TENEX USERS' GUIDE

Bolt Beranek and Newman Inc.

50 Moulton Street

Cambridge, Massachusetts 02138

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

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) 491-185Ø Ext. 484, or 617-491-6169 for a direct line to the Research Computer Center.

TENEX SUBSYSTEMS

HACKS

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].

```
#       MTA0:,DSK:SIM26=SIM26,PARAM.TST
#
# [read source from DSK:SIM26.ALG, DSK:PARAM.TST, write relocatable
# binary on device MTA0, 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/1000D=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 1000 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 026 keypunch or equivalent. The program first asks:

MAGTAPE UNIT NO.=

to which the user replies "0" 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(200 OR 800):

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

TO OR FROM MAGTAPE?(T OR F):

if the direction is from 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 026 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 to 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 (Ø 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 Ø (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	40	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 -Ø).
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 (1,m,n) where 1 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
```

```
TUG
TENEX CALENDAR SUBSYSTEM


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 DECsystem1Ø 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-1Ø 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.