A new version of the summary documents is being prepared for distribution. I would appreciate it if you could look this over and give me any suggestions ~~before~~ by Friday, as I would like to "publish" it next week.

I realize that part 1. will require rewiritng whenthe permanent disk directory structure is installed, but theother sections are thought to be useful after the trauma wthout revision. Please bring counter-indications to my attention.

Vance

Summary documents 66r using the user version of CAL TSS

```
USER ERROR
000006, 000003, 00000000000000 ERROR

BOO, BEAD GHOST HERE
@                                          )
```

Internal version, 26 May '71

Idiot's Guide for Running on the Demonstration Version of CAL TSS

I) Words of warning:

1. This is a demonstration system; there are known (and unknown) ways to crash it.
2. There is no way to preserve files on this system once the user has logged out or in the event of a system crash.

II) Time-sharing in four easy steps:

1. The teletype must be turned on and switched to the connection for the B machine; ask somebody if in doubt.
2. Attract the attention of the system by typing CTRL-SHIFT-P (hold down the buttons marked "ctrl" and "shift" and hit the letter "p").
   If the response is:

   ```
   CAL TSS VERSION 1.0
   NAME YOUR PERMANENT DIRECTORY
   .
   ```

   proceed; if the response is anything else, find expert advice.

mention NOSPACE

3. General information on dealing with the keyboard:
   a. All input goes to a piece of software called the line collector, which allows certain manipulations of the input. A few commonly useful features are listed below. (Appendix B contains a more complete description.)
   b. Input lines are terminated by the "return" key (no line feed).
   c. Typing CTRL-Q erases the previous character entered.
   d. Typing CTRL-Y erases all characters in the current line.
   e. Typing CTRL-I skips to the next tab boundary (cols. 11, 21, ...).
4. A typical run:
   Back in step 2 the system asked you to name your permanent directory; this means that it is ready for you to log on.
   a. Logging on checks you into the system:

   ```
   Type:       GUEST
   Response:   GIVE PASS WORD
               .
   Type:        GUEST    (on the same line following the period)
   Response:   ENTER TENTATIVE NAME FOR TEMPORARY DIRECTORY
               .
   Type:       any 7 or fewer alphanumeric characters which you feel will
               uniquely identify you (again immediately following the period).
   Response:   COMMAND PROCESSOR HERE
               !
   ```

   If some other response appears, such as: ERROR OCCURRED ON CALL TO CMMDS followed by several lines of junk, try another name; the one you chose was already taken.

   b. Doing your thing: You may now use BASIC, or any other subsystem which happens to be active. A sheet is available which describes BASIC, an interactive language. Use of other currently available subsystems generally involves the following steps:
      i. make or update a text file with the Editor (a sheet describing the Editor is available).
      ii. call the subsystem which is supposed to operate on the text file, for example, the SCOPE Simulator.
      iii. if errors, scan the result file with the Editor to locate the errors, and then go back to step i.
      iv. if no errors, print the result file with the Editor.
      v. repeat as patience and curiosity allow.

   c. When finished, ~~type: LOGOUT~~ get back into the COMMAND PROCESSOR and type "LOGOFF

Summary of the Editor

The Editor subsystem enables the TSS user to construct and edit files of coded information. A file consists of lines of coded characters ending with a carraige return character (generated by the RETURN key on the teletype).

The Editor is called by typing a command of the form:

```
        EDITOR fname
```

where fname is the name of the file to be created and/or edited. The Editor will respond by typing EDIT and awaiting a request. At any given time the Editor is looking at a specific line called the current line. When the Editor is first called, the current line is a pseudo-line which is always the top line of every Editor file.

The following requests may be typed to move about the file for the purpose of creating, deleting, or editing text lines. Each request is terminated either by a carriage return or, if more than one request is made on one line, by a semi-colon. Some requests contain a "stop condition" or line specifier, represented by sc below. Such requests affect all lines from the current line to the line specified by sc, inclusive. (If you've lost track of the current line, request "P" and the Editor will print it.) sc may be :

1) a decimal number, specifying the line that number of lines from the current line
2) ".str" (where str is any string of characters except semi-colon), specifying the next line containing that string of characters
3) "/str", specifying the next line starting with the given string of characters, ignoring leading blanks
4) "$", specifying the bottom, or end, of the file
5) omitted, specifying the current line.

After the Editor has processed the request, the line specified by the request becomes the new current line.

Requests	Meaning
I	Insert, after the current line, the lines which follow. Insertion is ended by entering a null line.
Dsc	Delete the specified lines.
T	Move to the top of the file (pseudo-line).
Msc	Move forward over the specified lines.
Bsc	Move backward the specified number of lines. (NOTE: sc can only be a number.)
Psc	Print the specified lines.
C/str1/str2/sc	Replace the first occurrence of str1 by str2 in the specified lines.
CG/str1/str2/sc	Replace every occurrence of str1 by str2 in the specified lines.
Esc	Edit the specified lines using the line collector.*
R,fname,uname	Insert the contents of the file fname,uname after the current line.
W,fname,uname,sc	Locate (or update if necessary) the file fname,uname and write into it the specified lines, including the current line.
F,fname,uname	Finished - create the file fname from the latest version; simply entering "F" causes the updated text to replace the original file specified when the Editor was called.
Q	Finished but do not save any file.

The Editor prompts with : . When it gets an incomprehensible command, it answers "????".

* Each line being edited is made the old line in the line collection and may then be altered using the line collector. (See Appendix B on the Line Collector.)

Summary of BASIC

BASIC is an easy-to-learn, general-purpose programming language similar to FORTRAN but created specifically for time-shared computing environments. For details see the description in the CAL Computer Center Users Guide.

BASIC accepts three types of statements: 1) indirect, which are saved to be executed sequentially as a program at some other time; 2) direct, which are carried out (executed) as soon as they have been entered using the carriage return key (direct statements, especially the PRINT statement, allow the teletype to be used as a very powerful desk calculator); and 3) Editor requests, which instruct the computer where and how to save the indirect statements as well as how to change (edit) them if necessary (see Editor document).

Although some statements may be used only directly (or indirectly), most statements may be used either way. All indirect statements must begin with a line number and are executed in order of ascending line numbers.

BASIC is called by typing a command of the form:

```
      BASIC
```

It responds with BASIC HERE after which either direct statements or a program of indirect statements may be entered.

BASIC prompts with : and types ???? in response to lines it does not understand.

There are two ways to enter a program of indirect statements:

1. Creating a new program file
   - a. type I
   - b. type the statements (if BASIC responds with ERROR...the line was not entered into the file and may be retyped).
   - c. enter a null line (cr only) to end the file
2. Reading an already existing program file called fname
   - a. type R,fname
   - b. any lines containing errors will print. Corrected versions may be inserted using ~~the~~ Editor requests after BASIC prompts.
   - c. BASIC prompts with : when the file is in.

Sample BASIC program starting from the Command Processor:

```
BASIC
BASIC HERE
I
100 PRINT "NUMBER", "SQUARED", "CUBED"
105 PRINT
110 FOR X=1 TO 10
120     LET S=X*X
130     PRINT X,S,X*S
140 NEXT X
150 END
empty line (cr only)
RUN

NUMBER          SQUARED          CUBED

 1               1                1
 2               4                8
 3               9                27
 4               16               64
 5               25               125
 6               36               216
 7               49               343
 8               64               512
 9               81               729
 10              100              1000
```

Now the user may:

1. Edit his program using Editor requests and rerun it.
2. Quit (and return to the Command Processor) by typing Q.
3. Quit and save the program on file fname (and return to the Command Processor), by typing F,fname; the program will be available for further manipulation.

List of Indirect or Direct Statements

LET var=[...var=]expr Each variable[1] takes on the value of the expression.
Example: 10 LET A=B=4.35-F

DIM array(dim list)[...,array(dim list)] Reserve space for arrays with more than two dimensions and/or dimensions > 10.
Example: 20 DIM A(60),L(5,N,3*N)

[1] A variable may only be a letter optionally followed by a digit, or by a list of expressions separated by commas and enclosed in parentheses.

SIG expr Number of significant digits printed for numbers is changed to the value of expr. Ex. 30 SIG N

DEF FN letter(param)=expr Defines a one line function whose name has three letters starting with FN and whose single dummy parameter is param. Example:
35 DEF FNG(X3)=X3/10 - A0/X3

READ var[...,var] Reads from a DATA defined list and assigns values to the variables in a sequential order. Example: 40 READ A,B,G2

INPUT var[...,var] Requests input values from the TTY by typing ? and assigns values to the variables in sequential order.
Example: 12 INPUT A,B,C

PRINT [... item] Prints and/or moves the teletype head as indicated by the item(s) which may be num expr, string var, "characters", TAB(expr), ,, ;, and :.
Example: 100 PRINT "VALUE +", TAN(B1*B1)

RESTORE Restores the pointer into the DATA bank to the top.

IF log expr GOTO lnum
IF log expr THEN lnum Transfers control the statement with line number lnum if the logical expression is true.
Ex. 105 IF A B/SIN(X) GOTO 115

GOTO lnum Transfers control to line number lnum. Example:
20 GOTO 300

ON expr GOTO lnum[...,lnum] If expr has value=1, GOTO statement having first lnum in list; if expr has value=2, GOTO statement having second lnum in list, etc. Ex.
10 LET X=1
20 ON X GOTO 30,40,50 transfers to statement 30.

REM char string A comment statement.

GOSUB lnum Go to the statement specified by the line number but return to the line following the GOSUB when a RETURN statement is encountered.

MAT READ c Reads values from DATA list into array c.
MAT PRINT c Prints values from array c.
MAT c = TRN(a) Matrix c becomes transpose of a.
MAT c = ZER Zeros every element in matrix c.
MAT c = IDN Square matrix c is set to identity matrix.
MAT c = CON Array c is set to all ones.
MAT c = a+b Array c is set to the sum of a and b.
MAT c = a-b Array c is set to the difference between a and b.
MAT c = (expr)*b Array c is set to the scalar product of expr and b.
MAT c = INV(a) Matrix c becomes the inverse of a.

List of Indirect Statements

DATA val[...,val] Forms a list of data values to be used by READ statements. Ex. 12 DATA 5,7.3,3E + 52

PAUSE[str] Execution pauses and str, if given, is printed. BASIC will accept direct statements or editing requests; execution resumes if CONTINUE is entered.

END Ends execution; must have highest line number.

STOP Stops execution (acts like a jump to END statement).

FOR val=expr TO expr[STEP expr]
NEXT var Defines the limits of a loop. The three expressions give the initial value of the control variable, the terminating value and the increments, if not equal to 1.

Example: 40 FOR I=1 TO 10 STEP .5
50 LET S=S+I
60 NEXT I

RETURN Execution goes to the line following the last GOSUB for which no RETURN has been executed.

List of Direct Statements

LIMIT integer Specifies a maximum number of statements that can be executed without control returning to the console; prevents infinite loops.

RUN Causes execution of the program beginning with lowest line number.

CONTINUE Execution continues where it last stopped.

Operators

Arithmetic	
↑	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

Relational	
=	Equal
< >, ><, #	Not equal
<	Less than
<=, =<	Less than or equal
>	Greater than
>=, =>	Greater than or equal

Logical

!	Logical OR
ε	Logical AND
NOT	Logical NOT

Functions

ABS(X)	$\lvert X \rvert$	LGT(X)	$\log_{10} x$
ACS(X)	arcos(x)	RND(X)	random num
ASN(X)	arcsin(x)	SGN(X)	sign(x)
ATN(X)	arctan(x)	SIN(X)	sin(x)
COS(X)	cos(x)	SQR(X)	$\sqrt{x}$
EXP(X)	e^x	TAN(X)	tan(x)
INT(X)	integer	TIM(X)	seconds used
LOG(X)	$\log_e x$		

Summary of SCOPE Simulator

The SCOPE Simulator provides an operating environment for programs written for CAL's 6400 batch system (SCOPE 3.0 or CALIDOSCOPE) and permits real-time control over the construction and execution of such programs by a user at a TTY.

The SCOPE Simulator is called by typing the following command:

```
SCOPE
```

The Simulator responds by typing the time and date. SCOPE prompts by typing >. Any legal SCOPE Simulator request or CALIDOSCOPE control request may be entered following the > sybmol. (See summary below.) Code running under the simulator which interacts with the TTY prompts with ↑.

SCOPE creates several standard files necessary for its operation whenever it is called, notably a SYSTEXT file called "OUTPUT". When it needs some other file in order to process a request, it tries to get it from the user's Temporary Directory; if there is no file by that name in the TD, it creates a null file and uses that. The "GET" request gives some flexibility in file usage.

SCOPE Simulator Requests:

Request	Meaning
TEXT,fname	Declare a new SYSTEXT file.[1]
FILE,fname	Read requests from the file fname.
MSG,OFF or ON	Suppress program messages to teletype or restart them.
GET,fname, uname	Get fname from the C.P. using implicit scan list
PUT,fname	Return fname to its directory.
STEP	Trace RA+1 calls made by program on Simulator. The Simulator will print each RA+1 call in octal and then await response: B call BEAD debugger S perform request E ignore request and perform END instead G leave STEP mode; then perform request
FIN	Exit from Simulator.

Loading Requests:

~~L,fname~~	~~Load the file fname and GO~~
LGO,fname	
LDCTL,TSS	Set TSS mode for the loader: common blocks for each file loaded are allocated after all program blocks.
OVERLAY,fname	Contents of loaded and linked core (without banner words) are written onto file fname.

CALIDOSCOPE Control Requests:

COPY
COMPARE
COPYL
COPYN
COPYSBF
CATALOG
CPC
DMP
REWIND
RFL
RUN
COMPASS
UPDATE
SNOBOL

Library Programs:

REGDUMP
DEBUG
SETPRU
KOMMON
MEMORY
CFIO
TRACE
IO
IORANDM

All RUN FORTRAN Library Routines

[1] will not change a file which already exists in another mode to SYSTEXT.

untangle a TTY from an unknown & annoying state

TTY in unknown state: APPENDIX A - How to app ~~return to the~~ COMMAND PROCESSOR

In order to call subsystems, the user must be in the COMMAND PROCESSOR, which is the "ground state" of the process watching his teletype. If he forgets what he is doing, or gets lost ~~inherits a teletype in some unknown state~~, the table below explains how to tell what subsystem is in control and how to get back to the COMMAND PROCESSOR. Procedure:

1. If there is a prompt character printed by the teletype, check which subsystem uses that character.
2. If not, ~~enter a null line~~ type WHO CR and observe the response:
   a. If there is a response, it may identify the subsystem, or else the user should be able to identify the subsystem from the table.
   b. If there is no response, he should not try to use that teletype without getting expert advice; it may be blown up or it may be involved in a remote function such as printing. (this assumes that the TTY is connected & logged in, etc.)
3. Having identified the active subsystem, the user may dismiss it or proceed.

SUBSYSTEM	PROMPT	RESPONSES TO INCOMPREHENSIBLE INPUT OR ERRONEOUS INPUT	HOW TO DISMISS IT (Commands are underlined below)
COMMAND PROCESSOR	!	BAD SYNTAX or UNEXPECTED F-RETURN DURING COMMAND... + possible other lines or UNEXPECTED ERROR IN COMMAND PROCESS... + possible other lines or ERROR OCCURRED ON CALL TO COMMDS + other lines	when finished, LOGOUT when not finished, you may call any available subsystem;
SERVICES	*	same as COMMAND PROCESSOR, except the message says SERVICES	FIN
BEAD GHOST (debugger)	@	same as COMMAND PROCESSOR, except the message says BEAD GHOST	to return to COMMAND PROCESSOR, PURGE to return to subsystem which made error originally, RETURN or RETRY
EDITOR	:	????	F or Q (see EDITOR document)
BASIC	:	???? or miscellaneous diagnostics relevant to erroneous BASIC statements	same as EDITOR
SCOPE	> ↑ (see SCOPE)	??NO??	FIN

TTY in known caps but undesirable state: For example, you are working on a 2000 line file with the Editor and have just typed "P/BLETCH" instead of "P/BLOTCH" and the Editor is dutifully printing all 2000 lines instead of the 3 you had intended. Or you have just fired up your shiny new BASIC program and it has been calculating for 3 minutes when you know it can't possibly take that long (you are stuck in an endless loop).

You can always interrupt the program which is currently running by sending a "panic" from your TTY. This is done by sending a CTRL-SHIFT-P. Civilized subsystems handle these panics on their own and print out some sort of (hopefully) useful message after being interrupted and then allow the user to proceed. Other subsystems are suspended wherever they happened to be and a message:

```
PANIC
BOO, BEAD GHOST HERE ?
```

comes out on the TTY. The caps Bead GHOST is the system debugger, and you can tell it to dismiss the subsystem which had gone astray by typing PURGE, after which the command processor should respond. Work that was in progress by the subsystem that you interrupted will have been cavalierly aborted and things may be in a fairly muddled state -- files might be half-modified or half-generated for example.

UNEXPECTED APPEARANCES OF THE BEAD GHOST: Scenario: You were running along happily in BASIC or the Editor and you just said M/100 and got a response:

```
USER ERROR
~~6,3,0 ERROR~~ → 000006, 000003, 0000000000000 ERROR
BOO, BEAD GHOST HERE
@
```

(The example is not chosen at random; the 6,3,0 error is the only one for which I can give you a remedy.) Whenever a subsystem makes an error in dealing with other subsystems or system-maintained objects, error processing is initiated. Some errors are handled automatically by various subsystems along the way and the user at the TTY is not even aware of them. Many are reported on the TTY by a given subsystem (~~CP~~ the COMMAND PROCESSOR is an outstanding example) to indicate that they were asked to do something illegal. Some represent unforseen circumstances for which no remedial procedures have been provided (called "bugs" for short) and they are reported to the user at his TTY by the BEAD GHOST (system debugger) in hopes that he knows what to do (like, complain to a system programmer). The 6,3,0 error is the only error (as of this writing) which the BEAD GHOST should report to the TTY under normal circumstances. It means that the amount of space allocated to the user has become insufficient for the job currently being done and more space will be required before the job can continue. ~~See Appendix C on the Space Command for further details.~~

APPENDIX B - THE LINE COLLECTOR

Unless the user does something extraordinary, all input to a TTY goes through a piece of software called the LINE COLLECTOR. The LINE COLLECTOR provides a large number of ways to correct/change the line being entered. The chart below indicates the various manipulations that can be performed; to invoke a given function, hold down the CTRL key and hit the relevant key. A detailed explanation is available in the "Users Guide", sec. III.2.3. Here we give two examples and encourage the user to experiment. Underlined characters represent one key or a combination of keys, not the sequence of keys given by the individual underlined characters; blanks that might otherwise be "invisible" are also underlined.

First note that the LINE COLLECTOR maintains the previously typed line as the "old line" and uses it, in conjunction with typed characters, to construct a "new line". Whenever the new line is accepted (by typing CR, for example), it becomes the old line.

You are talking to BASIC and have just entered the line (considered as the "old line") below (which will have provoked a message from BASIC objecting to the line)

old line: PRNIT X

type	meaning	and the teletype responds
CTRL-L	make an insert at the beginning of the "old line"	no response
10_	this is what is to be inserted	10_
CTRL-O	copy the rest of the "old line" (all of it) into the "new line" and accept the "new line".	PRNIT X and the carriage will return.

BASIC will issue another diagnostic as it still will not recognize the line as a valid statement.

old line: 10 PRNIT X

type	meaning	and the teletype responds
CTRL-D	copy the "old line" into the "new line" up to the first occurrence of the next character typed	no response
N		10 PR
IM	you wanted IN and made a mistake	IM
CTRL-Q	erase the M	←
N		N
CTRL-H	copy the rest of the "old line" into the "new line"	T_X
,Y	you remembered to print "Y"	,Y
CR	you are satisfied with your "new line"	and the carriage will return

BASIC should accept this line, which is

old line: 10 PRINT X,Y

6/71

APPENDIX C - Space Control

caps

~~The~~ Cal TSS has several types of storage for which there is currently no automatic algorithm to share the available space among the users. The only positive thing to be said for the scheme described below is that it is better than simply handing out space until it is all gone and then letting the system grind to a halt (or crash).

There are four types of space, arranged in a hierarchy:

TYPE	NOMINAL	MODERATE LIMIT	MAXIMUM
1) swapped ECS space (highest type)	7000	34000	100000
2) fixed ECS space	2000	?	?
3) MOT slots	not concurrently controlled		
4) temporary disk space (lowest type)	"	"	"

When a user logs on, he is allocated the nominal amount of space of each type. A command is available to obtain space in excess of this amount. If a user requests an amount of space larger than what is currently available, he is put into a queue waiting for someone to release space. If the request is for more space than the moderate limit, he is put in a special queue which prevents more than 1 user at a time from being "very large" in any particular type of space.

There is currently no mechanism to ~~make~~ force a user to release space once he has it. Several mechanisms tend to prevent space hogging. ~~currently.~~ First, whenever a user returns to the command processor, he is automatically reduced to nominal. Last, a user who has space over the nominal in some category is not allowed to get more space in that or any higher category without first releasing his space and going to the back of the queue.

The space command works as follows and may be typed to the BEAD GHOST or to SERVICES:

```
SPACE P1 P2 P3 P4
```

P1 through P4 are the amounts of swapped ECS space through temporary disk space, respectively, that are desired. The following algorithm is executed for each parameter starting with P4:

```
if = -1  ~~as much~~ space of this type is released to get down to nominal if possible
if =  0  ignored
if >  0  1) If space above the nominal for that type or higher type has been
            obtained, error.
         2) If parameter is higher than maximum permitted for this type, error.
         3) If parameter, greater than moderate limit, enter very large queue.*
         4) If parameter less or = nominal, no further action.
         5) Otherwise, accumulate this type of space until the amount this user
            has is up to the size of the parameter, waiting in queue if necessary.*
```

* A message will print if the space is not immediately available - ~~CSP~~ a panic (see sec 1.10) will remove you from the queue if you would rather not wait.

There are two different starting points from which you may find yourself requesting space:

1) You are about to call a subsystem and you know in advance how much space it will require: enter SERVICES and request the required amount of space and then go back to the CP and call the subsystem. The request has to be big enough - see below!

2) A subsystem you have called runs out of space and makes a 6,3,0 error which invokes the BEAD GHOST: if you have not already requested space, you may do so now with the space command. After you have gotten the space, type RETRY (do not type RETURN) and the subsystem will resume. If you already have space, there is no way to save yourself - you must type PURGE, which erases whatever work the subsystem may have done for you, and start over in the Command Processor.