer Parameter Commands

BYTE size-of-data-connection

The argument is an ASCII-represented decimal integer specifying the byte size for the data connection. The size must be 36, 32 or 8 bit bytes.

TYPE data-type

The argument is a single ASCII character code specifying the representation types.

The following codes are assigned for type:

A - ASCII

The data is transferred in ASCII form. The transfer byte size must be 8 bits. This type would be used for transfer of text files.

L - Local Byte
The manner in which data is to be transformed depends on the byte size for data transfer. This type is identical to the Image type for byte size which are integral multiples of or factors of the computer word length.

I - Image
on output the data is transformed from contiguous bits to bytes for transfer. On input, the data is transformed from bytes into bits, storing them contiguously independent of the byte size chosen for data transfer.

The following codes are <u>not</u> yet implemented:

E - EBCDIC
The data is transferred using the EBCDIC character code and 8-bit transfer byte size.

IMAGE

Declares IMAGE file type.

TENEX

Shorthand for IMAGE, BYTE 36.

ASCII

Shorthand for TYPE A, BYTE 8.

FORM format

The argument is a single ASCII character code specifying the format.

The following codes are assigned for format:

U - Unformatted
The representation type as specified is unaffected by any format transformations.

The following codes are <u>not</u> yet implemented:

P - Printfile
Data is transferred as either ASCII or EBCDIC type

in accordance with ASA (Fortran) vertical format control statements. The data is to be transferred in 8-bit bytes.

STRUCTURE structure-of-data

The argument is a single ASCII character code specifying file structure.

The following codes are assigned for structure-of-data:

F - File

The following codes are not yet implemented:

R - Record

MODE transmission-mode

The argument is a single ASCII character code specifying the data transfer rate modes.

The following codes are assigned for transfer modes:

S - Stream
The file is transmitted as a stream of bytes of the specified byte size. The EOF is signalled by closing the data connection. Any representation type and byte size may be used in the stream mode with file structure.

The following codes are not yet implemented:

B - Block
The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines last file block (EOF), last record block (EOR), restart marker, or suspect data. Record structures are allowed in this mode, and any representation type or byte size may be used.

T - Text
The file is ASCII text transmitted as a sequence of 8-bit bytes in the ASCII representation type, and optional Printfile format. Record structures

are allowed in this mode. The EOR and EOF are defined by the presence of special "TELNET-control" codes (most significant bit set to one) in the data stream. The EOR code is 192 (octal 300, hex C0). The EOF code is 193 (octal 301, hex C1). The byte size for transfer is 8 bits.

H - Hasp
The file is transmitted as a sequence of 8-bit bytes in the standard Hasp-compressed data format. This mode achieves considerable compression of data for print files. Record structures are allowed in the Hasp mode.

FTP Service Commands

GET REMOTE-FILE to LOCAL-FILE

The REMOTE-FILE is transferred from the host site to the user´s site and is given the LOCAL-FILE name. Note: REMOTE-FILE and LOCAL-FILE use standard filename formats.

SEND LOCAL-FILE to REMOTE-FILE

The LOCAL-FILE is transferred from the user´s site to the host site and is given the REMOTE-FILE name. Note: LOCAL-FILE and REMOTE-FILE use standard filename formats.

MULTIPLE GET/SEND

Only TENEX sites permit the use of "*" in the specification of filenames for GET and SEND. The standard "*" formats must be used.

APPEND LOCAL-FILE to REMOTE-FILE

The LOCAL-FILE is transferred from the user´s site to the host site and appended to the REMOTE-FILE at the host site. Note: LOCAL-FILE and REMOTE-FILE use standard filename formats.

RENAME REMOTE-FILE to be NEW-REMOTE-FILE

The filename for REMOTE-FILE is changed to NEW-REMOTE-FILE. Note: REMOTE-FILE and NEW-REMOTE-FILE use standard filename formats.

DELETE REMOTE-FILE

This command deletes the REMOTE-FILE. Before the file is actually deleted, the user is asked "Do you really want to delete? (Y or N)". Note: REMOTE-FILE uses standard filename formats.

DIRECTORY of REMOTE-FILE-GROUP

A list of the files in the REMOTE-FILE-GROUP will be typed out. (e.g., <SMITH>*.MAC, if remote site is a TENEX site)

STATUS of remote-system

Status information about the remote-system will be typed out.

MAIL <FILE> to REMOTE-USER

Sends LOCAL-FILE to mailbox at remote site.

HELP

Types summary of FTP commands.

Miscellaneous Commands

VERBOSE

Types out all comments in long form.

BRIEF

Types out all comments in short form.

QUOTE arbitrary-FTP-line

Sends arbitrary-FTP-line to remote-site without interpretation.

STATISTICS

Turns on typeout of timing statistics.

NOSTATISTICS

Turns off typeout of timing statistics.

FTP Control Characters

Control-G

Type BELL (^G) to abort a file transfer and return to command level.

```
# Control-O
#
#      Type ^O to clear typeout buffer.
#
# Control-V
#
#      Use ^V to quote characters in LOGIN.
#
#                        Example of FTP Use
#
#      @FTP                               ;call in the subsystem
#      *BBN                               ;connect to host BBN
#      *LOG SMITH SECRET 12345            ;declare name, password,
#                                         ;account
#                                         ; the password will not be
#                                         ; echoed.
#      *DIR *.MAC
#      (to local file) TTY: [confirm]     ;get a partial directory
#                                         ; listing
#      *GET PROGRAM.MAC                   ;must end with carriage
#                                         ; return
#      (to local file> <esc>PROGRAM.MAC   ;escape causes same name
#                                         ; to be used
#      *DISCON                            ;break the network
#                                         ; connections
#      *QUIT
#      @
```

INDEX

Commands are given in capital letters.

HOSTAT

HOSTAT obtains the network site status information maintained by the network survey site (MIT-DMS as of Dec. 1, 1974). It then types this out in columnated form, grouped by status. If the site name of a particular site is not known to the local TENEX, the octal site address will be given instead.

The relevant status categories are:

Logging:	The survey site was able to complete the ICP to this site´s logger socket. (code 5)
Refusing:	This site responded to an RFC to the logging socket, but did not complete the ICP. (code 4)
Not Responding:	This site did not respond to an RFC to the logging socket. (code 3)
No NCP:	This site´s NCP did not respond at all. (code 2)
Dead:	This site is disconnected from the network. (code 1)
TIPs up:	These sites are TIPs which responded to a probe from the survey site. (code 4)
TIPs Down:	These sites are TIPs which are disconnected from the network. (code 1)
Status Code Unknown:	The status code given by the survey site is not known.
Unable to Poll:	The survey site was unable to poll this site successfully. (code 7)

For more information about the survey service and codes see RFC 530 (A report on Survey Project by Abhay Bushan) and the subsequent note on this RFC (NIC Journal 20248).

If the survey site is down, HOSTAT will call NETSTAT to provide the status information in the local TENEX´s internal tables. Note that this information is updated somewhat randomly (ie. only when interactions occur or are attempted with the particular site involved).

To use HOSTAT:

```
@HOSTAT
Getting survey from MIT-DMS... OK.
Survey of 29-NOV-73 1:57PM-EST
```

```
#  Logging:
# SRI-ARC  CASE-1Ø USC-44  UCLA-CCN LL-TX-2 USC-ISI   MIT-AI
# RAND-RCC AMES-67 UCSD-CC MIT-DMS  CMU-1ØA UCLA-CCBS MIT-ML
# SU-AI
#
#  Not responding:
# LL-67 PARC-MAXC
#
#  Dead:
# UCLA-NMC    MIT-MULTICS CMU-1ØB  CCA-TENEX SRI-AI    BBN-TENEXB
# UCSB-MOD75 SDC-LAB     I4-TENEX UKICS-36Ø BBN-TENEX 243
# UTAH-1Ø    HARV-1Ø
#
#  TIPs Up:
# UTAH-TIP  RADC-TIP USC-TIP  SDAC-TIP   CCA-TIP  NCC-TIP
# AMES-TIP  NBS-TIP  GWC-TIP  ARPA-TIP   FNWC-TIP NORSAR-TIP
# MITRE-TIP ETAC-TIP DOCB-TIP BBN-TESTIP RML-TIP  UKICS-TIP
```

NETED, AN ARPA NETWORK COMMON EDITOR

NETED is a context editor which will present "the same" user interface on many ARPA Network Hosts. The design is a moderate extension of an extremely scaled-down editor derived from the "ed" family of editors on CTSS (which was one of the very first general purpose time-sharing systems). No claims are advanced for its elegance or modernity; it is meant to be "a" common editor for the Network, not "the" common editor -- but the underlying design has stood the test of time. It was designed and initially implemented by volunteers from the Network User Interest Group (USING), who stand ready to supply both source and object code to all Hosts. The major needs which it is intended to satisfy are those of learnability and wide-spread applicability. That is, it is a deliberately "un-fancy" editor, so that new users can pick it up quickly; and it is meant to be available on as many Network Hosts as possible, so that a single investment in learning an editor will have a large payoff to those who use more than one Host.

Although it is believed that the user interface is as similar as possible from implementation to implementation, there are two levels on which differences will be found. The more substantive level is that of unavoidable system differences, such as the availability or unavailability of "type-ahead", the specification of characters for editing within a line, limits on line length and file size, availability of both alphabetic cases, and the like. On this level, the implementers´ intent was to allow the user to create a file which "makes sense" to the system on which it is being created, rather than to attempt to constrain ourselves to an unusable but identical "virtual" file format. So there will be some unstated assumptions which differ from system to system, which users will have to pick up as they go along -- but we have tried hard to keep them to a minimum.

The second level of differences is "cosmetic". That is, we have not striven to make the form of all messages from the editor identical; but we have striven to make their sense identical. Because implementation strategies differ, and because the implementers were volunteers, it did not seem appropriate to attempt to "lock-step" in this area. An attempt has been made to keep "normal" responses close enough to allow for the possibility of invoking the editor from a program, but "abnormal" responses have not been given the same attention.

If, in actual practice, cosmetic variations prove to be a major nuisance to users -- or if users wish to offer comments on other aspects of NETED -- the USING NETED committee can be reached by Network mail through the Network Information Center´s "Journal" mechanism (address: yourident/neted), or through the Network

# Feedback mechanism. Also, the Network Technical Liaisons at
# Hosts running NETED implementations should be able to put users
# in touch with the committee.
#
# (A final introductory point: on systems which do permit
# "type-ahead" the policy is to ignore any pending requests when an
# abnormal condition is encountered while processing the "current"
# request. This strategy is employed in order to prevent the
# possibility of erroneously processing a request which was in some
# sense contingent upon the one which did not complete normally.
# Taking advantage of type-ahead, then, requires getting the "feel"
# of the particular implementation being used. The editor is,
# however, quite usable even without type-ahead -- and, as noted
# above, not all implementations even offer it. Note also that
# "prompting" is available, so that the user can be certain the
# previous request is complete before typing a new one.)
#
# "Modes"
# As is typical of "context editors", the NETED command is used
# both for creating new files and for altering already existing
# files -- where "files" are named collections of character encoded
# data in the storage hierarchy of a time-sharing system.
# Consequently, NETED operates in two distinct "modes" -- called
# "input mode" and "edit mode".
#
# When NETED is used to create a file (that is, when it is invoked
# from command level with an argument which specifies the name of a
# file which does not already exist in the user´s "working
# directory"), it is automatically in input mode. It will announce
# this fact by outputting a message along the lines of "File
# soandso not found. Input." Until you take explicit action to
# leave input mode, everything you type will go into the specified
# file. (Actually, it goes into a "working copy" of the file, and
# into the real file only when you indicate a desire to have that
# happen.) Lines consisting of only a new-line are permitted.
# Blanks at the end of input lines are handled differently by
# different operating systems, and there is no general policy in
# NETED on this topic (that is, some systems strip trailing blanks
# "automatically" in their input/output subsystems -- before a
# NETED implementation sees them). To leave input mode, type a
# line consisting of only a period and the appropriate new-line
# character: ".<NL>", where <NL> is whatever it takes to cause a
# Telnet New-Line to be generated from your terminal.
#
# After leaving input mode, you are in edit mode. Here, you may
# issue various "requests" which will allow you to alter the
# contents of the (working) file, re-enter input mode if you wish,
# and eventually cause the file to be stored. Note that edit mode
# is entered automatically if the argument you supplied to NETED
# specified an existing file.

```
# Regardless of how it was entered, being in edit mode is confirmed
# by NETED´s outputting a message of the form "Edit." Editing is
# performed relative to a (conceptual) pointer which specifies the
# current line, and many requests pertain to either moving the
# pointer or changing the contents of the current line. (When edit
# mode is entered from input mode, the pointer is at the last line
# input; when entered from command level, the pointer is at the
# "top" of the file.)
#
# (Note that in the examples which follow although the command´s
# name and the requests are shown in lower case for typing
# convenience, upper case also works; indeed, on many systems only
# upper case is available.)
#
# Invocation of the Command
# Some time-sharing systems do not easily allow the passing of
# arguments (or "parameters") to a program being invoked as a
# command, while to others this is a natural approach. Because
# NETED is specifically intended to be usable on all ARPA Network
# Server Hosts, it has been given two methods of invocation in
# light of these two styles of argument treatment:
#      1) neted
#      2) neted filename
#
# Method 1) will work for all NETED implementations; method 2)
# actually works for most implementations, but not all. Therefore,
# it is advisable to invoke NETED on a system new to you via method
# 1) the first time you use it, and by whichever method you prefer
# of the ones available subsequently. Note that in method 1), the
# command will immediately ask you to enter a file name. As
# indicated above, you will be in edit mode if the specified file
# exists, and in input mode if not. A message will be output
# indicating which mode has been entered.
#
# Requests
# NETED´S edit mode requests follow, in an order intended to be
# helpful. Two important reminders: the requests may only be
# issued from edit mode, and each one "is a line" (i.e., terminates
# in a newline / carriage return / linefeed as appropriate to the
# User Telnet being employed). Syntax Note: if the request takes
# an argument, there must be at least one space (blank) between the
# request´s name and the argument. In certain of the requests (in
# which the argument is a string of characters to be located or
# inserted) leading blanks "in" the argument may be desirable;
# thus, in the "l", "i", and "r" requests if more than one blank
# has been used to separate the request name from its argument, the
# additional blanks are significant, while in the rest of the
# requests which take arguments such leading blanks are permitted
# but ignored (see also Examples, below).
#
```

```
# 1. n m
# For unsigned m, the n(ext) request causes the pointer to be moved
# "down" m lines. If m is negative, the pointer is moved "up" m
# lines. If m is not specified, the pointer is moved one line.
# The line is output unless printing has been suppressed by the "v"
# request (14.). If the end of the file is reached, an "End of
# file reached by n m" message is output by NETED, and the pointer
# is left "after" the last line.
#
# 2. l string
# The l(ocate) request causes the pointer to be moved to the next
# line containing the character string string (which may contain
# leading and/or internal blanks); the line is output, unless
# printing is suppressed. If no match is found, a message of the
# form "Top of file reached by l string" will be output (and the
# pointer will have returned to the top of the file). The search
# will not wrap around the end of the file; however, if the string
# was above the starting position of the pointer, a repetition of
# the locate request will find it, in view of the fact that the
# pointer has been moved to the top of the file. To find any
# occurrence of the string -- rather than the next occurrence -- it
# is necessary to move the pointer to the top of the file before
# doing the locate (see following request).
#
# 3. t
# Move the pointer to the top of the file (actually to a position
# "above" the current first line of the contents).
#
# 4. b
# Move the pointer to the bottom of the file and enter input mode.
# "Input." will be printed when the request has completed.
#
# 5. .
# Leave the pointer where it is and enter input mode. (First new
# line goes after current old line.) "Input." will be printed when
# the request has completed. Note that you remain in input mode
# until a line consisting of only ".<NL>" is given, as discussed
# above.
#
# 6. i string
# The i(nsert) request causes a line consisting of string (which
# may contain leading and/or internal blanks) to be inserted after
# the current line. The pointer is moved to the new line. Edit
# mode is not left. If no string is given, a blank line will be
# inserted into the working file.
#
# 7. r string
# The r(eplace) request causes a line consisting of string (which
# may contain leading and/or internal blanks) to replace the
# current line.
```

# 8. p m
# The p(rint) request causes the current line and the succeeding m
-
1 lines to be output. If m is not specified, only the current
line will be output. End of file considerations are the same as
with "n". The pointer is moved to the final line printed.
#
9. c /s1/s2/ m g
The c(hange) request is quite powerful, although perhaps a bit
complex to new users. (Several examples of its use are given
below.) In the line being pointed at, the string of characters
s1 is replaced by the string of characters s2. If s1 is void, s2
will be inserted at the beginning of the line; if s2 is void, s1
will be deleted from the line. Any printing character not
appearing within either character string may be used in place of
the slash (/) as a delimiter. If a number, m, is present, the
request will affect m lines, starting with the one being pointed
at. All lines in which a change was made are output, unless
printing has been suppressed by the "v" request. The pointer is
left at the last line scanned. If the letter "g" is absent
(after the final delimiter) only the first occurrence of s1
within a line will be changed. If "g" (for "global") is present,
all occurrences of s1 within a line will be changed. (If s1 is
void, "g" has no effect.) Note well: blanks in both strings are
significant and must be counted exactly. End of file
considerations are the same as with "n".
#
10. d m
The d(elete) request causes m lines, including the current one,
to be deleted from the working copy of the file. If m is not
specified, only the current line is deleted. The pointer is
moved to the immediately previous undeleted line (that is, it is
moved "up"), except for the case where all the lines in the file
have been deleted, in which case the pointer will have been
backed up to the top of the file.
#
11. w filename
Write out the working copy into the storage hierarchy under the
name filename if this argument is present, or with the original
name if the argument is not given, and remain in NETED. See also
discussion of "quit" request (18.). A message of the form
"filename written." is output when the request has completed.
Pointer position is preserved.
#
12. save
Write out the working copy into the storage hierarchy and exit
from NETED.
#
13. v
The v(erify) request reverses the setting of the internal
variable which governs printing of lines reached or affected by

# several requests ("c", "l", and "n" -- but not "p"). It is
# typically employed to suppress the large quantities of output
# which result from multi-line changes, although experienced users
# sometimes employ it merely to avoid waiting for responses when
# they are confident they know what they´re doing. Default is "on"
# (i.e., such lines will be printed unless the "v" request is
# given, and will again be printed if the "v" request is given
# again).
#
14. *
# Reverse the setting of the internal variable which governs
# printing of "*" as a "prompt" when ready for a new request or
# line of input. Because the editor is expected to be used heavily
# by new users, the default for this variable is "on" in order to
# offer initial reassurance that the editor "is there". That is,
# prompting will occur unless the "*" request is given, and will be
# resumed if the request is given again.
#
# 15. m filename
# The m(erge) request causes a copy of the contents of the already
# existing file designated by filename to be inserted into the
# working file, beginning immediately after the current pointer
# position. The pointer will be moved to the final line of the
# inserted material, and a message of the form "filename merged."
# will be output, when the request has completed.
#
# 16. h
# The h(elp) request actuates a per-implementation help mechanism,
# which ranges from a simple statement that the present
# implementation has no known deviations from the standard (as
# expressed in this document), through a list of local extensions
# and (possibly) idiosyncrasies, to a full-blown tutorial on the
# use of NETED. Each implementation will include an announcement
# of its "erase" and "kill" characters (which, respectively, cause
# one or more immediately previous characters or "print positions"
# within a line to be erased, or cause the whole line up to that
# point to be discarded); see also Note c), below.
#
17. ?
# This request causes a list of the available requests to be
# output.
#
# 18. quit
# The quit request causes immediate exit from NETED; no writing out
# of files occurs as a result of this request, although any
# previous writing is unaffected by it. (If, for example, you wish
# to alter a given file but preserve a copy of the original intact,
# you would edit it, write the new version out under another name,
# then "quit". To change the original, on the other hand, you
# would exit either by "save" or by the sequence "w", "quit".)
# N.B., work can be lost by improper use of "quit".

# Notes
a) The requirement that "save" and "quit" be fully spelled out
# is based on a concern that these two particularly powerful
# functions not be invoked by the accidental mistyping of a single
# letter.
#
b) At both the "top" and the "end" of the file, the editor is
dealing with a state rather than with a line. Messages of the
form "Top of file reached by:" and "End of file reached by:"
convey this idea when requests cause the editor to reach these
states. It is specifically declared to be an invalid operation
to attempt to r(eplace) or c(hange) when in such a state, and
implementations may vary in their responses to p(rint). (The
intended responses are "<Top of file.>" and "<End of file.>", but
by some implementation strategies the "reached by" type messages
or explicit error messages may be output instead.) In actual
practice, matters are less confusing than the above might seem to
indicate, for the only operation which one typically desires to
perform when at the top of the file is an insertion of some sort,
which works. (A change to the first line of the file´s contents
is effected by "t","n", "c...".)
#
c) There are no defined "erase" and "kill" characters in NETED,
for two reasons: the user has available to him the editing
characters of his User Telnet (and/or the generic ones of the
Telnet Protocol) before transmission and the Server´s after
transmission, so there is no real need for such a facility; and
the proliferation of special purpose characters on a per-command
basis is generally accounted an undesirable design practice.
(Use the "h" request to determine a given implementation´s "line
editing" characters.)
#
d) On some systems, the system itself will emit a prompt
character whenever a call is made to read from the user´s
terminal; the "*" request cannot, of course, turn off prompting
# at that level.
#
e) By convention, per-implementation requests´ names will begin
with an "x". (Use the "h" request to determine whether a given
implementation offers any such requests.)
#
Examples
In the following, the prompting "*´s" have been omitted because
they lead to considerable clutter and make the examples much
harder to read and follow. Please note that in actual use, the
editor will output a "*" when it is ready for a line of input (in
both edit mode and input mode). Those who prefer not to be
prompted in this fashion can turn prompting off with the "*"
request. (Remember that if you are creating a new file you must
shift to edit mode first. It is felt by the designers that this
slight inconvenience is offset by the greater convenience

inexperienced users are afforded by having prompting be "on" by default, but we are sorry that the binary choice forces us to inconvenience anyone.)

1. Input and edit modes
Assuming that there is no file named "sample" in your directory, the command

```
          neted sample
```

would cause the response

```
          File not found.
          Input.
```

Typing the following

```
          This is line 1.
          This is line 2.
          This is line 3.
          .
```

would cause the three lines of text to be placed in the working copy of the file, and generate the response (because of the mode change request ".")

```
          Edit.
```

The following sequence would write a copy of the working copy out, move the conceptual pointer to the top of the file, insert a line there, then re-enter input mode at the bottom of the file:

```
          w
          sample written.                        (response)
          t
          i This is line 0.

          b
```

(Response after the "b" request is "Input.".) Now we add two lines at the bottom and return to edit mode:

```
          This is line 4.
          This is line 5.
          .
```

(Response is "Edit.") At this point,

```
          save
```

will write out the (six-line) file and return to command level. Note that had it been desired to input more than one line at the top of the file (or elsewhere in the file) the "." request could have been used conveniently to enter input mode.

2. Pointer-moving requests
Continuing with the file "sample", the following would leave the pointer at the final line:

```
#          neted sample
#          Edit.                 (response)
#          n 6
#
# Note  that the argument to the "n" request is "6" rather than "5"
# because the top of the file is a null line rather than the  first
# line  of  the  contents,  in order to facilitate insertion of new
# material at the top of the file.  (If you had done  an  immediate
# "p"  request  after  entering  edit  mode from command level, the
# response would have been "<Top of file.>")  An alternate  way  of
# moving the pointer to the last line (instead of "n 6") is
#          l 5
#          This is line 5.                 (response)
#
# This  latter  method,  usually known as "locating by context," is
# the more common.  At this point,
#          n -2
# would cause the response
#          This is line 3.
#
# As noted above, "t" moves the pointer to the top of the file, and
# "b" moves it to the bottom (and enters input mode).
#
# 3. Changing existing lines
# Assume the pointer is still located at "This is line 3."
#          c /is/was/
#
# would result in
#          Thwas is line 3.
#
# Ah well.  Blanks are significant.  To fix the mess  and  do  what
# was intended:
#          c /was/is/
#          This is line 3.                 (response)
#          c / is/ was/
#          This was line 3.                (response)
#
# To change all instances of a character string on a given line:
#          c /i/x/ g
#          Thxs was lxne 3.                (response)
#
# (Note  the  space  before the "g".)  An easy way to fix that line
# would be
#          r This is line 3.
#
# which simply replaces the current line.  ("c /x/i/ g" would  also
# work, of course.)
#
# The  following  request  (the  pointer  is not changed by the "r"
# request)
#          c /line/entry/ 2
# would result in the response
#          This is entry 3.
#          This is entry 4.
#
# with the pointer now at "This is entry 4."
#
# To append to the beginning of a line,
#          c //tag:/
#          tag:This is entry 4.                    (response)
#
# And to remove a string from a line,
#          c /tag://
#          This is entry 4.              (response)
#
# Note that "/" need not be used as the delimiter.  I.e., "c
# xtag:xx" would also have worked in the last instance.
#
# 4. Miscellaneous requests
# Still using "sample" consider the following:
#          t
#          n
#          This is line 0.               (response)
#          d 2
#          Top of file reached by "d 2"            (response)
#          i      This is the beginning.
#          c /in/inn/
#               This is the beginning.             (response)
#          l 3
#          This is entry 3.              (response)
#          d 99
#          End of file reached by "d 99"           (response)
#          .
#          Input.               (response)
#               This is the end.
#          .
#          Edit.                (response)
#          t
#          p 99
#          <Top of file.>                (response)
#               This is the beginning.             (response)
#          This is line 2.               (response)
#               This is the end.                   (response)
#          End of file reached by "p 99" (response)
#
```

Note that the first "d" request took care of the lines ending
with "0." and "1." and the second took care of "3." through "5."
The response to the first illustrates what happens to the pointer
after a "d" request. The insertion of "This is the beginning."
shows the handling of leading blanks in edit mode; the insertion
of "This is the end." shows the handling of leading blanks in
input mode. The "." after the "d 99" could also have been a "b"

# or an "i" request. A "save" request at this point would leave
# you with a file containing only the three text lines which were
# printed in response to the "p 99".
#
# No attempt has been made here to offer examples of all possible
# requests, in the belief that the foregoing examples convey the
# "feel" of the editor and the descriptions of the other individual
# requests should be sufficient for the user to learn from, once
# the overall mechanism has been learned. Special note should be
# taken of the "h" request, however, as in many implementations the
# response to it will furnish important additional information.
#

NETSTAT

To obtain information on the status of the network as seen from TENEX:

```
        @NETSTAT%
```

NETSTAT responds with:

```
        *
```

and awaits a command.

(Note: Throughout this description of NETSTAT "%" indicates where carriage return is typed.)

There are two types of information which can be requested:

# 1) A list of the network sites which the local
# host considers to be up.
2) A list of network connections with the local host.

The following commands select the type of information to be given:

*ALL	All sites and all connections.
*CONNECTIONS	All connections.
*HOSTS	All hosts (i.e. not TIPS).
*SITES	All sites.
*SPECIFIC CONNECTIONS	Only connections of a specified type.

A blank line terminates commands and starts the type out.

If no command has been given then ALL is assumed.
eg:

```
        @NETSTAT%
        *%
```

Will list all sites and all connections.

SITES:

The site names are typed in tabular form with (octal) site addresses given for sites whose names are not known to TENEX.
eg:

```
@NETSTAT%

*SITES %
*%


THE FOLLOWING  ARE UP:

UCLA-NMC    BBN-TENEXB SU-AI    MITRE-TIP USC-ISI DOCB-TIP
UCLA-CCN    MIT-DMS   ILL-ANTS RADC-TIP  Ø27     SAAC-TIP
SRI-ARC     MIT-AI     I4-TENEX NBS-TIP   USC-TIP ARPA-TIP
UCSB-MOD75 LL-67       AMES-67  ETAC-TIP  GWC-TIP Ø35
BBN-TENEX  LL-TX-2     AMES-TIP
```

For SITES it is possible to specify that only sites of a specific type (or types) are of interest.
The possible types are:

TENEX	PDP-1Ø TENEX systems
ITS	Incompatible Timesharing Systems
DEC1Ø	DEC PDP-1Ø systems
TIP	Terminal Interface Processors
MTIP	Magnetic tape TIPs

eg:

```
*SITES TENEX %
*%


THE FOLLOWING  ARE UP:

BBN-TENEX BBN-TENEXB I4-TENEX USC-ISI
```

or:

```
*SITES TIP ITS %
*%


THE FOLLOWING  ARE UP:
```

```
MIT-DMS AMES-TIP  RADC-TIP ETAC-TIP GWC-TIP  SAAC-TIP ARPA-TIP
MIT-AI   MITRE-TIP NBS-TIP  USC-TIP  DOCB-TIP
```

<u>HOSTS</u>:

The HOSTS command is the same as the SITES command except that if no site types are specified then only sites which are not TIPs or MTIPs are typed.

<u>DECIMAL.HOST.NUMBERS:</u>

Causes decimal site address numbers to be typed with the site names for the SITES or HOSTS commands.

<u>OCTAL.HOST.NUMBERS:</u>

Causes octal site address numbers to be typed with the site names for the SITES or HOSTS commands.

<u>NO.HOST.NUMBERS:</u>

Causes no site address numbers to be typed with the site names for the SITES or HOSTS commands. This is the default case.

CONNECTIONS:

The connection information is given in the following tabular form:

```
@NETSTAT%

*CONNECTIONS
*%


ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST       4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

  0 OPND 30324600305 BBN-TENEX   30324000142   5         944      6  B 8
  5 LSNG          21
  6 LSNG           1
 13 OPND 30324000146 BBN-TENEX   30325000301   7         952      6 T106
 20 OPND 30325000300 BBN-TENEX   30324000147  10       17728      2  B 8
 21 OPND 30324000147 BBN-TENEX   30325000300  10       17696      2 T106
 22 LSNG           3
         .               .               .
          .               .               .
           .               .               .
111 OPND 30324000070 NBS-TIP         4200003   2         928      6 T101
116 OPND 30324000151 NBS-TIP         1600002  11         640      1 T102
117 OPND 30324000071 NBS-TIP         4200002  23          64      1 T101
121 LSNG          15
130 OPND    33200101 BBN-TENEX      33200200   2           0      0  B36
134 RFN1 30324000153 SAAC-TIP         200002   3           8      0  B 8
135 LSNG          17
136 OPND    33200200 BBN-TENEX      33200101   2           0      0  B36
```

where:	I	is the index of the connection in local system tables.
	STATE	is the state of the connection.
	LCL-SOCKET	is the absolute local socket number.
	HOST	is the name of the foreign host for this connection (or its address in octal if its name is not known to TENEX).
	4N-SOCKET	is the absolute foreign socket number.
	LNK	is the link of the connection.

BITS-ALLOC	is the present number of bits (decimal) allocated for this connection.
M-ALL	is the present number of messages allocated for this connection.
BS/VT	is either the byte size for this connection or the virtual terminal number of the connection (assumed byte size of 8).

For LSNG connections only I, STATE, and LCL-SOCKET are given.

SOCKETS:

The SOCKETS command is the same as the CONNECTIONS command.

SPECIFIC CONNECTIONS:

It is possible to specify that only connections of specific types are of interest. This is done by typing:

```
*SPECIFIC CONNECTIONS:
**
```

NETSTAT will now accept specification commands.

The possible specifications are:

SITES or
HOSTS to specify only connections with specific hosts or types of hosts. Names may be either the official network names or nicknames known to the local system. Host numbers are in octal.
eg:

```
**HOSTS UCLA-CCN 201 TENEX %
```

JOBS to specify only connections whose local socket numbers are relative to specific jobs. Job numbers are in decimal.
eg:

```
**JOBS 11,0%
```

SIZES to specify only connections which have given byte sizes. Byte sizes are in decimal.

eg:

```
**SIZES 8,32%
```

STATES to specify only connections which are in given states.
eg:

```
**STATES LSNG CLSW %
```

TTYS to specify only connections which are with specific virtual terminals. Terminal numbers are in octal.
eg:

```
**TTYS 101,105%
```

If no arguments are given for TTYS then all connections with virtual terminals are specified.
eg:

```
**TTYS %
```

USERS to specify only connections whose local socket numbers are relative to specific users.
eg:

```
**USERS JSMITH JDOE%
```

# Specification commands to select connections involving specific
# sockets are:
#
LOCAL.SOCKETS or
SOCKETS to select connections with specific local socket
numbers
#
FOREIGN.SOCKETS to select connections with specific foreign socket
numbers.
#
The specification arguments for the SOCKETS commands can be any
# of the following (all numbers are octal):
#

```
# SKT          A single specific socket:  S = SKT
# SKT1-SKT2    Any socket in range:  SKT1 <= S <= SKT2
# SKT+INCR     Any socket in range:  SKT <= S <= SKT+INCR
# <SKT         Any socket in range:  S <= SKT
# >SKT         Any socket in range:  SKT <= S
```

The arguments for all of the above selection commands may be separated by spaces, commas, or, for those arguments where altmode recognition may be done, by the altmode used in completing the previous argument. Altmode may be used to recognize all arguments which are not numbers, as well as the selection commands themselves. Only enough need be typed to uniquely identify the command or argument.

eg. The following are all equivalent and legal (ALTMODE is represented by "$"):

```
**ST$ATES LSNG,CLSW OPND%

**STATES L$SNG CLSW,O$PND %

**ST LS,CLS$W OP%
```

It is also possible to precede any of the above specification commands with the word "NOT". This has the effect of specifying all connections except those of the given type(s).

eg:

```
**NOT STATES LSNG OPND %
```

will specify all connections which are not in the LSNG or OPND states.

If a particular specification command has already been given without NOT, it is an error to later give the same command with NOT.

eg:

```
**HOSTS TENEX %
**NOT HOSTS ??
**
```

When more than one specification command is given, only the connections that satisfy all of the given specifications (ie. the logical AND of the conditions specified) are listed.

To terminate specifications and start typeout (or return to regular commands if no specifications were given) type carriage return on a blank line.

ie:

```
**%
```

REPEAT:

NETSTAT normally returns to EXEC when it completes its type out. This command causes NETSTAT to return to accept a new set of commands after each type out. Typing a blank line at that point will cause NETSTAT to again give the status information specified by the last set of commands.

QUIT:

Forces NETSTAT to return to EXEC.

BRIEF:

Causes no headings or messages to be typed when giving status information.

VERBOSE:

Causes headings to be always typed.
The default case is to print the heading for the connection information only the first time.

SPECIAL CHARACTERS:

^A (ctrl A) may be used to delete the last character typed.

? will prompt a help message.

RUBOUT will abort a line.

Besides causing recognition, ALTMODE will also prompt a description of the type of argument required when typed as the first character when an argument is expected.

^O (ctrl O) will stop the typeout. It acts independently on the site and the connection information.

: semi-colon can be used at the start of a line to indicate a line to be ignored.

Examples:

```
@NETSTAT%

*SPECIFIC CONNECTIONS:
**SIZES 8%
**NOT TTYS %
**%


ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST        4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

 15 OPND 30327000325 SRI-ARC      30324100024  27         184     19  B 8
 16 OPND 30327000305 BBN-TENEX    30324000140  5          952      6  B 8
 17 OPND 30327000324 SRI-ARC      30324100025  7        18168      2  B 8
 23 OPND 30327000304 BBN-TENEX    30324000141  6        18168      2  B 8
 25 OPND 30324000240 ETAC-TIP          400003  2            0      0  B 8
 33 OPND 30324000241 ETAC-TIP          400002  4            0      0  B 8
 45 OPND 30327000301 BBN-TENEX    30324000074  2          944      6  B 8
 50 OPND 30325500003 CMU-10B              400  62         392     19  B 8
 55 OPND 30327000300 BBN-TENEX    30324000075  3        18168      2  B 8
 57 OPND 30325500002 CMU-10B              401  3        17728      2  B 8
131 CLZW 30327000311 I4-TENEX     30324000056  36          48 262142  B 8
136 CLZW 30327000310 I4-TENEX     30324000057  3        18168      2  B 8


@NETSTAT%

*SPECIFIC CONNECTIONS:
**NOT HOSTS TIP %
**TTYS %
**%


ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST        4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

 13 OPND 30324000166 SRI-ARC      30326500003   2         952      6 T102
 21 OPND 30324000167 SRI-ARC      30326500002  17       12976     15 T102
 27 OPND 30324000220 SRI-ARC      30326500005   3         952      6 T103
 35 OPND 30324000221 SRI-ARC      30326500004  11       15832     17 T103
 64 OPND 30324000075 BBN-TENEX    30327000300   3       18168      2 T114
 65 CLZW 30324000315 CASE-10      30324600002  20       15736      6 T105
 66 OPND 30324000235 SRI-ARC      30326500010  53       14128     11 T101
111 OPND 30324000230 SRI-ARC      30326500007   4         904      6 T106
117 OPND 30324000231 SRI-ARC      30326500006   3       13752     20 T106
140 OPND 30324000074 BBN-TENEX    30327000301   2         944      6 T114
```

```
141 OPND 30324000314 CASE-10      30324600003   2          944        6 T105
142 OPND 30324000234 SRI-ARC      30326500011   5          944        6 T101


@NETSTAT%
*SPECIFIC CONNECTIONS %
**STATES OPND %
**SIZES 32%
**%

There are no connections of the specified type.


@NETSTAT%

*H TEN,TIP%
*S%

**JOBS 8
**


THE FOLLOWING  ARE UP:

SRI-ARC    BBN-TENEXB MITRE-TIP ETAC-TIP GWC-TIP  SAAC-TIP
UTAH-10    I4-TENEX   RADC-TIP  USC-ISI  DOCB-TIP ARPA-TIP
BBN-TENEX  AMES-TIP   NBS-TIP   USC-TIP

ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST       4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

 20 OPND 30325000300 BBN-TENEX   30324000375   4       17728      2  B 8
 45 OPND 30325000301 BBN-TENEX   30324000374   3         944      6  B 8


@NETSTAT%

*S%

**USERS TIP,BTHOMAS %
**%

ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST       4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

 23 LSNG    23100365
 32 LSNG     5600001
 46 LSNG     5600003
137 LSNG    23100371


@NETSTAT%

*SPECIFIC CONNECTIONS:
**NOT STATES LSNG %
**NOT HOSTS BBN%
**%

ACTIVE CONNECTIONS:
  I STATE LCL-SOCKET     HOST        4N-SOCKET LNK  BITS-ALLOC M-ALL BS/VT

 34 RFN1 30324000161 NBS-TIP        1600002  11         304     6 T102
 50 OPND 30325500003 USC-ISI    30324000020  31          96    20  B 8
 57 OPND 30325500002 USC-ISI    30324000021   3       17728     2  B 8
111 OPND 30324000070 NBS-TIP        4200003   2         720     6 T101
117 OPND 30324000071 NBS-TIP        4200002  23          48     0 T101
```

```
#                           RSEXEC
#
# RSEXEC is an experimental multi-computer Executive  Program.   It
# contains  several  self-documenting features.  The following is a
# typescript of an RSEXEC session:
#
# @RSEXEC
#
#  RSEXEC  2.7.1  BBN-TENEX  TUE 25-JUN-74 Ø9:Ø3-EDT
#  Type HELP<cr> for help.
# _HELP
#
#  "?" gives a list of commands.
#
# Use the  "DESCRIBE"  command  to  obtain  descriptions  of  other
# commands.  A good way to start is:
# _DESCRIBE RSEXEC<cr>
#
# Only enough of a command to uniquely identify it need  be  typed.
# "ESC"   invokes  command  recognition  and  completion.   Editing
# characters are:
#  ^A (Control A) - Character delete.
#  ^R (Control R) - Retypes current line or item.
# RUBOUT (or DEL) - Aborts current command (if  typed  while  still
# giving command or arguments).
#
# ^C and ^T are handled by RSEXEC.
#
# ^P may be used as a panic escape in case  your  terminal  becomes
# hung  while  linked.  It breaks the link, clears input and output
# buffers, and returns to the  higher  level  EXEC.   The  CONTINUE
# command  will  then  resume  the  RSEXEC  session  as if a ^C had
# occurred.
# _?
# Commands are:
#  ACQUIRE
#  APPEND
#  BIND
#  BREAK
#  CONTINUE
#  COPY
#  DELETE
#  DESCRIBE
#  DEVSTAT
#  DIRECTORY
#  ENTER
#  ESCAPE
#  EXEC
#  EXPUNGE
#  FULLDUPLEX
#  GET
#  HALFDUPLEX
#  HELP
#  HOSTAT
#  INITIATE
#  LEAVE
#  LINK
#  LIST
#  LOCATE
#  LOGOUT
#  NEED
#  NETNEWS
#  NETSTAT
#  PROEDIT
#  PROLIST
#  PURGE
#  QFD
#  QUIT
#  RECEIVE
#  REFUSE
#  RELEASE
#  RENAME
#  RESET
#  RUN
#  SERVERS
#  SITES
#  SNDMSG
#  START
#  TELCONN
#  TENXSTAT
#  TERMINATE
#  TIMECONSTANT
#  TRSTAT
#  TYPE
#  UNDELETE
#  USE
#  WHERE
#  WHO
#
# _DESCRIBE (command, term or ALL) RSEXEC
#
#  RSEXEC
#
# The Resource Sharing Executive is an evolutionary  multi-computer
# executive program.  It provides an environment in which the range
# of many features found on a single-Host time sharing  system  are
# extended beyond the boundaries of a single Host to encompass many
# Hosts on the ARPANET.
#
# At present RSEXEC includes facilities  for  inter-Host  user-user
# interaction  (see  descriptions  for  WHO,  WHERE,  SITES,  LINK,
# SNDMSG),  for  managing  "multi-Host"   file   directories   (see
# descriptions of ENTER and BIND) and for controlling multiple
# "jobs" on several Hosts (see descriptions for TRANSACTION and
# INITIATE).  In addition, the RSEXEC serves as a command language
# interpreter for TIP users.
#
# The DESCRIBE command can be used to obtain descriptions of all
# (accessible) RSEXEC commands and, in addition, the following
# terms:
#
#     BOUND-DEVICE, COMPONENT-DIRECTORY, COMPOSITE-DIRECTORY,
#     FILE-NAMES, INTERRUPT-CHARACTERS, MULTI-IMAGE-FILES,
#     PRIMARY-DIRECTORY, PROFILE, TRANSACTION, BUGCHK
#
# (TIP users accessing RSEXEC via the TIP "@n" command can use only
# a subset of the RSEXEC commands; they can obtain descriptions of
# only those commands (and related terms) they have access to.)
#
# The user interested in the design philosophy of RSEXEC and its
# implementation is referred to the paper "A Resource Sharing
# Executive for the ARPANET", Proceedings of 1973 National Computer
# Conference and Exposition, also NIC #14689).
#
# _DESCRIBE (command, term or ALL) ALL
#
# ACQUIRE (Component Directory) Component1 ... Componentn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command. Used
# to add the files in the component directories specified to the
# composite directory.  A directory need not be ACQUIREd in order
# to reference files in it or at the corresponding site.  However,
# file name recognition and completion will work only for files
# that are local or in ACQUIREd directories.  See also descriptions
# for COMPOSITE-DIRECTORY and COMPONENT-DIRECTORY.
#
# APPEND (file) FILE1 (to file) FILE2 <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
# Changes FILE1 by appending FILE2 to it.
#
#  BIND (device) DEVICE-NAME (to site) SITE-NAME <cr>
# or
#  BIND (device) DEVICE-NAME (to site) TIP-NAME <cr>
#  _(Port #) number <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
#
# Associates the device named with the Host named such that
# subsequent references to the device name refer to the device at
# the Host specified. For example, the sequence:
#
#     _BIND LPT MITRE-TIP
#       __(Port #) 1
#       _LIST TEST.DATA
#
```

# would produce a listing of file FOO.DATA at device port 1 on the
# MITRE-TIP. For binding to a TIP Port to work properly the TIP
# port in question must be set to "wild".
#
BOUND-DEVICE
The user can use the BIND command to specify that subsequent use
# of a particular device name is to refer to that device at a
specific site. Such a device is said to be "bound" to that site.
For example, the sequence of commands:
#

```
#       _BIND LPT USC-ISI <cr>
#       _COPY REPORT.DRAFT LPT: <cr>
#       _LIST PROGRAM.SOURCE <cr>
#
```

first binds the line printer to ISI and then causes two listings
to be produced by the ISI line printer.
#
BREAK<cr> Breaks terminal links (see LINK).
#
BUGCHK
RSEXEC contains a considerable number of internal consistency and
redundancy checks. If RSEXEC detects a malfunction it prints the
message:
#

```
#       _BUGCHK at NNNNN
#
```

and continues. Such messages are useful for debugging purposes
and recurring BUGCHK messages, together with the circumstances
under which they occur, should be reported to Bob Thomas
(BTHOMAS@BBN) or Paul Johnson (JOHNSON@BBN).
#
COMPONENT-DIRECTORY One of the file directories that may be
included in the user's composite directory. The syntax for
component directories is:
#

```
#            [Site]<Directory>
#
```

e.g., [NIC]<JONES>. There is an entry in the user's profile for
each component directory. The user may control which component
directories are, at any given time, included in his composite
directory via the ACQUIRE and RELEASE commands and the
subcommands of the ENTER command.
#
COMPOSITE-DIRECTORY
The collection of file directories specified in a user's profile
define his composite directory. The "contents" of the composite
directory are the union of the "contents" of the component
directories specified in the profile. Pathnames without site and

```
# directory qualification are interpreted with respect to the
# user´s composite directory. The ENTER command uses information
# in the profile to gather sufficient information to construct (a
# local copy) the user´s composite directory. See also
# descriptions for PROFILE and FILE-NAMES.
#
# CONTINUE <cr>
# Resumes execution interrupted by previous ^C.
#
#  COPY (file) FILE1 (to new file) FILE2 <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command. Makes
# a copy of FILE1 which is named FILE2.
#
#  DELETE (file) FILE <cr>
# or
#  DELETE (file) FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
# Deletes the file(s) specified. Files which have been deleted but
# not expunged may be "undeleted" by the UNDELETE command. Deleted
# files are automatically expunged at LOGOUT.
#
#  DESCRIBE (command, term or ALL) command<cr>
#  or
#  DESCRIBE (command, term or ALL) ALL<cr>
# Describes any (or all) command(s). In addition, DESCRIBE
# can be used to describe certain "terms" such as RSEXEC.
#
#  DEVSTAT <cr>
# Lists the (binding) status of all devices.
#
#  DIRECTORY <cr>
# or
#  DIRECTORY , <cr>
# __subcommand <cr>
#    .
#    .
#    .
# __<cr>
# or
#  DIRECTORY FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command. Prints
# information (as modified by subcommands, if any) about the
# file(s) specified or, if none are specified, all files in the
# user´s composite directory. The subcommands are:
#
#      __VERBOSE <cr>
#      __MULTI-IMAGE FILES <cr>
#      __SITES site1 ... siten <cr>
#      __DELETED-FILES <cr>
#
# See also descriptions of COMPOSITE-DIRECTORY and
# MULTI-IMAGE-FILES
#
#  ENTER (name) NAME (affiliation) AFFL (RSEXEC password) PWRD <cr>
# or
#  ENTER (name) NAME (affiliation) AFFL (RSEXEC password) PWRD , <cr>
# __subcommand <cr>
#     .
#     .
#     .
# __<cr>
# Grants access to distributed file system features of RSEXEC after
# constructing a composite directory for the user from his profile.
# If the user does not  have  a  permanent  profile  (e.g.,  hasn´t
# previously  used  the  ENTER  command  or  has chosen not to have
# RSEXEC maintain a permanent profile for him) RSEXEC will  acquire
# the  information  necessary to construct a profile for him.  NAME
# is the name the user has chosen to be known by to  RSEXEC;   AFFL
# is  his  affiliation  (e.g.,  AMES,  NIC,  ARPA  -  at present an
# arbitrary string);  PWRD is his RSEXEC password chosen  when  his
# permanent  profile  was  created.  W A R N I N G  :  RSEXEC
# distinguishes between upper and lower case letters in the  RSEXEC
# password.  The  user  may  use  the ENTER subcommands to control
# which components of his profile are acquired  for  his  composite
# directory.  The subcommands are:
#
#      ALL  all components are acquired
#      NONE no components are acquired
#      ACQUIRE COMPONENT1 ... COMPONENTn
#           components 1 through n are acquired
#
# See also the descriptions for  PROFILE,  COMPONENT-DIRECTORY  and
# COMPOSITE-DIRECTORY.
#
# ESCAPE (Character is) CNTL-CHAR <cr>
# Sets the  "return  from  transaction  escape  character"  to  the
# control  character  specified.  The escape character is initially
# ^Z.  See also the descriptions for INTERRUPT-CHARACTERS and USE.
#
# EXEC<cr>
# Runs the standard TENEX EXEC;  to return to RSEXEC use  the  EXEC
# QUIT  command.  If  he  has  previously ENTERed the user has the
# option of reacquiring the local  component(s)  of  his  composite
# directory  when he returns to RSEXEC from an inferior EXEC.  This
# is useful if he has added or deleted files while using the EXEC.
#
# EXPUNGE (deleted files) <cr>
# Irretrievably removes deleted files  from  the  user´s  composite
# directory.
# FILE-NAMES
# The RSEXEC extends the syntax for TENEX file names to  include  a
# Host component.  The syntax for file pathnames is:
#
#      [HOST]DEVICE:<DIRECTORY>NAME.EXTENSION;VERSION
#
# Where HOST is either the string "LOCAL" or the name of an ARPANET
# TENEX.   Partial  pathnames  may  be  used  within  RSEXEC.   For
# example, whenever the  site,  device  and  directory  fields  are
# omitted, the user´s composite directory is used as a default.  At
# present the TENEX "*" convention may  be  used  only  with  local
# files.  The  user must have a profile entry for a site before he
# can access files at that site.  A Component Directory for a  site
# need  not  be ACQUIREd in order to reference files at the site or
# in the Component Directory.  However, file name  recognition  and
# completion  will  work  only  for  files  that are local or in an
# ACQUIREd directory.  See description for PROFILE.
#
# FILE-TRANSFER-EXAMPLES
# The NEED command is a convenient way to move files  from  one  or
# more  Hosts  to  a  specified  destination  Host:  (assume in the
# following that the local Host is BBNA):  To move a group of files
# to the local primary directory:
#
# _NEED (files) F1 F2 ... FN <cr>
# e.g.,
#  _NEED (files) [ISI]<SUBSYS>NETSTAT.SAV [BBNB]<TENEX-132>SCHED.MAC
#  _[ISI]<SUBSYS>NETSTAT.SAV
#   [BBNB]<TENEX-132>SCHED.MAC
#
# To move a group of files to a specified directory at the local Host:
#  _NEED (files) F1 F2 ... FN <cr>
# __(in Component Directory) CD <cr>
# e.g.,
# _NEED (files) [ISI]<SUBSYS>NETSTAT.SAV [BBNB]<TENEX-132>SCHED.MAC ,
# NE=__(in Component Directory) [BBNA]<TENEX-132>
#   [ISI]<SUBSYS.NETSTAT.SAV
#   [BBNB]<TENEX-132>SCHED.MAC
#
# To move a group of files to another Host:
#  _NEED (files) F1 F2 ... FN , <cr>
#  _(in Component Directory) CD <cr>
# e.g.,
#  _NEED (files) [ISI]<SUBSYS>NETSTAT.SAV [BBNB]<TENEX-132>SCHED.MAC ,
#  _(in Component Directory) [ARC]<JONES>
#  _[ISI]<SUBSYS>NETSTAT.SAV
#   [BBNB]<TENEX-132>SCHED.MAC
#
# Note that component directories need not be "ACQUIREd" to move to
# or from them.  However, file name recognition and completion will
# only work for files that are local or in ACQUIREd directories.
# FULLDUPLEX<cr>
# Causes your terminal to be treated as fullduplex.
#
#  GET (Saved File) FILE <cr>
# *****fix description*****
# *****Not Implemented*****
#
#  HALFDUPLEX<cr>
# Causes your terminal to be treated as halfduplex.
#
#  HELP<cr>
# Prints a short help message.
#
#  HOSTAT<cr>
# Lists the status of network server hosts  as  maintained  by  the
# host survey program at MIT-DMCG.
#
#  INITIATE (transaction at) HOST-NAME (called) NAME <cr>
# Attempts to create a job for the user at the site specified.  The
# job  is  known  as  NAME.  The  user  will  be notified when the
# transaction is ready for use.  See also the descriptions for  the
# USE, TERMINATE, TRSTAT and PURGE commands.
#
#  INTERRUPT-CHARACTERS
# The following characters are handled as terminal interrupts
# by RSEXEC:
#
#       ^C (CNTL-C): interrupts the current activity, returning control
#                    to RSEXEC.  The CONTINUE command may be used to
#                    resume the interrupted activity.  when a transaction
#                    is being USEd, RSEXEC transmits the ^C to the
#                    remote transaction.
#
#       ^T (CNTL-T): prints CPU and console time used in RSEXEC
#                    session.  When a transaction is being USEd,
#                    RSEXEC transmits the ^T to the remote transaction.
#
#       ^Z (CNTL-Z): enabled only when a transaction is being USEd.
#                    Returns control from transaction to RSEXEC.
#                    The ESCAPE command may be used to change the
#                    transaction escape character from ^Z to
#                    another control character.
#
#       ^P (CNTL-P): RSEXEC "panic" escape.  Intended for use when your
#                    terminal becomes "hung".  It breaks all terminal
#                    links, clears terminal input and output buffers,
#                    and returns control to the top level EXEC.  The
#                    EXEC CONTINUE command may be used to resume the
#                    RSEXEC session.  When resumed in this way the
#                    RSEXEC acts as if the user had typed ^C.
#  LEAVE (distributed file system) <cr>
# Use of this command is limited to users who have gained access to
# the  file system features of RSEXEC via the ENTER command.  Makes
# the distributed file system features of the RSEXEC  inaccessible.
# Inverse of ENTER.
#
#  LINK (to tty #) number (at site) hostname<cr>
# or
#  LINK (to tty #)<cr>
# "Links" your terminal to  the  terminal  specified  at  the  host
# specified  such  that  the  output for either terminal appears on
# both.  If no hostname is given the local host is  assumed  and  a
# local link will be made.  The RSEXEC comment character ";" should
# be used when LINKed to prevent RSEXEC from interpreting  dialogue
# as commands.
#
# Links are broken by the BREAK command or by quitting RSEXEC.
#
#  LIST (file) FILE <cr>
# or
#  LIST (file) FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.  Causes
# a listing(s) of the specified file(s) to be output  to  the  line
# printer.  The command:
#       _COPY (file) FILE (to new file) LPT: <cr>
# will produce a listing without the formatting action of the
# LIST command.
#
#  LOCATE (file) FILE (at Component Directory) COMPONENT-DIR <cr>
# Use of this command is limited to users who have gained access to
# the  file  system  features  of  RSEXEC  via  the  ENTER command.
# Creates an image of FILE and places it in the Component Directory
# specified.  See also the description for MULTI-IMAGE-FILE.
#
#  LOGOUT<cr>
# Logs out from RSEXEC and TENEX.
#
#  MULTI-IMAGE-FILES
# The RSEXEC treats files with the  same  pathname  relative  to  a
# user's  composite  directory (i.e., identical name, extension and
# version components) as "images" of the same file.  Such a file is
# said  to  be  a multi-image file.  Although the profile file (see
# description of USER PROFILE) is transparent to the  RSEXEC  user,
# it is implemented as a multi-image file.
#
#  NEED (file) FILE1 ... FILEn <cr>
#
# or
#  NEED (file) FILE1 ... FILEn , <cr>
#  _(in Component Directory) CD <cr>
```

# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
# Creates an image(s) of the file(s) specified in the Component
# Directory specified. If no Component Directory is specified, the
# image(s) is placed in the local primary directory. See also the
# description of MULTI-IMAGE-FILE.
#
# NETNEWS<cr>
# Prints the latest network news.
#
# NETSTAT<cr>
# Runs the standard TENEX NETSTAT subsystem which gives network
# status information.
#
# PRIMARY-DIRECTORY
# For each site for which there are Component Directories there is
# a Component Directory designated the Primary Component Directory
# for that site. The Primary Directory for a site must be a
# "login" (rather than "files only") directory. It is used as the
# basis for access control checks at the site. The Primary
# Directory for a site is set by the user either implicitly (as the
# first "login" directory for the site added to his profile) or
# explicitly (via the PRIMARY command of the profile editor). See
# also descriptions for PROFILE, PROEDIT and COMPONENT- DIRECTORY.
#
PROEDIT <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command. Used
# to edit the user profile. PROEDIT subcommands are:
#

```
#      ADD     to add an entry to the profile
#      REMOVE  to remove an entry from the profile
#      LIST    to print the profile
#      CHANGE  to modify an existing profile entry
#      PRIMARY to change the primary directory for a site
#      UPDATE  to make the edits permanent (see below also)
#      QUIT    to return to the RSEXEC
#
```

# The syntax for a profile entry is:
#

```
#           [Site]<Directory>
#
```

# If successful, the ADD (REMOVE) command results in modification
# of the user's profile with the addition (removal) of the
# specified entry. In addition, if he has so chosen, his composite
# directory is modified by the addition (removal) of the
# appropriate files. If the user attempts to ADD an entry for a
# site for which the RSEXEC server program is down, the entry will
# be marked to indicate that it has not been verified (i.e.
name/password not checked) and that its files have not been added
# to the user's composite directory.

```
# If during the course of the  user's  RSEXEC  session  the  remote
# server comes up, the entry will automatically be verified and its
# files added to the users composite directory  (if  the  user  has
# indicated that he wants them).  The user has the option of making
# his edits permanent via the UPDATE command or when he leaves  the
# distributed file system environment via the LEAVE, QUIT or LOGOUT
# command.  The profile editor prompt character is "*".   See  also
# the descriptions for PROFILE, COMPONENT-DIRECTORY, and
# COMPOSITE-DIRECTORY and PRIMARY-DIRECTORY.
#
# PROFILE
# A  collection  of  user  specific  information  and  parameters
# maintained  for  the  user  by  the  RSEXEC.  At  present,  the
# information maintained includes an entry for each of  the  user's
# file  directories:  each entry consisting of Host name, directory
# name, password and account number or string.  The profile  editor
# (PROEDIT)  can be used to add or delete entries from the profile.
# If a user chooses to have the RSEXEC maintain a permanent  record
# of his profile a file named:
#
#      ]-RSPRF-[.NAME@AFFILIATION;1
#
# will be maintained in each directory named in the profile.  This
# file  is  itself  transparent  to the RSEXEC user.  Images of the
# profile file are suitably protected:  only the user  himself  may
# read  or  write  it  (its  protection attribute is P770000);  the
# passwords stored in it are encrypted  (using  the  user's  RSEXEC
# password  as a key).  The QUIT, LEAVE and LOGOUT commands ask the
# user if he wishes to have a permanent profile.
#
# PROLIST <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
#
# Prints the contents of the user's profile.
#
# PURGE (transaction) NAME <cr>
# break Causes forced termination of a previously INITIATEd job  or
# TELCONN  connection  by  breaking  network connection with remote
# site.  Intended for use only  when  TERMINATE  fails.   See  also
# descriptions for INITIATE, TELCONN, USE, TERMINATE, and TRSTAT.
#
#  QFD (file) FILE <cr>
# or
#  QFD (file) FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
#
# Prints a "quick" description of the file(s) specified.
#
#  QUIT<cr>
# Ends RSEXEC session.
#
#  RECEIVE (links)<cr>
# Sets terminal to accept links (default state).
# Undoes a previous REFUSE command.
#
#  REFUSE (links)<cr>
# Sets terminal to refuse links.
# Undone by a subsequent RECEIVE command.
#
#  RELEASE (Component Directory) Component1 ... Componentn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command. Used
# to remove from the composite directory the files in the specified
# component directories.
# See also the descriptions for COMPOSITE-DIRECTORY and
# COMPONENT-DIRECTORY.
#
# RENAME (file) FILE1 (to be) FILE2 <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.
# Changes the name of FILE1 to be FILE2.
#
# RESET <cr>
# Similar to the RESET command of the TENEX EXEC.
#
# RSEXEC
# The Resource Sharing Executive is an evolutionary multi-computer
# executive program. It provides an environment in which the range
# of many features found on a single-Host time sharing system are
# extended beyond the boundaries of a single Host to encompass many
# Hosts on the ARPANET.
#
# At present RSEXEC includes facilities for inter-Host user-user
# interaction (see descriptions for WHO, WHERE, SITES, LINK,
# SNDMSG), for managing "multi-Host" file directories (see
# descriptions of ENTER and BIND) and for controlling multiple
# "jobs" on several Hosts (see descriptions for TRANSACTION and
# INITIATE). In addition, the RSEXEC serves as a command language
# interpreter for TIP users.
#
# The DESCRIBE command can be used to obtain descriptions of all
# (accessible) RSEXEC commands and, in addition, the following
# terms:
#
#      BOUND-DEVICE, COMPONENT-DIRECTORY, COMPOSITE-DIRECTORY,
#      FILE-NAMES, INTERRUPT-CHARACTERS, MULTI-IMAGE-FILES,
#      PRIMARY-DIRECTORY, PROFILE, TRANSACTION, BUGCHK
#
# (TIP users accessing RSEXEC via the TIP "@n" command can use only
# a subset of the RSEXEC commands; they can obtain descriptions of
# only those commands (and related terms) they have access to.)
#
# The user interested in the design philosophy of  RSEXEC  and  its
# implementation  is  referred  to  the  paper  "A Resource Sharing
# Executive for the ARPANET", Proceedings of 1973 National Computer
# Conference and Exposition, also NIC #14689).
#
#  RUN (Saved File) FILE <cr>
# *****fix description*****
# *****Not Implemented*****
#
#  SERVERS<cr>
# Prints a list of the sites which (at times) run  RSEXEC  servers.
# These  sites  must  both  be  up  and  running  the  server to be
# accessible from RSEXEC.
#
# SITES (of user) username<cr>
# Lists the sites  (with  RSEXEC  servers  running)  at  which  the
# specified user is known.
#
# SNDMSG<cr>
# Runs a subsystem for sending messages  to  other  network  users.
# Messages  can  be  delivered only if the destination site runs an
# FTP  server  with  the  MAIL  command  implemented.   Undelivered
# messages will be deleted after a week.
#
#  START <cr>
# *****fix description*****
# *****Not Implemented*****
#
#  TELCONN (to site) HOST-NAME (with a connection called) NAME<cr>
# or
#  TELCONN (to site) HOST-NAME (with a connection called) NAME,<cr>
#  _option <cr>
#  _  .
#  _  .
#  _  .
#  _<cr>
# Attempts to establish a TELNET connection to the specified  HOST.
# If  successful,  the user´s terminal is connected to this network
# communication path.  To return to RSEXEC type ^Z (CNTL  Z).   Use
# of  the  connection may be resumed with the USE command.  Default
# socket is the logger (#1).   An  alternate  socket,  as  well  as
# desired  echo  mode and terminal characteristics may be specified
# in the second form of the command.  Type ?  in response to the
# option   prompt   to   list   available   options.   See  also  the
# descriptions of the USE and INITIATE commands.
#
# TENXSTAT<cr>
# Prints status information for TENEX  sites  with  RSEXEC  servers
# running.
# TERMINATE (transaction) NAME <cr>
# Terminates a previously INITIATEd job by sending it several  ^C´s
# and then logging it out.  Also can be used to terminate a TELCONN
# connection.  After  termination,  a  TELCONN  connection  can  no
# longer be USEd.
#
# TIMECONSTANT (for net connections is) value<cr>
# Sets the time  constant  used  for  interactions  with  non-local
# RSEXEC  server  programs.  If the remote server does not respond
# within the specified time the interaction is  aborted.  Possible
# values  are:  RAPID  (8 sec.), MODERATE (15 sec.), LETHARGIC (40
# sec.), and INFINITE (2 min.).  The  time  constant  is  initially
# MODERATE (15 sec.).
#
# TRANSACTION
# A user can instruct the RSEXEC to create a job for him at another
# site.  Such  jobs  are called transactions. See descriptions of
# the INITIATE, TELCONN, USE, TERMINATE and PURGE commands.
#
# TRSTAT <cr>
# Prints  status  of  previously  INITIATEd  jobs  and  TELCONN
# connections.  Possible status´ are:
#
#     PENDING             INITIATEd but login incomplete
#     USEABLE             can be used via USE command
#     USEABLE TELNET      TELCONN connection, can be used via USE
#                         command
#     TERMINATION PENDING TERMINATEd but logout incomplete
#     TERMINATED          TERMINATEd but not yet removed from
#                         RSEXEC´s transaction table
#
#  TYPE (file) FILE <cr>
# or
#  TYPE (file) FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the file system features of RSEXEC via the ENTER command.  Prints
# the file(s) specified on the user´s terminal.  Identical to  LIST
# command except for the use of the terminal.  The command:
#      _COPY (file) FILE (to new file) TTY: <cr>
# will copy a file to the user´s terminal  without  the  formatting
# action of the TYPE command.
#
#  UNDELETE (file) FILE <cr>
# or
#  UNDELETE (file) FILE1 ... FILEn <cr>
# Use of this command is limited to users who have gained access to
# the  file  system  features  of  RSEXEC  via  the  ENTER command.
# Revives the DELETEd file(s) specified by restoring the file(s) to
# normal  status;  e.g.,  they are once again accessible to RSEXEC
# commands.  The inverse of DELETE.
#  USE (transaction) NAME <cr>
# or
#  USE (transaction) NAME, <cr>
#  _option <cr>
#  _  .
#  _  .
#  _  .
#  _<cr>
#
# Connects the user´s terminal to a previously INITIATEd job or  to
# a previously established TELCONN connection.  To return to RSEXEC
# type ^Z (CNTL-Z);  to  transmit  ^Z  to  the  job  type  the  two
# character  sequence  <^><Z>.  In general typing the two character
# sequence  <^><x>  will  transmit  x  if  x  is  a  non-alphabetic
# character and CNTL-x if s is an alphabetic.
#
# Thus to transmit ^P (the RSEXEC "panic" escape) type <^><P>.   If
# the  user  has  ENTERed  and  uses  ^Z to return to RSEXEC from a
# transaction,  he  has  the  option  of  updating  his  composite
# directory  to  reflect  any additions or deletions resulting from
# his USE of the transaction.  The ESCAPE command may  be  used  to
# change the transaction escape character from ^Z.
#
# If the second form of the command is used options concerning  the
# network  connection  (echo modes, terminal characteristics, etc.)
# may be specified.  To list the allowable options  type  "?"  when
# prompted  with  "_".  While USING any TELNET connection (created
# by either INITIATE or TELCONN) certain TELNET  control  functions
# may  be  invoked  by  typing  the  TELNET  dynamic  option escape
# character  followed  by  a  character  indicating  the  desired
# function.  The  dynamic  option escape character is initially ^D
# (CNTL D) but may be changed using the DYNAMIC option.
#
# The recognized commands are:
#
#       <^D><L> means to do Local echoing
#       <^D><R> means to do Remote echoing
#       <^D><T> means to Transmit accumulated chars immediately
#       <^D><B> means send TELNET Break character
#       <^D><S> means send the TELNET Synch sequence
#       Any other character will be transmitted as data.
#
# To send a <^D> character through the network, type <^><D>.
#
# WHERE (is user) username<cr>
# Lists all active jobs belonging to  the  specified  user  at  all
# sites (with RSEXEC servers running).
#
# WHO<cr> or WHO (at site) hostname<cr>
# Lists users with  active  jobs  at  specified  (or  all)  network
# site(s) with RSEXEC servers running.
# _QUIT
#
# @
```

Telnet User Guide

User Telnet (hereafter called Telnet) provides facilities for communicating with host computers via the ARPA network utilizing the TELNET protocol. The purpose of the Telnet program is threefold. It converts various terminals connected to TENEX into a standard type of terminal called a network virtual terminal (NVT) by interposing programs in the character streams between the terminal keyboard and printer and the terminal port on the host computer. Secondly, it provides information about the network to assist a user in establishing connections. Thirdly, it multiplexes the terminal among several remote jobs.

Telnet Command Interpreter

Instructions to the Telnet program are given via the Telnet Command Interpreter. When in command mode (see below), characters typed on the user's terminal are read by the Telnet command interpreter and decoded as commands to perform various actions by Telnet.

The Telnet command interpreter has two unique features. The command interpreter will refuse to hear anything it does not understand. With full-duplex terminals, this means that no echo will appear for characters which are not valid successors of the previous input. In any case, the character is ignored and a bell is typed out. The input stream that has already been typed is not forgotten however. Therefore, it is only necessary to type the correct character and not the complete command. This feature may be turned off with the "no fancy.command.interpret" command.

The other unique feature of the Telnet command interpreter is the use of question mark to discover what the command interpreter expects next. Typing a "?" at any time in command mode will elicit a list of words the command interpreter is expecting. Thus, typing a "?" when nothing has been typed will yield a list of all possible top-level commands. Typing "co?" will yield a list of all commands starting with "co". Typing "connection.to ?" will yield a list of possible arguments to the "connection.to" command.

The command interpreter provides command completion whenever a terminator is typed (full-duplex terminals only) and an exact match is achieved with some command or a unique initial substring is typed. Command completion may be suppressed with the "concise" command. Terminators are space, comma, alt-mode, and carriage return. Terminators are often not distinguished and are thus equivalent. Where necessary, comma is used to separate list items, space terminates a command or option and signals the desire to specify more options, carriage return ends a command unless more information is necessary. Altmode is the same as space except that it will cause command completion in those modes where it is normally suppressed.

Command/Remote Mode

As mentioned above, characters typed on the terminal keyboard may be used in two ways: either as commands to Telnet, or as input to the remote host. The choice is made on the basis of whether Telnet is in remote mode or command mode. In command mode, characters typed on the terminal keyboard are read by the Telnet command interpreter and decoded as commands to perform various actions. TELNET is initially in command mode and will revert to command mode whenever the Telnet escape character (see below) is typed.

The opposite of command mode is remote mode. In remote mode, characters typed on the keyboard (with certain exceptions) are not examined by Telnet at all, but are merely passed on to the remote host computer. Remote mode is normally entered after any command is executed when the current connection exists. The "local.mode" command may be used to defeat this. The effect of the "local.mode" command is cancelled by the "remote.mode" command or by the "connection.to" or "retrieve.connection" commands.

Escaping Back to Command Mode

At any time, typing the Telnet escape character (initially control-Z (SUB)) will cause Telnet to stop whatever it is doing and return to command mode. Occasionally, a slight delay may be experienced due to the need to clean up whatever was happening at the time. Telnet announces the switch to command mode by the appearance of a sharp sign "#" at the left margin. Telnet also indicates the transition out of command mode by the appearance of another sharp sign followed by a new line.

WARNING: If you have control-Z anywhere in your programming, you
# should change your escape character for Telnet to other than
control-Z to avoid mishaps.

Making a Connection

There are two ways to make a connection. Typing "connection.to <host> [<qualifiers>]" or simply typing "<host> [<qualifiers>]" will cause a connection attempt to be made. If successful, the connection will be said to be complete and the terminal will be
# placed in remote mode. If unsuccessful, the connection will be
# said to be "incomplete because ---" with a reason given; also if

# the remote host is down, a line is typed telling why and for how
long. By terminating the host name with a space, one or more qualifiers may be specified. Ordinarily socket 1 is assumed. Thus without a qualifier, the connection will be made to the "logger" on the remote system. By using an octal number as a qualifier, the connection will be made to the socket so specified. A set of names is available for specifying the socket desired. This set consists of names for all the standard sockets.

The "wait" qualifier may be used to camp-on the connection. This qualifier causes Telnet to repeat the attempt to connect in the event of a failure until it finally succeeds. An initial failure causes a message to that effect to be printed. When the attempt finally succeeds, bells are typed out to wake the user up. The attempt to connect may be aborted by typing the Telnet escape character.

The "load.settings.from..." qualifier (possibly qualified with "no") may be used to cause (inhibit) the mode flags to be initialized from the mode file. When inhibited, the current modes are used.

The "name.for.connection" qualifier may be used to specify a name for this connection other than the one assigned by Telnet. A name for the connection may also be given later by the "name.for.current.connection" command.

Disconnecting

The "disconnect" command is used to close the current connection. This will not necessarily log you out from the remote host so you should perform the logout procedure for that host before disconnecting. The disconnect command takes an optional argument specifying the name of a particular connection to be disconnected. See multiple connections and connection names below.

In the event that the network connections are severed by a network failure, the message "IO error for connection <name>" is printed, the connection is disconnected, and Telnet reverts to command mode. This may happen even if the error occurs on a connection which is not current. If the remote host initiates a disconnect, a message to that effect is printed and the same action is taken.

# If the remote host on the current connection stops responding
# when input is being sent, a line is typed, "Host not responding
# on connection xxx." (In this case the connection is <u>not</u> lost.)

# When the remote host resumes operating, the user is informed:
# "Service restored on connection xxx."

Echo Control

Telnet allows several options concerned with echoing. Echos may be generated by the terminal, by Telnet, or by the remote host. Telnet determines if the terminal is generating echoes when started by examining the mode word for the terminal. The "terminal.type.is" command may be used to change this.

If the terminal is echoing, then Telnet will do everything possible to cause the remote host to not generate echoes, and Telnet will not generate echoes itself. If the terminal is not generating echoes, then Telnet determines whether it should echo or not by information in the mode file (if any) or by the "echo remote"/"echo local" commands, or by information sent from the remote host.

Telnet keeps the remote host informed about how echoing is being done and if the remote host is suitably equipped, it will follow along. If not, then the user will have to give commands to the remote host to achieve the proper echoing. Telnet also will respond to commands from the remote host concerning who should be echoing. If Telnet believes the terminal is doing its own echoing, it will respond to any request from the remote host to not echo by an "I´ll echo" command.

Line Buffering and End of Line Conventions

Telnet provides an optional line buffer for use with line-oriented operating systems. In this mode, characters typed in remote mode are stored in a local buffer up through an end of line. Prior to the end of line, the currently buffered line may be edited using control-A (SOH) or control-H (BS) to delete characters, control-X (CAN) to delete everything, and control-R (DC2) to retype the current contents. Telnet always converts the TENEX EOL into the NVT EOL. TENEX in turn converts a carriage return into the TENEX EOL. Thus typing a carriage return will cause the buffered line to be transmitted. Linefeed may also be used to terminate a line. In this case, the transmitted line will end with only linefeed, not the NVT EOL.

Telnet provides an optional linefeed echo for carriage return. If the remote host provides a linefeed also, then the echo generated by Telnet should be suppressed with the "echo no linefeed.for.carriage.return" command. In remote echo mode, Telnet generates no echos whatsoever. In this mode, all echos must be provided by the remote host.

Status Commands

Several status commands are available for discovering facts about the network. None of these commands will affect the state of the current connection. The status commands include where.am.I, status.of, netstatus, and socket.map. These commands are summarized below.

Special Characters

Several commands are available to send characters which do not appear on the terminal. "Code" takes an octal (decimal if preceded by "D", hexadecimal if preceded by "H") argument and sends the character with that code. The word "code" may be omitted and just the argument typed. "Control" takes a character argument and sends the corresponding control character (the low order five bits of the character) is sent. The "!break!" command sends the NVT break character which is mapped by some systems into the equivalent of the attention, quit or break key which appears on some terminals.

To facilitate operation with systems requiring frequent use of special characters or lower/upper case graphics which a particular terminal may lack (e.g. 33 Teletypes have no lower case), case shift characters may be defined for upper/lower character/lock shifts and characters may defined which will translate into attention or break (NVT 201), and the synch sequence. The "case.shift.prefix.for", "attention.character=", and "synch.character=" commands are available to independently set each of these characters. In addition, a character may be defined ("quote.prefix" command) to be a single character quote. The character following this character is always sent regardless of any special action it may otherwise have.

If possible, case shift characters will be used to indicate the case of both input and output. Thus the case shift characters may not be echoed when typed but rather before the output.

All special characters are listed by the "current.modes.are" command. This includes the escape character and the clear output buffer character.

Leaving Telnet

To leave Telnet, it is first necessary to return to command mode by typing the escape character. This is because while in remote mode all characters except the escape character are passed on to the remote host or modify characters passed to the remote host. Once in command mode, you may return to the EXEC by typing control-C (ETX) or by using the "quit" command. Continuing from the EXEC will resumé with no loss. The "logout" command will disconnect from any remote job and logout your local job. The "exec" command will start up an inferior EXEC under Telnet. From this inferior EXEC, it is possible to perform assemblies or any other task involving the running of subsystems. The "run" command allows an arbitrary program to be run in an inferior fork of Telnet. The "run" may be interrupted by the Telnet escape character.

Multiple Connections

Telnet provides a facility for multiplexing a user´s terminal among several remote jobs thus allowing several simultaneous activities. This is done by giving a name for each connection as it is created. The user may specify the name, or Telnet will default the name to a number. The "retrieve.connection..." command causes the named connection to be made current and remote mode to be entered. Non-current connections remain active, but any output received is buffered until that connection again becomes active. Terminal input goes only to the currently active connection.

The name of the current connection may be changed after it is established by means of the "name.for.current.connection" command. The name so specified may be up to 6 characters in length and must be unique.

Typescript

# Telnet provides a means of saving on a file a copy of the
# typescript for a session. This is useful for producing hard copy
# of the session when using a scope terminal or for producing
documentation of procedures or demonstrations. Telnet always keeps a typescript on the temporary file "TELNET.TYPESCRIPT;T" in the connected directory. The "typescript.to.file" command may be used to specify a different file. The typescript consists of a nearly exact copy of what appears on the terminal with the exception of that which occurs during the execution of the "exec" or "run" or "ddt" commands. "Nearly" refers to slight differences in the spelling of file names in certain Telnet
# commands. For privacy, the typescript file is given a protection

that allows no access to anyone but "self".

Diverting Output

The output stream may be diverted to some other file with the "divert.output.to.file" command. While diverting output, Telnet sends all output to the indicated file and sends a line to the terminal only when the terminal´s output buffer is empty. Thus the terminal monitors the transmission of the stream to the file. The diverted output consists only of characters from the remote host. Telnet commands and responses do not appear in the diverted information. This mode is useful as a primitive file
# transfer mechanism or to allow printing of large amounts of
# terminal output to be done with the lineprinter. It is cancelled
# by "no divert.output ...".
#
#
#
Input from a File
#
The input stream to a remote job may be taken from a file instead
# of the local terminal by means of the command
"take.input.stream.from.file". Telnet blocks terminal input to
the connection current when the file is specified, and transmits
# characters from the named file (echoing as usual according to
# current modes). However, input to other connections and in
# command mode is from the user´s terminal. When the given file
# reaches EOF the file is closed and released, and input reverts to
# the terminal. The user may also manually cancel file input by
escaping to command mode and giving "no take.input ...". This
# mode is useful for routine sequences performed in the remote job.
Note that a connection must be established and current when input
to it is diverted to a file.

Telnet Command Summary

Connection.to <host> or host name

Performs ICP to connect to the indicated host. Options are available for specifying initial
# connection socket name or number, and initializing
# modes from the mode file via the following
# subcommands. Note that if <host name> is used as
# a command, only the name of a server host may be
# given (e.g., BBN-TENEX). The argument for
# "Connection.to" may be any host name or an octal
# host number.

<octal number>

An ICP is performed to connect to the indicated service socket. Normally socket 1 is assumed.

Logger

Sets socket to 1.

Wait

The connection attempt is repeated until successful.

Name.for.connection.is <name>

Sets the name for this connection as specified.

[no] load.settings

Determines whether to use current mode settings or to load new ones from the mode file.

Disconnect <cr>

Disconnects the current connection. This will not necessarily log you out from the remote host.

Perform the necessary operations before disconnecting.

Disconnect <name>

Disconnects the connection with the specified name.

# Net.exec
#
# Connects to BBN socket 15600031 where-in the
# RSEXEC (Resource-Sharing Executive) is found.

Status.of <host>

Performs ICP with the indicated host and prints its status.

Echo.mode.is

Sets echo mode according to the following subcommand.

[no] remote

Turns off echoes generated by Telnet and signals the remote computer to generate echoes. Some hosts are not yet equipped to handle this signal and may require additional action to cause the remote computer to generate echoes. If Telnet believes it is connected to a local half-duplex terminal, it will complain about remote echoes but do it anyway.

[no] local

Turns on Telnet generated echoes and signal the remote computer to not generate echoes. Note that Telnet never generates echoes for terminals it believes have local echo of their own.

[no] linefeed.for.carriage.return

TENEX translates carriage return to EOL, Telnet sends the EOL as the TELNET EOL (i.e. carriage return-linefeed). For some systems, the TELNET EOL is translated into carriage return. For these systems, the appropriate echo is carriage return. Other systems translate the TELNET EOL into carriage return-linefeed. For these systems the appropriate echo is carriage return-linefeed. This subcommand causes the latter echo to be generated.

[no] control.character.echo.for <list of characters>

Turns on local echoes for the indicated control characters. Normally only control-G,J, and M (bell, linefeed, and carriage return) are enabled.

Terminal.type.is

Allows the user to change Telnet´s opinion of his terminal according to the following subcommands. Each command may be preceded by the word "no" to negate its meaning.

Half-duplex

Terminal generates its own echoes.

Full-duplex

Terminal does not generate its own echoes.

[no] lower.case

The terminal has lower case characters.

Local.mode

If connected, this command prevents Telnet from returning to remote mode after each command.

Remote.mode

If connected, this command causes Telnet to return to remote mode after each command. If not connected, it does nothing.

No

May appear before some commands to reverse their action.

Current.modes.are

Prints the state of connection terminal mode flags, and all special characters.

[no] character.mode

Causes each character typed to be transmitted as it is typed.

[no] line.buffer

Causes Telnet to accumulate a line of text before transmitting. A line ends on linefeed or EOL or altmode (esc). The line may be edited with control-_a_, _x_, and _r_.

[no] raise

Causes lower case letters to be transmitted as their upper case equivalents.

[no] lower

Causes upper case letters to be transmitted as their lower case equivalents.

[no] transparent.mode

Causes all characters to pass through Telnet and TENEX untouched. This is needed for special terminals such as the IMLAC using special character stream protocols.

[no] case.shift.prefix.for

Allows the specification of the four case shift characters according the following four subcommands.

Lock.lower.case

Same as the "Lower" command. Subsequent upper case input will be converted to lower case.

Char.lower.case

Converts the following letter to lower case.

Lock.upper.case

Same as "Raise" command. Subsequent lower case input will be converted to upper case.

Char.upper.case

Converts the following character to upper case.

[no] unshift.prefix

Causes all following characters to be unshifted. I.e. undoes both an upper case lock and a lower case lock.

[no] quote.prefix

Causes the following character to be transmitted without regard to any special significance it may have.

[no] synch.character

The specified character will be converted to the TELNET synch sequence. The TELNET synch sequence is used to cause the remote host examine its input stream to the current point for any special characters (interrupts, attentions etc.). All non-special may be thrown away.

[no] attention.character

The specified character will be converted to the TELNET break or attention character. This character is equivalent to the attention, quit, or break key on certain terminals and may be necessary for using some systems. The !Break! command generates the same character.

Concise

Turns off automatic command completion. Saves typeout at the expense of readability.

Verbose

The opposite of concise.

[no] fancy.command.interpret

Commands are checked character by character. If a character does not fit, it is ignored and not echoed (full duplex terminals only).

[no] divert.output.stream.to.file

Causes all subsequent output from the remote computer to be written on the specified file. Use "No divert..." to stop this.

# [no] take.input.stream.from.file
#
Causes subsequent input to the remote host on the
# current connection to be read from the specified
# file; input to other connections and in command
# mode is still from the user's terminal. File is
# automatically closed and released at EOF; user
# may force this by "No take.input...", after
# escaping to command mode.

[no] typescript.to.file

A record of the session is kept on a file including both input and output. This is useful for providing hard copy with scope terminals.

Escape.character=

The specified character becomes the Telnet escape character. This character must be a TENEX interrupt character. "?" will type what these are.

# WARNING: If you have anywhere in your programming
# a control-Z you should change your escape
# character in TELNET to other than control-Z to
# avoid mishaps.

Clear.output.character=

The specified character becomes the clear output buffer character. Typing this character generates an interrupt which causes the terminal output buffer and any accumulated output to be cleared.

Help

Prints the file <SYSTEM>TELNET.HELP on the user's terminal.

Netstatus

Runs <SUBSYS>NETSTAT.SAV.

Socket.map

Prints a list of all current connection on the system. Optional arguments may be used to select a particular host and a particular connection state.

Run

Runs the specified file. Like the EXEC's run command.

Quit

Returns from Telnet to the superior fork (usually the EXEC). May be continued with no loss.

Logout

Logs out the local job (not the remote one). Requires confirmation with a carriage return.

Reset

Re-initializes Telnet producing an essentially virgin copy.

Ddt

Enters ddt. If ddt is not loaded, this will result in an unexpected interrupt. No harm is done if this happens.

Exec

Starts up an inferior EXEC under Telnet. This EXEC may be used like an ordinary EXEC to run subsystems etc without disturbing any existing connections. The Telnet escape character will return to Telnet however.

Code

Transmits the character specified by the argument. The argument is a taken as an octal number unless preceded by "d" for decimal or "h" for hexadecimal. The argument may be preceded by "o" for octal.

The "code" command argument may be used as a command by itself and will cause the indicated code to be transmitted.

!break!

Transmits the TELNET break character.

!synch!

Transmits the TELNET synch sequence. Occasionally the "!synch!" command will work where the synch character will not since the command bypasses the buffering which may interfere with the use of the synch character.

Write.modes.for.host

Causes the current mode flags to be saved on the <SYSTEM>TELNET.MODES file under the specified host. Requires write access to the file and is thus not available to ordinary users.

Retrieve.connection.under.name

Retrieves the connection previously saved under the specified name.

Wait.for.any.active.connection

Used with multiple connections to wait for and switch attention to the next connection that has any output waiting. Useful when several independent tasks are being run and you wish to know when one completes and switch to that task.

Where.am.I

Prints a summary of the local job, system, user, terminal and the remote host and socket.

[no] Signal.waiting.output

Causes all non-current connections to print a message when output becomes available.

# Host.names
#
# Lists all current host names with corresponding
octal host numbers.

List.connections

Lists the name, local socket, foreign host, and foreign socket of all connections.

Flush.host

Marks all connections to the specified host as dead and sends a reset to that host. Requires wheel or operator special capability.

An initial semi-colon causes the remainder of the line to be ignored. Useful for comments or typing to links.

Index

Commands are given in all caps.

TIPCOPY

TIPCOPY DOCUMENTATION NOV. 1, 1974

Tipcopy will copy an ASCII text file to a TIP port.

The program is set up so that the receiving TIP is the TIP of the controlling terminal. The port number on that TIP is defaulted to be 2. You may change these settings. (To be explained later).

To use the program type:

```
@TIPCOPY <cr>            OR    @TIPCOPY <space>
```

The program will ask for a file name. To send the file to the TIP and port as explained above, just type a carriage-return to confirm your file selection. To be allowed to select option such as a specific TIP or port, confirm with a comma (,).

At this point TIPCOPY will allow you to type commands. Typing a question mark (?) will print a list of the commands currently available. A list is shown below. You may select any option you wish, or may undo a defaulted option by typing NO followed by the command. It is not possible to undo the destination TIP or a device command. To change these just reuse the command. Type an escape when you have completed your options selection.

```
     L FOR LIST OF OPTIONS ALREADY SELECTED
     T FOR TIP NAME IF OTHER THAN THE CONTROLLING TERMINALS TIP
     F FOR FORMFEEDS OVER PERFORATION
     H FOR HEADINGS
     P FOR PAUSES BETWEEN PAGES
     SI FOR SIMULATE FORMFEEDS OVER PERFORATIONS
     SP FOR SPECIFIC PAGES
     D FOR OCTAL DEVICE, IF OTHER THAN PORT 2
     C FOR COLUMN 1 FORTRAN INTERPRETATION
     M FOR MULTIPLE COPIES OF A FILE
     X FOR SELECTABLE LINE SPACING (DEFAULT=1)
     Y FOR PRINTING CONTROL CHARACTERS
     Z FOR SELECTABLE NUMBER OF FORMFEEDS AT END (DEFAULT=3)
     ? FOR HELP, PRINTS THIS INFORMATION

 TYPE ESCAPE WHEN COMPLETED OPTIONS SELECTION
 USE THE "NO" COMMAND TO UNDO A SELECTION, OR TO
 REMOVE THE TITLE PAGE WITH THE COMMAND "NO T".
 THE DEFAULT SETTINGS INCLUDE FORMFEEDS OVER
 PERFORATIONS, HEADINGS AND NOT PRINTING CONTROL CHARACTERS.
 CONTROLLING TERMINALS ON THE BBN-TESTIP WILL
 AUTOMATICALLY SEND FILES TO THE NCC-TIP.
```

# From time to time a new version of TIPCOPY will be written with
# the modifications users requested. This program will be called
# NTIPCOPY. I urge you to try this program and report to MALMAN
# any problems you may have had.
#
#
#
#
#
#
#
# <IMP>PLOT.SAV
#
# This is a Tipcopy like program except:
#
#   2 commands
#
#   T   TIP command if other than TESTIP
#
#   D   DEVICE command if other than port 22[8]
#
#
# 5 bit bytes (CALCOMP STYLES) to a tip port

XED - Experimental Editor

? outputs this message.

H HELP

The command letter which follows the H will be explained. H<CR> generates an editor manual. H<space> describes the interrupt characters.

The Legal Command Letters are:

```
^A,^H,DEL Delete char   ^W Delete word   ^X Delete line   ^R Re-type line
^Q Abort command        ^V,^^,^\ are all escape characters
A(ppend)        B(ackup)        C(hange)        D(elete)
E(xit)          F(ind)          G(roup)         H(elp)
I(nsert)        J(am)           K(ill)          L(ist)
M(odeset)       N(otemodes)     O(utput)        P(rint)
Q(uit)          R(ead)          S(earch)        T(ype)
V(iew)          W(rite)         X(change)
$ Go to End     = Current Line  - Move Back     + Move Forward
, Context       ^ Print Prior   / Print Current LF Print Next
_ Format        ; Comment
% Call SNDMSG   ! Run program   @ Call EXEC     " Mode dialogue
```

H HELP - Help accepts a command character following the H command. A description of the command represented by that character is then printed on the terminal. H(elp) followed by a carriage return additionally requests a file name. A brief reference manual is created in the file.

; COMMENT - All text input from the terminal following the ";" is ignored until the occurrence of a ^Z or a ^Q.

POSITIONING AND SEARCHING COMMANDS

Most commands in the editor operate on the current line of text within the text buffer. Lines in the text buffer are numbered consecutively starting at 1. The commands in this section describe ways to examine the contents of the text buffer and to change to current line.

$ DOLLARS - $ makes the last line in the text buffer the NEW current line. It also prints the number of lines in the text buffer.

NUMBERS - Numbers which are input as commands cause the current line to become the line with the number input. If a digit is incorrectly input, it may be canceled with a ^A, ^H, or DEL. The entire number may be cancelled with ^Q.

The commands ,(context), +, -, K(ill), T(ype), and M(odeset) all expect to receive numbers following. These numbers may be similarly edited with ^A, ^H, and DEL. ^Q cancels the entire command.

: COLON - K(ill) and T(ype) expect to receive a count of the number of lines on which they should operate. However, if the number is preceded by a :, they operate on the current line through the line number input instead of n lines starting at the current line.

Example: 12T:15. is the same as 12T4.

. DOT - . is a convenient character to terminate numbers, and is a null operator to the editor. Space and carriage return would also work as well.

= EQUALS - = prints the line number of the current line.

^ UP ARROW - ^ moves the current line to the line preceding the current line in the text buffer and prints the NEW current line. This is the same as -/.

LF LINEFEED - LF advances the current line 1 line forward in the text buffer and prints the NEW current line. This is the same as +/.

+ PLUS - Plus accepts a number following it, and advances the current line that number of lines forward. If no number follows, the current line is advanced 1 line forward. If this command is input by accident, typing ^Q will cancel it.

- MINUS - Minus accepts a number following it, and moves the current line that number of lines backwards. If no number follows, the current line is moved 1 line backwards. If this command is input by accident, typing ^Q will cancel it.

F FIND - Find accepts a string from the terminal. It then locates the first occurrence of the string in the text starting with the current line throughout the entire text buffer. The NEW current line is the line found, or the current line if no match was found. If the search string was last found at the current line, the search will start at the NEXT line (so repeated F(ind)s will locate successive occurrences of the search string in the text). If just a <CR> is typed to the prompt for a string, Find uses the string from the last F(ind) or S(earch). If the search string is a single null character (specified by control-V control-@ [^V^@] or control-^ @

[^^@]), then XED will search for a null line (no text, only <CR><LF>). If this command is input by accident, typing ^Q will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode. This command normally looks for the string any place in the line, independent of upper and lower case differences.

^E If ^E is specified, the match will be exact only.
^B If ^B is specified, the match will be only at beginning of lines.
^S If ^S is specified, XED will enter Change on the line(s) containing the string, positioning the Change cursor at the found string.

During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

S SEARCH - Search accepts a string from the terminal. It then locates ALL occurrences of the string in the text, starting with the current line. The NEW current line is the last occurrence found, or the starting line, if no occurrences were found. If just a <CR> is typed to the prompt for a string, Search uses the string from the last F(ind) or S(earch) command. If the search string is a single null character (specified by control-V control-@ [^V^@] or control-^ @ [^^@]), then XED will search for a null line (no text, only <CR><LF>). If this command is input by accident, typing ^Q will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode. This command normally looks for the string any place in the line, independent of upper and lower case differences.

^E If ^E is specified, the match will be exact only.
^B If ^B is specified, the match will be only at beginning of lines.
^S If ^S is specified, XED will enter Change on the line(s) containing the string, positioning the Change cursor at the found string.

During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

LINE EDITING COMMANDS

I INSERT - The insert command accepts text lines from the terminal. These text lines are inserted BEFORE the current line in the text buffer. This command is normally terminated by a ^Z. It ^Q is typed, any text typed on the current line is ignored and XED returns to command mode. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

A APPEND - The append command accepts text lines from the terminal. These text lines are appended AFTER the current line in the text buffer. This command is normally terminated by a ^Z. It ^Q is typed, any text typed on the current line is ignored and XED returns to command mode. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

^I (tab) TAB APPEND - acts like APPEND, with a tab as the first character of the first line appended.

D DELETE - The delete command deletes the current line. Once input there is no way to recover the line. For this reason K(ill) is considered a safer command.

K KILL - Kill accepts a number following it, and deletes that number of lines starting at the current line. The NEW current line is the line AFTER the deleted lines. If no number follows, only the current line is deleted. If the number following is preceded by a :, instead of killing n lines, it will kill all lines from the current line up to, and including, line n.

Example: 12k:15. kills lines 12,13,14,15.

All the lines which are deleted are saved in the text dump. Each subsequent kill destroys the previous text dump contents. Commands P(rint text dump) and J(am text dump) operate on the text dump. If this command is input by accident, typing ^Q will cancel it.

P PRINT TEXT DUMP - Each K(ill) operation saves the deleted lines in the text dump. P(rint text dump) prints the current contents of the text dump. See also J(am text dump). While this command is executing, typing ^Q interrupts it and returns to XED command mode.

' SWITCH DUMP - Exchanges the current contents of the text buffer with that of the text dump (see K(ill), J(am), P(rintdump) commands). This command is useful for performing operations with sections of large files. By K(ill)ing a section of text into the dump, then switching the dump with the buffer, operations may then be performed on the small section. Repeating the command causes repetitive switching (i.e., two ' commands in a row have no effect).

J   JAM - Each K(ill) operation saves the deleted lines in the text dump. J will jam the current contents of the text dump after the current line. The NEW current line will be the last line appended from the text dump. See also P(rint text dump).

/   SLASH - / prints the current line.

^   UP ARROW - ^ moves the current line to the line preceding the current line in the text buffer and prints the NEW current line. This is the same as -/.

T   TYPE - Type accepts a number following it, and prints that number of lines starting at the current line. The NEW current line is the last line printed. If no number follows one line is printed. If the number following is preceded by a :, instead of typing n lines, it will type all lines from the current line up to, and including, line n.

Example: 12t:15. types lines 12,13,14,15.

If a T command follows a T command, then the current line is advanced before the second T command, so the same line is not printed twice. If this command is input by accident, typing ^Q will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

V   VIEW - starting at the current line, V(iew) prints the next page of text (approx. 20 lines). The NEW current line is the last line printed. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

,   CONTEXT - ,(context) accepts a number n following, after which it types the n lines BEFORE the current line, the current line, and n lines AFTER the current line. The effect is to type (2n+1) lines, with the current line in the middle. If n is not given, 5 is assumed. ,(context) does NOT change the current line. If this command is input by accident, typing ^Q will cancel it.

L   LIST - List will print the entire text buffer from begining to end on the terminal. If is the same as saying TTY: to W(rite). There is a way to have a document line stating the current file name, time, and person who edited the file in the front of the file each time it is written. See) and (for further details. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

INTRA-LINE EDITING COMMANDS

C CHANGE - The change command is used for editing a single line of text. There are three basic operations in the change command. First characters may be copied from the old copy into the new line (SPACE,E,S). Characters may be deleted from the old copy (D,K,R). Finally characters may be added to the new line from the terminal (I,R). C(hange) operates on the CURRENT line. To get a full explanation of the C(hange) subcommands input a ? while within the change command. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

Change Subcommands:

?	Prints this message.
n SPACE	Space n characters forward copying from old to new.
B	Break line - Insert current contents of new line before the current line. Change subcommands may continue with the rest of the old line, however Q and ^Q can not affect the inserted text.
n D	Delete n characters forward from the old copy.
E	Move to end of line copying from old to new.
I	Insert mode - all input inserted until next CR,LF, or ^Z.
n K x	Delete forward until nth occurrence of 'x'.
n L x	Force alphabetics to be Lower case up to the nth occurrence of 'x'.
P	Print the rest of old line and current new line.
Q	Abort all changes since last B subcommand.
n R	Delete next n characters forward and enter insert mode.
n S x	Space forward to before the nth occurrence of 'x'.
n U x	Force alphabetics to be Upper case up to the nth occurrence of 'x'.
n V x	inVert case of all alphabetic characters up to the nth occurrence of 'x'.
CR	Copy rest of old line to new, update current line to new
LF	Update current line to new, as is
^D	Start C(hange) over, forgetting edits so far
ESCAPE	Copy rest of old line to new, update current line to new, and re-enter C(hange) on the updated current line.

If the 'x' in K, S or V command is a CR, then it is interpreted as meaning "to the end of the line"; if it is ESCAPE, it is interpreted as "use the same character as last time."

G GROUP - Group combines the current line with the following line to produce a single new text line. The current line is set to the new combined line. If this command is executed accidentally, using the "B" subcommand of the C(hange) command can separate the two lines.

X XCHANGE - Xchange takes 2 strings, called OLD and NEW. It then replaces occurrences of OLD with NEW, according to the user's directions. Upon finding a line with an occurrence of OLD, it replaces all the instances of OLD with NEW, types the line on the terminal, and asks the user for confirmation. Typing a "?" to the confirmation request displays the possible options. If the user types just a <CR> to the prompt for NEW, then the occurrences of OLD will be deleted. If <CR> is typed for both OLD and NEW, Xchange uses the same strings it used the last time. If the PRINT XCHANGEs mode is set, all XCHANGEd lines will be printed even if NOCONFIRM (*) is specified. If the search string is a single null character (specified by control-V control-@ [^V^@] or control-^ @ [^^@]), then XED will search for a null line (no text, only <CR><LF>). If this command is input by accident, typing ^Q will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode. This command normally looks for the string any place in the line, independent of upper and lower case differences.

^E If ^E is specified, the match will be exact only.
^B If ^B is specified, the match will be only at beginning of lines.
^F If ^F is specified, XED will do case-sensitive XCHANGEs.

During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

X(change) Confirmation Codes
Y,<space>,CR,LF
Accept printed changes and continue through the text buffer.
N,^H(Backspace),^A,DEL
Reject printed changes and continue through the text buffer.
. Accept printed changes and stop here.
E Reject printed changes and stop here.
`*` Accept printed changes and continue through the text buffer, accepting all changes without requesting confirmation.
If any other character is input, the confirmation request is repeated.

FILE INPUT/OUTPUT COMMANDS

R READ - Read accepts a file name from the terminal and reads the contents of the file into the text buffer by appending the contents AFTER the current line. The old text buffer contents are not changed. Input and output operations to files will be encrypted if the ENCRYPT mode is set. If this command is input by accident, ^Q or DEL will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

E EXIT - Exit accepts a file name from the terminal and writes the contents of the text buffer into this file. When the write is complete the editor exits back to the Executive (@). If the termination is not desired, use the W(rite) command. There is a way to have a document line stating the current file name, time, and person who edited the file in the front of the file each time it is written. See) and (for further details. Input and output operations to files will be encrypted if the ENCRYPT mode is set. If this command is input by accident, ^Q or DEL will cancel it. After this command has returned to the Executive(@), inputting CONT<CR> will continue the editor with no changes to the current line, text buffer etc.

B BACKUP - When a file name is entered to R(ead), W(rite), or E(xit), the editor remembers the name. When executed, B(ackup) writes the text buffer into specific versions of that file name. Normally, B(ackup) writes the buffer into version 2 of the file. If version 2 previously existed, it is renamed to version 1 before the write (the previous contents of version 1 being lost). Thus, repeated uses of the B(ackup) command write over the same files, minimizing the proliferation of new versions of the file. If the SPECIAL BACKUP mode is set, B(ackup) uses the two version numbers numerically following the highest version of the file last read or written instead of versions 2 and 1. This has the advantage that the files created by the B(ackup) are the highest versions of the file and will be referenced when the version number is defaulted (which is almost always the case). To recover the backup copy of the file after a system crash simply read as normal. Read always checks for version 2 backup copies with later write times than the requested versions. There is a way to have a document line stating the current file name, time, and person who edited the file in the front of the file each time it is written. See) and (for further details. Input and output operations to files will be encrypted if the ENCRYPT mode is set. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

W WRITE - Write accepts a file name from the terminal and writes the contents of the text buffer into this file. If the user plans to Q(uit) immediately following the W(rite), the E(xit) command might be more convenient. There is a way to have a document line stating the current file name, time, and person who edited the file in the front of the file each time it is written. See) and (for further details. Input and output operations to files will be encrypted if the ENCRYPT mode is set. If this command is input by accident, ^Q or DEL will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

() PARENTHESES - Each time a file is written by B(ackup), W(rite), E(xit), or L(ist) it is possible to have an extra line added to the top of the file stating the current file name, time, and person who edited it. In addition to this information, there is also facility for a prefix string and a suffix string to be put before and after this information. These latter strings are provided to make these lines into comments for the programs that might process the files. This line will automatically be generated ONLY IF a prefix string has been specified.

(PREFIX - The prefix command accepts a string from the terminal which will be used as the prefix to the automatically generated documentation line when the file is written. The documentation line is NOT added to the text buffer, but is obviously included when the file is next read into the editor. The prefix is required to trigger the automatic document line generation. Inputting (<CR> clears the prefix string and turns off the automatic document line generation. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

() PARENTHESES - Each time a file is written by B(ackup), W(rite), E(xit), or L(ist) it is possible to have an extra line added to the top of the file stating the current file name, time, and person who edited it. In addition to this information, there is also facility for a prefix string and a suffix string to be put before and after this information. These latter strings are provided to make these lines into comments for the programs that might process the files. This line will automatically be generated ONLY IF a prefix string has been specified.

) SUFFIX - The suffix command accepts a string from the terminal which will be used as the suffix to the automatically generated documentation line when the file is written. The documentation line is NOT added to the text buffer, but is obviously included when the file is next read into the editor. The PREFIX is required to trigger the automatic document line generation. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

O OUTPUT - this command writes a formatted output file suitable for printing on a hardcopy device such as a line printer (it is NOT intended for normal file writing!). It can provide titles, page numbers, and a summary of editing changes. It is affected by several modes which determine what kind of headings will appear in the output, how page numbering is to be done, and the size of the output page. These modes are found in the Format/Output mode group of the mode dialogue. When invoked, O(utput) first prompts for a heading (unless the mode affecting the heading specified not to prompt). It then asks for the name of the output file. If a <CR> or <ESCAPE> is specified for the file name, then LPT: is assumed. If this command is input by accident, typing ^Q will cancel it. While this command is executing, typing ^Q interrupts it and returns to XED command mode.

Q QUIT - Quit will exit the editor WITHOUT writing a new copy of the file. After this command has returned to the Executive(@), inputting CONT<CR> will continue the editor with no changes to the current line, text buffer etc.

MODES

" MODE DIALOGUE - Puts the user in an interactive dialogue, allowing the user to set and examine the XED mode settings. It divides the settings into groups of modes, and allows the user to select or skip each group. Within a group, the user then may examine and optionally set any of the modes within the group. At any time during the dialogue, the user can skip the rest of the group selections and individual mode selections, and can also cancel any changes made. It also allows the user to define mode files to contain specified mode settings, and recall them later.

PROGRAM INVOKING COMMANDS

% Call SNDMSG - allows the text buffer to be sent to SNDMSG without going through EXEC or using SNDMSG's ^B option. XED prompts for subject of message, after which user is prompted for To:, cc:, etc., just as in SNDMSG. The message will have the text buffer inserted as the body. It is important not to type ahead while the message body is being inserted, as the input will be merged randomly with the text buffer into the message body. When the message has been sent, XED will return to command mode, with the text buffer untouched. If this command is input by accident, typing ^Q will cancel it. During the input of text for this command the following control characters are active. Detailed explanations are available from H(elp): ^A, ^H(backspace), ^X, ^W, ^R, ^Q, ^V, ^^, ^\, DEL and $(escape).

! Run program - allows the user to call any TENEX subsystem or other runnable program (including the TENEX EXEC) without disturbing the state of the text buffer, current pointer, etc. XED prompts for the name of a program to run (directory defaults to <SUBSYS>, extension to .SAV) and then runs the requested program. When the program is finished running (such as by QUIT to the EXEC), control returns to XED, leaving it in the same state as when the program was called. If this command is input by accident, ^Q or DEL will cancel it.

@ Call EXEC - starts up the TENEX EXEC without disturbing the state of the text buffer, current pointer, etc. When a "QUIT" command is given to the EXEC, control returns to XED.

[Start Fork - starts last program run by a program-running command (!,%,@). If no program has been run, or the last fork was killed by the > command, the user is prompted for the name of a new program as in the ! command.

< Continue Fork - continues last program run by a program-running command (!,%,@). If no program has been run, or the last fork was killed by the > command, the user is prompted for the name of a new program as in the ! command.

> Kill fork - kills the last program run, and its fork. If no program has been run, or it has already been killed, this command has no effect.

FORMATTING

_ FORMAT - The format command allows the user to do limited text formatting and justification on portions of (or all of) the text buffer. If the command is followed by a <CR> or <SPACE>, then the whole buffer is formatted. It can also be followed by a line range specification, analogous to the T(ype) or K(ill) commands. Thus,

_40 means format the next 40 lines in the buffer, starting at the current line

_:125 means format the lines starting at the current line and extending to line 125

[Note that the formatter is an experimental facility and subject to change as its features are used, critiqued and refined.]

Several modes are relevant to the format command (set and examined by the " command). The command is enabled by the ENABLE FORMAT mode (in the Command Enable/Disable group). If justification (adding of additional spacing to even the right margin) is desired, then the Justification mode (in the Miscellaneous group) must be turned on. The margin used by the formatter is also settable by the " command (in the Miscellaneous group). Its default value is 70.

In order to protect the user from performing a format operation which produces a result the user does not expect, the lines which are being formatted are first copied to the text dump. If the user has formatted the whole file, he can retrieve it with the ' (Switch dump) command; otherwise, he can use the J(am dump) command.

For normal operations, the formatter requires no explicit commands. Paragraphs are separated by blank lines. Each paragraph may contain "flags" which direct the formatter to perform indentation on the entire paragraph. The blank lines are retained after formatting, so that subsequent formats of the same text will have the same paragraph boundaries.

The formatter recognizes two special text lines as command directives. The first is a line with only a single, isolated tab. It directs the formatter to turn off formatting until the occurrence of another isolated tab line. This is useful to allow certain pre-formatted information (such as tables) to be skipped over by the formatter. [Note that unintended results can occur if the isolated tab lines are not properly paired!]. The second command is a line with an isolated space. This acts as a paragraph break and directs the formatter to indent the following paragraph the same as the previous one. To help the user recognize the occurrence of these special command lines, XED highlights them (on terminals so equipped) or prints special identifying lines (e.g., << isolated tab >>).

XED PARAGRAPH FORMATTING

Note: [This description has been formatted by the XED format command. Thus it both describes the process and demonstrates the available features.]

I. Normal Paragraph Processing

The first line of a formatted block is treated as a new paragraph. A new paragraph is also created after a blank line (2 consecutive carriage-returns). The paragraph is terminated either:

- the end of the block of lines being formatted
- a blank line
- a line with a single isolated tab

In the latter case, formatting is also turned off until the next line containing an isolated tab is encountered.

A. There are two items in the XED mode file which affect the formatting process. They are the justification flag and the line width.

   1. The justification flag specifies whether the text should be justified to the right margin. If the flag is on, lines will be filled with as many words as can fit, then spaces will be added to even the right margin. This example is justified.

   2. The line width determines how wide the formatted lines will be. (In the case of no justification, this is the maximum length.) This example has the line width set to 7Ø.

B. The normal paragraph processing puts the proper number of words on each line. If justification is requested, each line is also justified. The basic features of this formatting process (besides line-filling and justification) are:

   1. Paragraph indentation is not changed. If the first word is flush to the left margin, the formatted paragraph will also start flush; if the first word is indented five spaces, the formatted paragraph will also have the first word indented five spaces. However, all other spacing characters (space, tab, carriage return, line feed) within the paragraph are simply used to separate words.

      All indentation (whether just the first line of a paragraph or for an entire paragraph) is done with spaces. Tabs are converted into the proper number of spaces.

   2. Hyphenated words are recognized and possibly broken. This has the side effect of possibly adding spaces (after the hyphen) in the middle of hyphenated words in later versions.

   3. All periods, colons, question marks, and exclamation points appearing at the end of words are followed by two spaces (minimum), instead of the one normally used between words.

II. Flagged Paragraph Processing

In addition to normal paragraph processing, the facility also provides for paragraphs which are entirely indented. These are called "flagged" paragraphs.

A. The general form of a flagged (indented) paragraph

1. A flagged paragraph starts with at least one spacing character (tab or space).

2. This is followed by a flag. The flags are described in section III below.

3. Finally, the flag must be followed by at least one spacing character.

* To reiterate, in order for a flag to be recognized, it must be the first word in a paragraph with spacing both before and after. This is done to prevent normal paragraphs from being mistaken for flagged paragraphs. Following are the differences between the processing of flagged paragraphs and normal paragraphs.

B. Flag Formatting:

The flag will be indented in the formatted result the same amount it was indented in the original text. This is similiar to the initial paragraph indenting provided for normal paragraphs. The flag will be followed by two spaces which will not be available for expansion during justification.

C. Succeeding Lines in Flagged Paragraphs:

All lines of the flagged paragraph will line up with the first line of the paragraph and will be indented from both margins.

INDENTED PARAGRAPHS WITHOUT FLAGS

When you want more than one paragraph to line up under a flag, insert a line with a single isolated space between the two paragraphs. The next paragraph will line up automatically with the previous one. This section makes extensive use of this facility.

III. Description of Flags

Flags are used to indent entire paragraphs. There are four kinds:

SINGLES: A single is one of the following: * (star), + (plus), - (hyphen), % (percent), and o (lower-case letter O).

SECTIONS: A section flag can be a single letter, a letter and a number (Ø-9 only), any number Ø-99, or a number (Ø-9) and a letter. It cannot be two letters since a paragraph could start with Mr. Smith or Dr. Jones.

Examples: A,1,A1,22,1a. A section cannot be more than 2 characters.

ROMANS: Any roman numeral between 1 (I) and 399 (CCCXCIX). The roman numerals can be upper- or lower-case.

NAMES: Any string containing only alpha-numerics, . (periods) or - (dashes).

Four classes of flags are recognized by the following puncutation:

A. Any SINGLE by itself (spacing before and after) is considered to be a flag.

B. Any SINGLE, SECTION, or ROMAN followed by a . (period) or a) (close parenthesis) is considered to be a flag.

C. Any SINGLE or SECTION enclosed in balanced brackets is considered to be a flag. Recognized brackets are: <>, {}, [], and ().

D. Any NAME followed by a : (colon) is also considered to be a flag.

EXAMPLE FLAGS

Note: [This section is not formatted. The following line contains a single isolated tab which turns off the formatting process.]

```
Example flags with appropriate punctuation:
                        SINGLE  SECTION ROMAN   NAME
Selected from:

Flag group A:           *
SINGLE ALONE            +
                        -
                        o
                        %

Flag group B:           (*)     [A]
BRACKETED               <o>     (D3)
                        {+}     <7>

Flag group C:           o)      23.     IV)
SECTION DESIGNATOR      *.      2a)     xxxi.
                        +)      Q.      CX)

Flag group D:                                   Flagword:
KEYWORD                                         Case-176:
                                                STATUS.LAST:
```

CONTROL CHARACTERS

Control characters are input by depressing the CONTROL key (which is a shift key and by itself does not transmit any code to the computer) followed by the appropriate character. Control characters are represented as an ^ preceding the character. E. G.: ^A represents Control A.

INTERRUPT CONTROL CHARACTERS

These characters may be input at any time. ^C and ^T are available anywhere in TENEX, while the others are only supported by XED.

^C This stops the editor and goes immediately to the Executive(@). The editor may be continued by inputting CONT<CR>.

^T This will print the current load average of the system.

^N This will print the current line number. This is useful for tracking the progress of R(ead), W(rite), B(ackup), E(xit), F(ind), and S(earch).

TEXT INPUT CONTROL CHARACTERS

These characters are available whenever the user is inputting a text string from the terminal.

^A ^A, ^H(backspace), DEL(rubout), will all delete the last character input. DEL(rubout) additionally aborts commands which accept file names. In this file name situation DEL is active during R(ead), W(rite), E(xit), H<CR> [Manual Request], and when B(ackup) requests a file name because not file name has yet been specified.
This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^W ^W deletes the last word input. Words are delimited by tabs and spaces. ^W also deletes and spaces or tabs which might have followed the last word input.
This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^X X deletes the current line being input from the terminal.
This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^R ^R re-types the current line being input.

This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^Q This will stop an executing command. It can stop B(ackup), F(ind), L(ist), O(utput), P(rint text dump), R(ead), S(earch), T(ype), V(iew), W(rite) and X(change).

^Q is also active whenever a command or text is being input. If typed while entering the number parameter to a command like K(ill) or T(ype), it will abort the command. If typed while Inserting or Appending, the current text line will be discarded and XED will go back to command mode.

^V ^V "quotes" the next character input. It is used to input characters which would normally cause control or editing functions (e.g., ^Z, ^X, ^Q, ^B). For example, to put "ABC^Z" in a text line, one would enter ABC^V^Z. ^V would be entered into text by ^V^V.

This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^^ ^^ "control shifts" the next character input. (e.g. ^^C inputs ^C, while ^^c inputs).

This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

^\ ^\ "case shifts" then next character input. (e.g., ^\C inputs c, ^\c inputs C, while ^\ inputs ^C).

This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

SEARCH MODIFICATION CONTROL CHARACTERS

These characters cause special actions to be performed during the execution of the certain commands. To enter them as text for these commands, they must be "quoted" by ^V or ^^.

^B ^B specifies that Find, Search, and Xchange must match only at the Beginning of a text line. (The default is to match anywhere in the text line). The ^B may be typed either before the command it affects (F,S or X), or during the typing of the string affected. If this command is entered more than once, it reverses the effect of the previous one (or the prevailing default setting), i.e., it acts as a toggle.

^E ^E specifies that Find, Search, and Xchange must Exactly match the text string, including case. (The default is to match exclusive of case). The ^E may be typed either before the command it affects (F,S or X), or during the typing of the string affected. If this command is entered more than once, it reverses the effect of the previous one (or the prevailing default setting), i.e., it acts as a toggle.

^F This command affects the X(change) command. When specified, it causes XED to do case-sensitive XCHANGEing. In this mode, XED categorizes the found string into one of four classes (UPPER CASE, lower case, Capitalized, or aRbitrAry). When it substitutes the replacement string, it maps it into the same class as the found string (if the found string is arbitrary, XED makes no change). The ^F may be typed either before the X(change) command, or during the typing of the OLD string. If this command is entered more than once, it reverses the effect of the previous one (or the prevailing default setting), i.e., it acts as a toggle.

^S This command affects the searching commands (F,S). When specified, it causes XED to enter Change for the line within which the string was found, positioning the Change cursor at the found string. The ^S may be typed either before the command it affects (F,S), or during the typing of the string affected. A mode in the Miscellaneous mode group allows the user to specify that the C(hange) cursor be positioned at the end of the found string. Normally, it is positioned at the beginning. If this command is entered more than once, it reverses the effect of the previous one (or the prevailing default setting), i.e., it acts as a toggle.

$ ESCAPE TO CHANGE - During the insertion of text lines, the escape character passes the line currently being input to C(hange). When C(hange) is terminated in the standard way, the line input process continues. If escape is typed when the text is empty, the text line will first be initialized to the LAST string entered in the current context, e.g., if the string is being entered for a FIND command, the string used in the last FIND/SEARCH command will be used; if the string is being entered into the text buffer by an APPEND, the last string entered during the last APPEND/INSERT command is used. This control character is active during A(ppend), I(nsert), C(hange), F(ind), S(earch), X(change), ((set prefix),) (set suffix), and % (SNDMSG).

ZTYPE

ZTYPE is a program for listing text files (RUNOFF output files, for example) on a 300 baud terminal such as Diablo, Anderson Jacobson, Bedford, etc.

Example of its use:

@ZTYPE<space> (File) afilename (Starting at page) a page At this point ZTYPE will output a formfeed and ring the bell. It then waits for the user to type a character (signifying the paper is ready in the terminal). The user typed character will not be echoed on the paper. If you wish to start at page 1 of the file, you can end the file name with a carriage return. Typing carriage return to the page number request will also assume page 1.

Whenever ZTYPE rings the bell at the end of a page, you may type one of the following "special" characters:

"X" - This stops ZTYPE and returns you to the EXEC

"P" - Position to a page number. Type "P" followed by the desired page number

"B" - Starts over at the beginning of the file.

Any other character simply tells ZTYPE the paper is ready and to type out the next page.

None of the typed in characters will be echoed on the terminal.

Insert feature: It is possible to type in text on-line while ZTYPE is running. Whenever a control-F is found in the text file, ZTYPE will stop and echo any characters you type in. When a control-F is typed, ZTYPE will continue listing the file.