July 1971

INTRODUCTION TO CAL TSS

July 1971

PREFACE

This document is intended to provide inexperienced users with quick and easy access to many CAL TSS facilities. It is not intended to be logically complete or fastidiously accurate.

The first part gives a brief description of the logical structure of the system as seen by the user. The second part is a collection of examples of some useful interactions. The examples provide a cookbook approach which may be adequate for some users, and it is hoped that the section on general concepts will be helpful in easing the user into productive and flexible use of the system. However it is doubtful that these pages will answer all questions or transform someone with no previous experience into a proficient user without some work.

Fortunately, one need not be an expert to use the system. One of the advantages of interactive systems is that the user can "try it and see if it works" without incurring a prohibitive cost in money or time. Thus, a light reading of this document should be more than enough to prepare the user to start experimenting on the system itself. Of course, having assistance from someone who knows CAL TSS is very helpful. But in the absence of expert advice, going back and forth between the examples, the console, and the description of general concepts is hopefully a reasonable route to expertise.

The third section gives brief summaries cf the subsystems available on CAL TSS. These summaries are not intended to teach people how to use the subsystems. Rather, they are intended as convenient "crib sheets" for people who already know how to use them.

1.1 Access to CAL TSS

To use CAL TSS, one must satisfy two requirements. The first is to make arrangements with the Computer Center accounting office, or a TA, or some such authority who has time to dispense. He will provide the name of a permanent directory which will pay for use of the system, and a password, which will verify the right to use that directory. The second is to have access to a teletype (or other teletype compatible terminal), connected to the 6400 B system. It is assumed that the reader has access to such equipment and knows how to operate the equipment itself. Below are noted a few useful features of keyboard input to CAL TSS:

a) input lines are terminated by the RETURN key (no line feed)
b) typing CTRL-Q erases the previous character entered
c) typing CTRL-Y erases all characters in the current line
d) typing CTRL-I skips to the next tab boundary (cols 11,21,...)

1.2 Files and Directories

Files are system-maintained objects in which a user can keep information (source code, programs, data, etc.). In particular, when a user is not active on the system, virtually all the information he wants to keep is stored on the disk in files. Directories keep track of the names and locations of all the files in the system, plus various other information. Each user has his own directory which keeps track of his own personal files and contains information pertaining to him. This directory stays on the disk when the user is not active and is called the user's permanent directory to distinguish it from other directories which are described later.

1.3 Login, logout

The process of making contact with CAL TSS is called LOGIN. The user tells the system he is present by typing CTRL-SHIFT-P on the console. The system then starts to construct the machinery necessary to give him access to his files and to the various subsystems available to manipulate files. Nominal amounts of system resources are reserved for him. This nominal amount is sufficient to run a small BASIC program or to use the EDITOR to modify a text file. The console responds by asking the user to name his permanent directory and to prove that he is authorized to use it by giving the password.

A temporary directory is then created to hold the files that come and go as he uses the system. The console asks him to name his temporary directory. Since this name will be used globally across the system, it must not be the same as someone else's temporary directory (if it is the same name as another's, the user is then asked to choose a different name). The appearance of the Command Processor signals successful completion of the LOGIN procedure.

The temporary directory and any files which it owns will be destroyed when the user finishes using the system and logs out. It is easy to logout: simply get into the Command Processor and type 'LOGOUT' (see examples).

Note that once the user has successfully logged in, he starts being charged for the resources necessary to be active on the system. This charging will stop only after LOGOUT (not when the console is turned off).

1.4 Command Processor, subsystems

When the LOGIN procedure is completed, the user will be talking to the Command Processor. The Command Processor does not do many things for the user itself, rather, it accepts commands to set up various subsystems to work for him. Some standard subsystems which are always available on the system are introduced in Table 1. A user may also code and call (through the Command Processor) his own subsystems. The exact method of doing this is not described here.

Table 1

SUBSYSTEM NAME	WHAT IT DOES
EDITOR	prepares and modifies text files.
BASIC	Prepares and runs programs in the BASIC language.
SCOPE	simulates most of the functions provided by the operating system which runs batch jobs on the A machine; gives access to the FORTRAN, SNOBOL, and COMPASS languages, and executes programs compiled with them.
BCPL	a programming language aimed at non-numeric applications.
PRINTER	prints files on the line printer.
SERVICES	manually manipulates user's files and directories.

The Command Processor and all the subsystems print some character at the beginning of the line when they are ready to accept a command. This is called a prompt character. A table in section 1.9 shows the different prompt characters for all the system-provided subsystems. After the Command Processor prompts, the user might tell it

```
        !EDITOR INPUT
```

intending to edit a file called 'input' (the ! at the beginning of the line was typed by the Command Processor, not the user). A general example of the form of commands accepted by the Command Processor is

```
        !command param param ...  param
```

where *command* and *param* are strings of characters separated by spaces. How the Command Processor turns the characters at the console into internally meaningful information is a long story, which is introduced next.

1.5 *Names, objects, name spaces, access locks, access keys*

When the user types

```
        !EDITOR INPUT
```

to the Command Processor, 'EDITOR' and 'INPUT' are examples of what are called *names* in this document. The handling of both these names makes use of the concept of *name space*. The trick is to turn a string of characters into some internal form which will give access to a file or a subsystem. A name space can be thought of as a dictionary which translates a string of characters (name) into the required internal form. There are several different types of internal forms all of which are referred to as *objects*. Files and directories are examples of objects. A directory contains the names of objects and also information about those objects. Thus, one form of name space is a sequence of directories to be searched in turn for the given names.

Another important concept in changing names into objects is that of an *access key*. A given name in a directory may be shared by having an *access lock* attached to it. In order to get access to the named object, an *access key* must be presented along with the name. Access locks not only control whether or not access is permitted, but also what kind of access is permitted. Thus, a given file name in some directory may be protected with two different access locks such that when it is looked up with one key, the file may only be read from, while it may be read, written, or destroyed if it is looked up with the other key.

The most common form of name space is a sequence of pairs (directory, access key). The scope and power of a given name space are determined by what directories are searched and what access keys are used.

There are several different name spaces attached to each user, and different ones are used in different circumstances.

1.6 Command Processor name space, BEAD name space, SCANL name space, PERMDIR, TEMPDIR, PUBLIC, OWN.KEY, null key, PUB.KEY

The first parameter typed to the Command Processor is looked up in the command processor name space (see Table 2). PERMDIR is a name used to refer to the user's permanent directory. TEMPDIR is a named used to refer to his temporary directory. PUBLIC is the name of a directory which contains the names of all system-provided subsystems. For example, it contains the name 'EDITOR'. If the user has just typed:

```
        !EDITOR INPUT
```

the Command Processor is guaranteed to find the name 'EDITOR'. Having found the object named EDITOR, the Command Processor assumes that the object is a file which it can use to construct the EDITOR subsystem. It procedes to do this. Note that if a file named EDITOR were in the user's temporary directory, the Command Processor would find that file because it searches TEMPDIR first. It would then try to start up a subsystem constructed from the user's file, which is fine if the file contains the user's own private version of the EDITOR. Otherwise, an error results. It is always best for the user to know what he is doing before he tries it.

The interpretation of the parameters after the first one is dependent on the subsystem being called; each subsystem specifies the name space it uses to evaluate parameters. The three possible names spaces are shown in Table 2. The BEAD name space is an old form left over from previous incarnations of the system. It is being phased out. The SCANL name space is initially as shown in Table 2, but the user may modify it to suit himself.

Much of the complexity of the name space situation stems from considerations about the sanctity of permanent files (owned by the permanent directory) and the reliability of subsystems. Consider the nature of the files in the user's permanent directory as opposed to the nature of the files in his temporary directory. Many subsystems use temporary or scratch files which are not of interest to the user. These files come and go in TEMPDIR without troubling the user. They automatically disappear when he log outs. Free access to these files is essential to the operation of the various subsystems. Presumably it is no great loss if a subsystem runs wild and a temporary file gets clobbered. PERMDIR, on the other hand, gives access to the user's permanent disk files. The user would be justifiably annoyed to discover that one of his files had been used as a scratch file by some subsystem. There is no automatic backup of these files. If some subsystem has access to a user's files and uses one for scratch or goes wild and destroys files, he is in trouble. His files are gone, and it will be monstrously inconvenient and expensive to recover them. Therefore the system does not automatically allow access by subsystems to the files in the permanent directory. If the user trusts all the subsystems he is going to call, there are ways he can grant those subsystems access to files in PERMDIR (see 2.2-2.3), but great caution is advised. It is as though those files were the only copy of the information.

One difference between the various name spaces is indicated by the access key used when looking in the permanent directory. The null key can only be used on one's own directories (PERMDIR, and TEMPDIR in most cases of interest). It gives unrestricted access to any file in those directories. OWN.KEY is the user's personal key which was created along with his permanent directory. It is unique to him, unless he gives it away. The user may grant access to a given file in his permanent directory from name spaces less powerful than the command processor name space by attaching an access lock matching OWN.KEY to the file. The access may be restricted (to read only access, for example) by turning off suitable 'option bits' in the lock one puts on the file (see examples). PUB.KEY gives read only access to the files in the PUBLIC directory.

Now it may be clear that there must be at least two name spaces. On the one hand, unrestricted access tc the files must be possible, otherwise the user might not be able to do something with his file that he wants to do. On the other hand, there must be name spaces which keep unreliable subsystems frcm wreaking havoc. The existence of more than twc name spaces is an unfortunate historical accident.

The existence and use of the name spaces is complicated by compatibility features for subsystems following the conventions of an extinct early version of the system. For both 'old' and 'new' subsystems, the command name is looked up in the command processor name space, but the processing of the subsequent parameters varies.

Old subsystems have all parameters looked up in the BEAD name space. During execution, they may request further objects from the Command Processor, which are also looked up in the BEAD name space. All existing subsystems are being converted to the new conventions as quickly as possible.

New subsystems have their parameters looked up in the command processor name space. During execution, they may request further objects in two ways. If the subsystem makes up the name of the object, it is looked up in the SCANL name space. Objects may be obtained from the command processor name space only if the user types in the name from the TTY. Thus, in either case, permanent files are protected from unruly subsystems and from accidental use as scratch files.

Table 2 - Name Spaces

CCMMAND PROCESSOR NAME SPACE		SCANL[1] NAME SPACE		BEAD NAME SPACE [2]	
DIRECTORY	ACCESS KEY	DIRECTORY	ACCESS KEY	DIRECTORY	ACCESS KEY
SOME SPECIAL NAMES E.G., 'LOGOUT' and 'SERVICES'	NOT APPLICABLE	--	--	--	--
TEMPDIR	NULL	TEMPDIR	NULL	TEMPDIR	NULL
PERMDIR	NULL	PERMDIR	OWN.KEY	PERMDIR	OWNKEY
PUBLIC	PUB.KEY	PUBLIC	PUB.KEY	--	--

1.7 SERVICES, BEAD GHOST, errors

For use of CAL TSS beyond the trivial, a knowledge of these two special subsystems is required. SERVICES and the BEAD GHOST are similar to normal subsystems, but are actually just new 'hats' donned by the Command Processor appropriate to the occasion.

SERVICES is a general utility subsystem allowing manual manipulation of files, directories, etc. The main reason for removing this function from the Command Processcr proper is to minimize the number of reserved words which may nct be used as names cf user subsystem ('SERVICES', 'LOGOUT', etc.).

Unlike SERVICES, which is troublescme because it must be called, the BEAD GHOST is annoying because it appears without being called. The BEAD GHOST is the system debugger and its appearance is prompted by some error. Whenever a subsystem makes a mistake in dealing with some object or some part of the system, errcr processing is initiated. Some errors are handled automatically by various subsystems along the way,

[1] methods for altering SCANL from the console are available.

[2] The BEAD NAME SPACE really occurs in several forms. This is the most common form. Other forms are not of crucial interest and are not described here.

i

and the user usn't even aware of them. Many are reported to the console by a given subsystem to indicate that they were asked to do something illegal or impossible (the Command Processor is an outstanding example of this). Some represent unforeseen circumstances for which no remedial procedures have been provided (called 'bugs' for short). They are reported to the console by the BEAD GHOST in hopes that the user will know what to do (like complain to a system programmer). Currently, only class 6 errors ("6,n,m ERROR") should be reported to the console by the BEAD GHOST under normal circumstances. Other appearances of the BEAD GHOST should be reported, along with all the relevant console printout, to the system staff.

Class 6 errors mean that the resources reserved for the user have become inadequate for the task being performed. When they occur, the user must either obtain additional resources or abort what he was doing, which introduces the next topic.

1.8 Space Control

CAL TSS has several types of storage for which there is currently no automatic algorithm for sharing 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).

Table 3

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

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 fcr more space than the moderate limit, he is put in a special queue which prevents more than one user at a time from being "very large" in any particular type of space.

There is currently no mechanism to force a user to release space once he has it. Several mechanisms tend tc prevent space hogging. 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 :  space of this type is released to get down to nominal
             if possible

if = 0 or not typed (trailing parameters):  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.[3]
          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.[3]
```

There are two different starting points from which the user may find himself requesting space:

1) He is about to call a subsystem and knows in advance how much space it will require: enter SERVICES and request the required amount of space and then go back to the Command Processor and call the subsystem. The request has to be big enough - see below!

2) A subsystem he has called runs out of space and makes a class 6 error which invokes the BEAD GHOST: if he has not already requested space, the user may do so now with the space command. After he has gotten the space, he types RETRY (not RETURN) and the subsystem will resume. If he already has space, there is no way for him to save himself - he must type

[3] A message will print if the space is not immediately available - a panic (see 1.9) will remove the user from the queue if he would rather not wait.

PURGE, which aborts whatever work the subsystem may have done for him, and start over in the Command Processor.

1.9 'WHO' and PANICs (how to untangle a console and how the user stops something he wishes he hadn't started)

WHO is a request that may be typed at the console to determine which subsystem is in control. PANICs are a way of interrupting whatever is going on if the user has somehow lost control. PANICs come in two flavors:

MINOR PANIC (or PANIC for short) - hold down the CTRL and SHIFT keys and simultaneously type P to send a minor PANIC;

MAJOR PANIC - hold down the BREAK key for at least three seconds to send a MAJOR PANIC

The difference between a PANIC and a MAJOR PANIC is that subsystems may handle PANICs on their own if they wish to, but a MAJOR PANIC always invokes some arm of the Command Processor.

The remainder of this section gives three procedures covering different cases of console problems, plus a table telling how to recognize and/or dismiss subsystems.

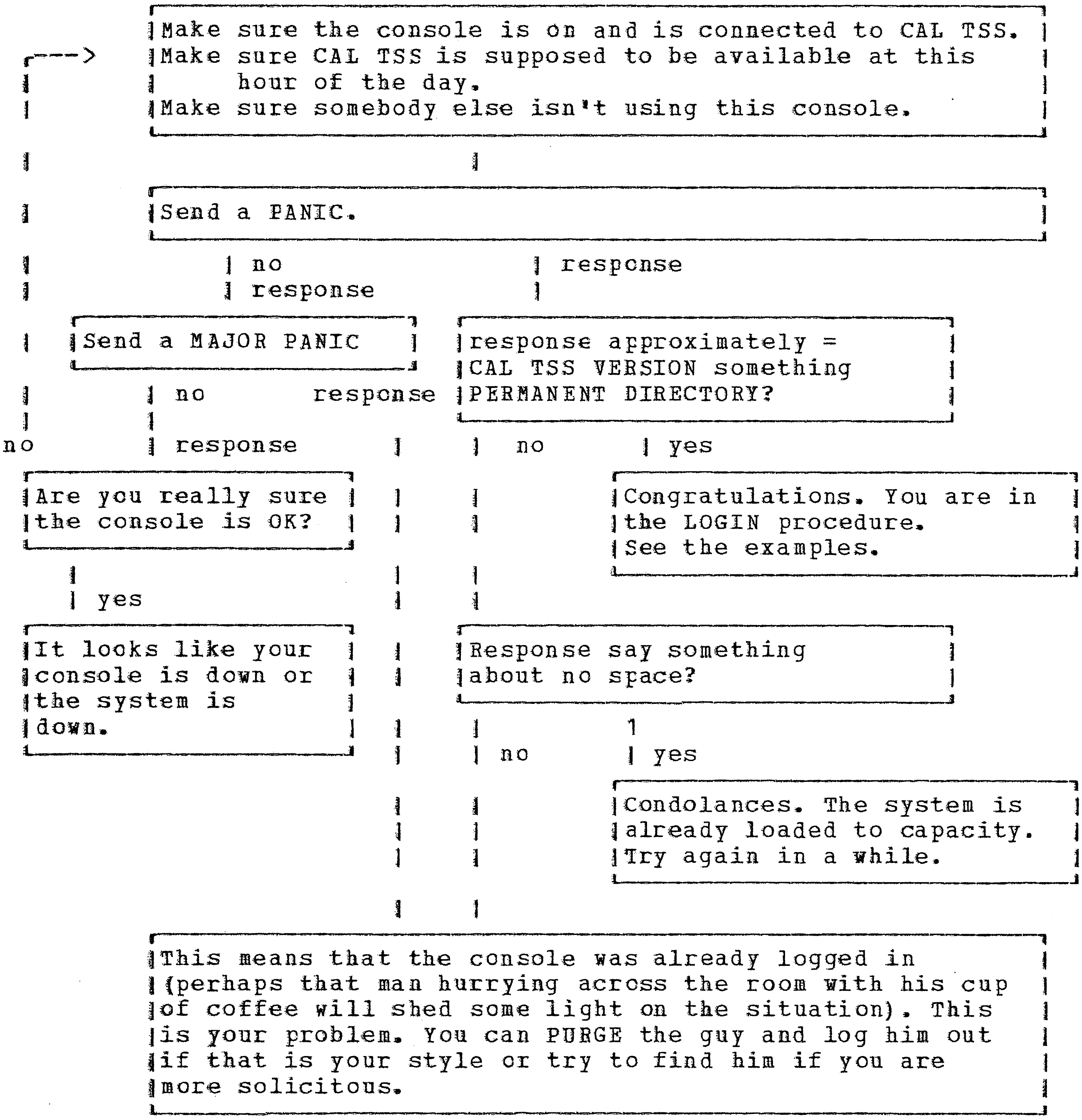
PROCEDURE I covers how to approach a console initially.

PROCEDURE II tells what the user does if he is already logged in and using the console but has either forgotten what he was doing or the console stopped responding the way he expects it to.

PROCEDURE III is for those times when the user has started something that he wants to stop (e.g., the EDITOR is printing 2000 lines because he mistyped something or his BASIC program has been computing silently for an ominous length of time, etc.).

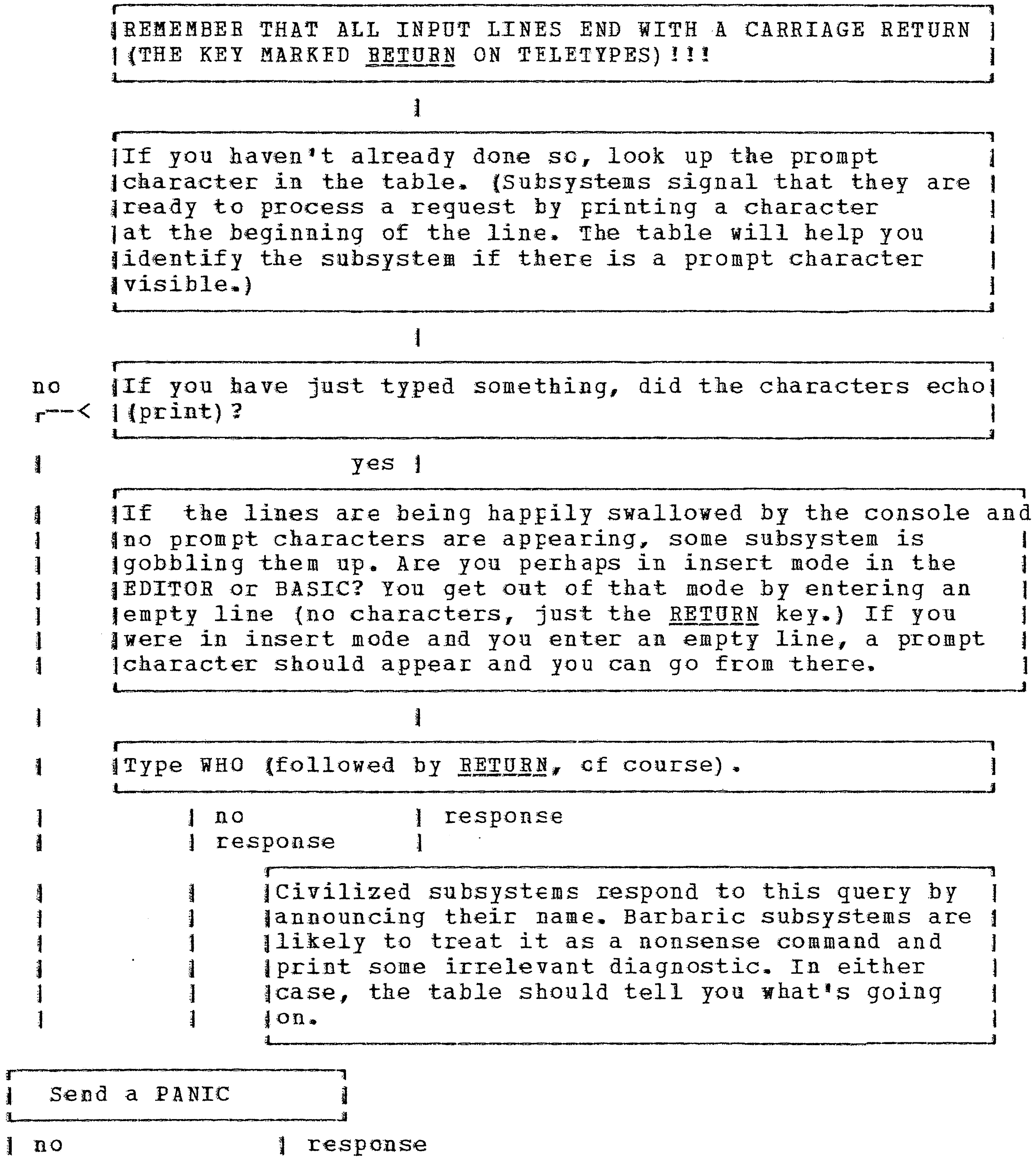
Sometimes the relevant procedure has a happy ending and the user can continue. But, alas, the procedure may suggest that the console is down, or the system is down, or there is a bug in the system. The user can frequently distinguish between a sick console and a sick system by seeing if other consoles in the area are operating. If they are, it looks like the console is sick. If they aren't, it looks like the system is. The current procedures for reporting troubles of this nature should be available from some other sources. They are not included here because they are in a state of flux.

PROCEDURE I - a user is just approaching a console to try to establish contact with CAL TSS



```
           ┌──────────────────────────────────────────────────────────┐
  ┌--->    |Make sure the console is on and is connected to CAL TSS.  |
  |        |Make sure CAL TSS is supposed to be available at this     |
  |        |     hour of the day.                                     |
  |        |Make sure somebody else isn't using this console.         |
  |        └──────────────────────────────────────────────────────────┘
  |                              |
  |        ┌──────────────────────────────────────────────────────────┐
  |        |Send a PANIC.                                             |
  |        └──────────────────────────────────────────────────────────┘
  |                 | no                 | response
  |                 | response           |
  |    ┌──────────────────────┐   ┌───────────────────────────────┐
  |    |Send a MAJOR PANIC    |   |response approximately =       |
  |    └──────────────────────┘   |CAL TSS VERSION something      |
  |        | no      response    |PERMANENT DIRECTORY?           |
  |        |                      └───────────────────────────────┘
 no        | response       |     | no      | yes
  ┌─────────────────┐       |     |         ┌───────────────────────────┐
  |Are you really sure|     |     |         |Congratulations. You are in|
  |the console is OK?|      |     |         |the LOGIN procedure.       |
  └─────────────────┘       |     |         |See the examples.          |
     |                      |     |         └───────────────────────────┘
     | yes                  |     |
  ┌─────────────────┐       |   ┌───────────────────────────────┐
  |It looks like your|      |   |Response say something         |
  |console is down or|      |   |about no space?                |
  |the system is     |      |   └───────────────────────────────┘
  |down.             |      |     |         1
  └─────────────────┘       |     | no      | yes
                            |     |         ┌───────────────────────────┐
                            |     |         |Condolances. The system is |
                            |     |         |already loaded to capacity.|
                            |     |         |Try again in a while.      |
                            |     |         └───────────────────────────┘
                            |     |
           ┌──────────────────────────────────────────────────────────┐
           |This means that the console was already logged in         |
           |(perhaps that man hurrying across the room with his cup   |
           |of coffee will shed some light on the situation). This    |
           |is your problem. You can PURGE the guy and log him out    |
           |if that is your style or try to find him if you are       |
           |more solicitous.                                          |
           └──────────────────────────────────────────────────────────┘
```

PROCEDURE II - the user is logged in and using the console and has either forgotten what he was doing or gotten into some mysterious state where the console doesn't respond the way he expects it to:



REMEMBER THAT ALL INPUT LINES END WITH A CARRIAGE RETURN (THE KEY MARKED RETURN ON TELETYPES)!!!

If you haven't already done so, look up the prompt character in the table. (Subsystems signal that they are ready to process a request by printing a character at the beginning of the line. The table will help you identify the subsystem if there is a prompt character visible.)

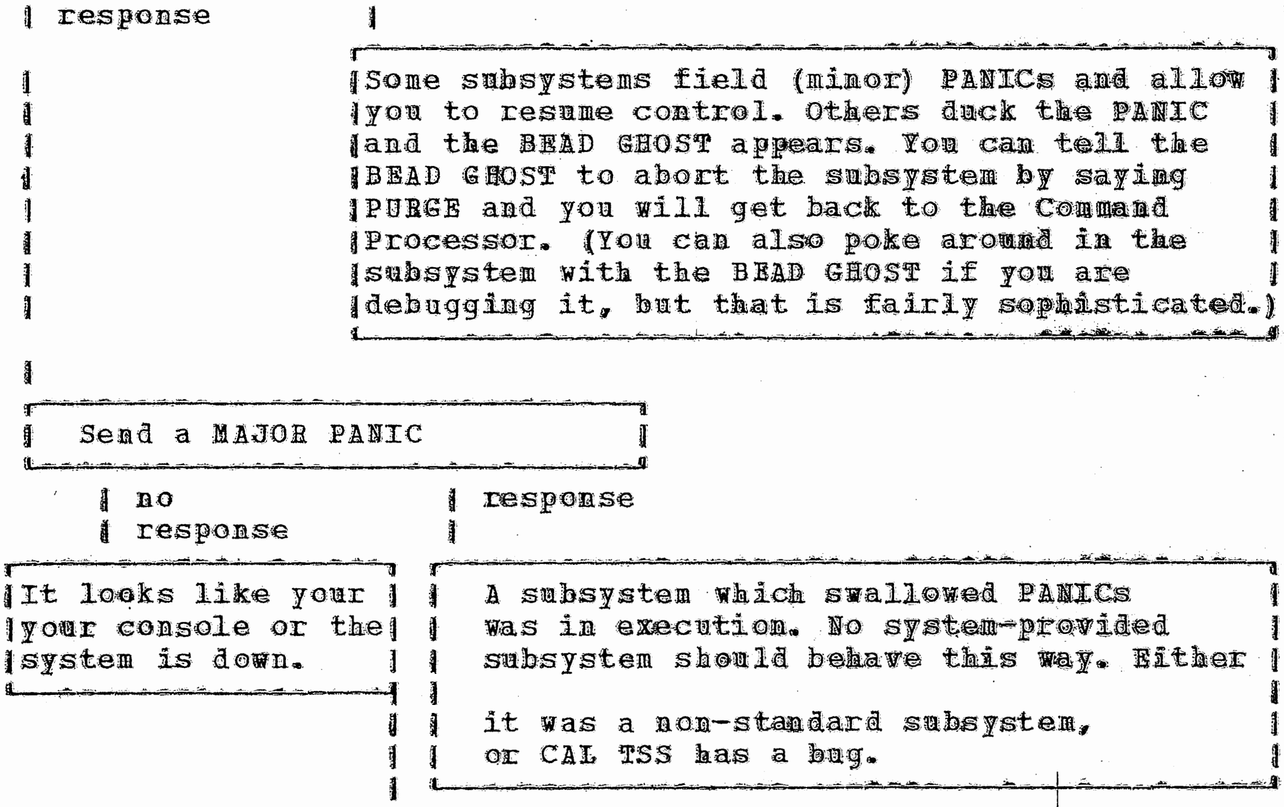
If you have just typed something, did the characters echo (print)?

If the lines are being happily swallowed by the console and no prompt characters are appearing, some subsystem is gobbling them up. Are you perhaps in insert mode in the EDITOR or BASIC? You get out of that mode by entering an empty line (no characters, just the RETURN key.) If you were in insert mode and you enter an empty line, a prompt character should appear and you can go from there.

Type WHO (followed by RETURN, of course).

Civilized subsystems respond to this query by announcing their name. Barbaric subsystems are likely to treat it as a nonsense command and print some irrelevant diagnostic. In either case, the table should tell you what's going on.

Send a PANIC



response

Some subsystems field (minor) PANICs and allow you to resume control. Others duck the PANIC and the BEAD GHOST appears. You can tell the BEAD GHOST to abort the subsystem by saying PURGE and you will get back to the Command Processor. (You can also poke around in the subsystem with the BEAD GHOST if you are debugging it, but that is fairly sophisticated.)

Send a MAJOR PANIC

no response

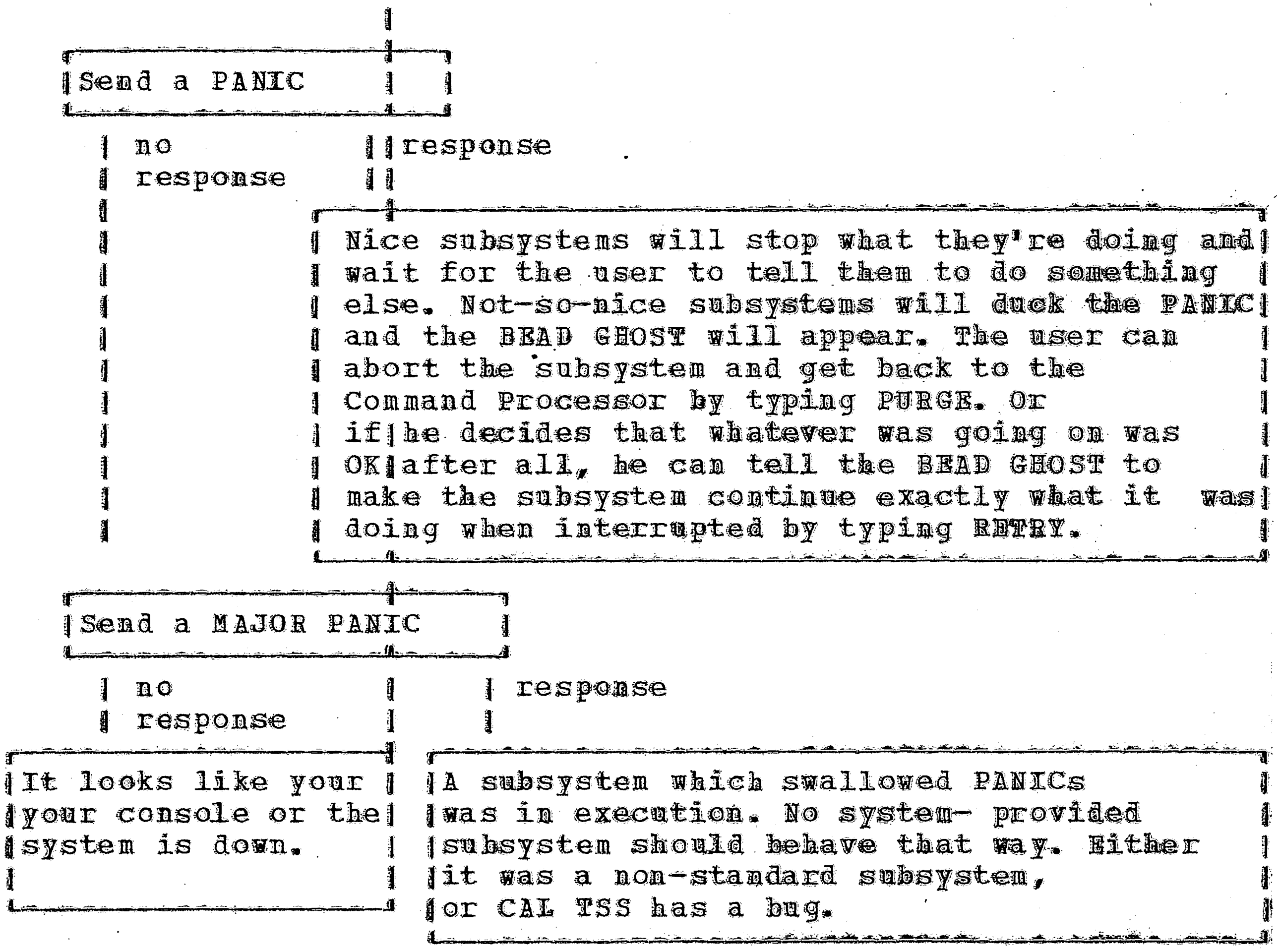
It looks like your your console or the system is down.

response

A subsystem which swallowed PANICs was in execution. No system-provided subsystem should behave this way. Either

it was a non-standard subsystem, or CAL TSS has a bug.

PROCEDURE III — the user has just started something he wishes he hadn't



Send a PANIC

no response

response

Nice subsystems will stop what they're doing and wait for the user to tell them to do something else. Not-so-nice subsystems will duck the PANIC and the BEAD GHOST will appear. The user can abort the subsystem and get back to the Command Processor by typing PURGE. Or if he decides that whatever was going on was OK after all, he can tell the BEAD GHOST to make the subsystem continue exactly what it was doing when interrupted by typing RETRY.

Send a MAJOR PANIC

no response

response

It looks like your your console or the system is down.

A subsystem which swallowed PANICs was in execution. No system- provided subsystem should behave that way. Either it was a non-standard subsystem, or CAL TSS has a bug.

TABLE 3 - HOW TO RECOGNIZE AND/OR DISMISS STANDARD SUBSYSTEMS

SUBSYSTEM	PROMPT	RESPONSES TO INCOMPREHENSIBLE OR ERRONEOUS INPUT	HOW TO DISMISS IT
COMMAND PROCESSOR	!	BAD SYNTAX or SAY AGAIN or UNEXPECTED F-RETURN or UNEXPECTED ERROR or ERROR OCCURRED ON CALL TO CMMDS	This is the ground state of a console. From here, the user may call subsystems or 'LOGOUT' when he is finished.
LOGIN PROCESSOR	.	same as COMMAND PROCESSOR	The user has to successfully finish the login (see examples)
SERVICES	*	same as COMMAND PROCESSOR	'FIN'
BEAD GHOST (debugger)	@	same as COMMAND PROCESSOR	'PURGE' will return to the COMMAND PROCESSOR; 'RETRY' or 'RETURN' will return to the currently active subsystem.
EDITOR	:	????	'F' or 'Q' (see EDITOR document)
BASIC	: or ?	???? or miscellaneous diagnostics relevant tc erroneous BASIC statements	same as EDITOR
SCOPE	(see SCOPE)	??NO??	'FIN'

1.10 The Line Collector

Unless the user does something extraordinary, all console input goes through a piece of software called the Line Collector, which 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 type 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 underliked.

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 RETURN, for example), it becomes the old line.

Suppose the user is talking to BASIC and has 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-I	make an insert at the beginning of the old line	<
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

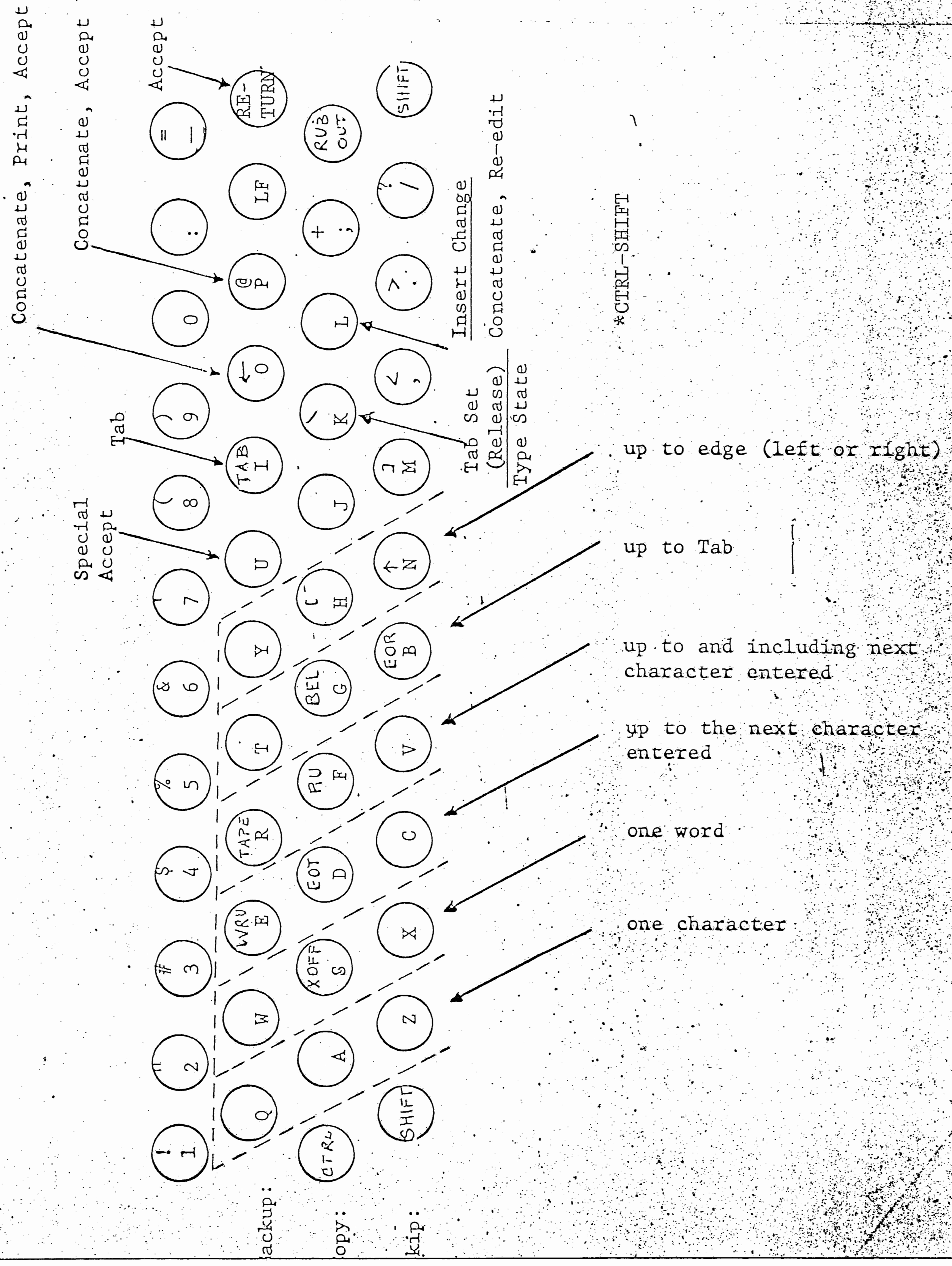
type	meaning	and the teletype responds
CTRL-D	copy the old line into the new line up to the first occurrence of the next character typed	no response
N		10 PR
IM	you wanted IN and made a mistake	IM
CTRL-Q	erase the M	<-
N		N
CTRL-H	copy the rest of the old line into the new line	T_X

,Y	you remembered to print Y	,Y
RETURN	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
```

Figure 1. (33/35) Teletype Keyboard and Control Characters



2. Examples.

These examples are not all-inclusive. They are provided to give a feeling of how CAL TSS works, plus a few pointers on how to do some commonly useful things. The first example is heavily commented, subsequent ones are commented only where they contain points of special interest. Characters typed by the system have been underlined in the first example to distinguish them from the things that the user typed. Subsequent examples are not underlined.

Example 2.1

```
     CAL TSS VERSION 2.Ø
     2Ø:35:14 1Ø/21/71
1.0  PERMANENT DIRECTORY?
     .GUEST                                  1.1
     GIVE PASSWORD
     .GUEST                                  1.2
     TEMPORARY DIRECTORY?
     .JOHN                                   1.3
2.0  COMMAND PROCESSOR HERE
     !BASIC                                  2.1
     BASIC VERSION 2.Ø                       3.1
     -PRINT PI                               3.2
      3.141593
     -1Ø LET X = 13
     -2Ø LET Y = 19←8                        3.4
3.3  -3Ø PRINT X,YX*Y                        3.5
      ERROR  OPERATOR MISSING
     -3Ø PRINT X,Y,X*Y
     -4Ø END
     -RUN                                    3.6
3.0   13             18           234
      EXECUTION COMPLETE
     -LIST 3Ø                                3.7
     3Ø PRINT X,Y,X*Y
     -EDIT 3Ø                                3.8
     3Ø PRINT X,Y,X*Y,X/Y                           3.9
     -RUN                                    3.10
      13             18           234               .7222222
      EXECUTION COMPLETE
     -FIN                                    3.11
     CHANGES NOT SAVED
     -FIN
     COMMAND PROCESSOR HERE
     !LOGOUT                                 4.1
     2Ø:37:1Ø 1Ø/21/71
     CONNECT TIME = 97782.
     CPU TIME = 63116Ø2.
4.0  FIXED ECS = 34468155Ø.
     MOT SLOTS = Ø.
     SWAPPED ECS = 4Ø7565312.
     TEMP DISK = Ø.
     MONEY =$.297
     GOOD DAY
```

EXAMPLE 2.1 - SIMPLE USE OF BASIC, NO FILES KEPT

1.0 These lines constitute the login procedure. Prior to the first line, the user has attracted the attention of CAL TSS by typing P while holding down the CTRL and SHIFT keys.

1.1 'GUEST' is the name given for the permanent directory.

1.2 The password to use the GUEST directory is also 'GUEST', but the password is not usually the same as the directory name.

1.3 'JOHN' is the name the user chose to give to the temporary directory.

2.0 The appearance of the Command Processor signals the successful completion of the login procedure.

2.1 The user tells the Command Processor that he wants to use the BASIC subsystem.

3.0 All these lines are a conversation with the BASIC subsystem.

3.1 BASIC announces its presence and signals that it is ready to process commands by printing '-'.

3.2 The user gives it an immediate command to print the value of pi and it responds with the value.

3.3 Now the user decides to construct a simple BASIC program, so he begins entering indirect statements. These lines constitute the text of the BASIC program being constructed.

3.4 This is an example of erasing a mistake. The arrow printed because the user typed CTRL-Q to erase the 9. The actual line entered was '20 LET Y = 18'.

3.5 The user forgot a comma in this line, so BASIC does not recognize it as a valid statement and complains. The correct line is entered.

3.6 The user tells BASIC to run the program he just constructed and it runs the program and prints the results.

3.7 He decides to change the program and types the request 'LIST 30', which types line 30 for inspection.

3.8 The user tells BASIC that he is going to edit that line, so it is made the old line in the Line Collector.

3.9 This line was constructed by typing CTRL-H, which copied all of the old line, and then typing ',X/Y' followed by RETURN.

3.10 The user now runs his program again and the new results appear.

3.11 The FIN command tells BASIC that the user is finished. BASIC warns the user that changes have been made in the program which will be lost if the user does not use the SAVE command to save the new program. The user repeats FIN to inform BASIC that he does not wish to save the program he has constructed.

4.0 The Command Processor resumes control of the console.

4.1 The user signals that he is finished using the system by typing 'LOGOUT'. The system prints the accounting data for the run and after it wishes him a good day, the console goes dead.

This page no longer contains information.

Example 2.2

```
          CAL TSS VERSION 1.2
          PERMANENT DIRECTORY?
          .USER:VV
  1.0     GIVE PASS WORD
          .ØRBL
          TEMPORARY DIRECTORY?
          .V
  2.0     COMMAND PROCESSOR HERE
          !SERVICES
          SERVICES HERE
          *NEWDF PERMDIR:AUTO        3.1
          *PCAP OWN.KEY
          77777777777777ØØ2737          3.2
          ØØØØØØØØØØØØØØØ53ØØ2
  3.0     *ADDKEY 53Ø02 77777777777777 PERMDIR:AUTO   3.3
          *NEWDI←F PERMDIR:MANUAL               3.4
          *MCAP PERMDIR:MANUAL TEMPDIR:M           3.5
          *FIN                                3.6
          COMMAND PROCESSOR HERE
          !EDITOR AUTO
          :I
          1Ø PRINT 1Ø*PI
          2Ø PRINT 2ØPI
          3Ø END

  4.0     :F
          COMMAND PROCESSOR HERE
          !EDITOR M
          :I
          1Ø LET X = 1Ø
          2Ø LET Y = 2Ø
          3Ø PRINT X*PI,Y*PI
          4Ø END

          :F
          COMMAND PROCESSOR HERE
          !LOGOUT
          GOOD DAY
```

EXAMPLE 2.2 - CREATION OF PERMANENT DISK FILES TO BE KEPT FOR FUTURE SESSIONS

1.0 This is the login procedure again, except that the permanent directory name is 'USER:VV' and the password is 'QRBL'. 'V' has been chosen as the name for the temporary directory.

2.0 The user tells the Command Processor to call the subsystem SERVICES.

3.0 These lines are a conversation with SERVICES.

3.1 The user requests SERVICES to make a new disk file by saying NEWDF. He has asked that it be created in his permanent directory and named AUTO.

3.2 The command 'PCAP OWN.KEY' causes the user's private access key to be displayed. This is done so that he can see the number of the access key, which is required by the command which adds locks to names. The number is the 53002 which occurs in the second line.

3.3 This command adds lock 53002 matching his OWN.KEY, to the file AUTO in his PERMDIR. The string of 7's are the kinds of access which the user is allowing, namely all kinds of access. The addition of this lock to the name 'AUTO' makes the file AUTO available in the BEAD name space, and it will automatically be available whenever he logs on in the future.

3.4 A mistake was made in entering this line; the first 'I' was erased by typing CTRL-Q. The line actually entered was 'NEWDF PERMDIR:MANUAL', which creates a new file MANUAL in the user's PERMDIR.

3.5 Because the user decided not to have automatic access to MANUAL, he set up a name in TEMPDIR which can be used to access MANUAL during this console session. The sense of this command is to allow the file MANUAL in PERMDIR to be referred to as M in TEMPDIR.

3.6 This dismisses SERVICES and the Command Processor returns.

4.0 The Editor is used to put some text in the files AUTO and MANUAL, alias M, for future sessions.

October 1971

Example 2.3.1

```
      CAL TSS VERSION 2.Ø
      2Ø:4Ø:39 1Ø/21/71
      PERMANENT DIRECTORY?
      .USER:VV
      GIVE PASSWORD
      .QRBL
      TEMPORARY DIRECTORY?
      .V
      COMMAND PROCESSOR HERE
    { !BASIC
    { BASIC VERSION 2.Ø
    { -LOAD AUTO  ← - - - - - - - - - - - → 1.1
    {  ERROR  OPERATOR MISSING}
    { 2Ø PRINT 2ØPI           }← - - - → 1.2
    { -LIST  ← - - - - - - - - - - - - - → 1.3
1.0 { 1Ø PRINT 1Ø*PI}
    { 3Ø END        }← - - - - - - 1.4
    { -2Ø PRINT 2Ø*PI ← - - - - - - - → 1.5
    { -RUN  ← - - - - - - - - - → 1.6
    {  31.41593
    {  62.83185
    {  EXECUTION COMPLETE
    { -SAVE AUTO  ← - - - - - - → 1.7
    { -FIN
      COMMAND PROCESSOR HERE
      !LOGOUT
      2Ø:41:43 1Ø/21/71
      CONNECT TIME = 47156.
      CPU TIME = 7245765.
      FIXED ECS = 1662249ØØ.
      MOT SLOTS = Ø.
      SWAPPED ECS = 224351232.
      TEMP DISK = Ø.
      MONEY =$.296
      GOOD DAY
```

EXAMPLE 2.3.1 - USE OF A PREVIOUSLY CONSTRUCTED FILE IN BASIC

1.0 Only the interaction with BASIC is described, although the reader should note that no special manipulations were done after login to get access to AUTO.

1.1 The command 'LOAD AUTO' tells BASIC to load the file AUTO.

1.2 The user may not have noticed the mistake made when constructing AUTO, but BASIC does notice. It prints a diagnostic message followed by the offending statement.

1.3 After BASIC has read the whole file, it prompts again. The user tells it to list the program.

1.4 The program is printed and he sees that the statement in error has been left out.

1.5 This is the correct form of the statement.

1.6 He asks that the program be run and the results are printed out.

1.7 Because the user made a correction to his program, he wants to save the new version, so he does a 'SAVE'. The FIN leaves BASIC destroying the program in it.

Example 2.3.2.1

```
      CAL TSS VERSION 2.Ø
      2Ø:42:27 1Ø/21/71
      PERMANENT DIRECTORY?
      .USER:VV
      GIVE PASSWORD
      .QRBL
      TEMPORARY DIRECTORY?
      .V
      COMMAND PROCESSOR HERE
1.0   !EDITOR MANUAL
      :T;P$

      :Q
      COMMAND PROCESSOR HERE
2.0   !SERVICES
      SERVICES HERE
      *MCAP PERMDIR:MANUAL TEMPDIR:M          2.1
      *FIN
      COMMAND PROCESSOR HERE
      !BASIC
      BASIC VERSION 2.Ø
3.0   -LOAD M
      -RUN
       31.41593       62.83185
       EXECUTION COMPLETE
      -FIN
      COMMAND PROCESSOR HERE
      !LOGOUT
      2Ø:43:58 1Ø/21/71
      CONNECT TIME = 71783.
      CPU TIME = 1Ø849976.
      FIXED ECS = 253Ø386ØØ.
      MOT SLOTS = Ø.
      SWAPPED ECS = 31967488Ø.
      TEMP DISK = Ø.
      MONEY =$.443
      GOOD DAY
```

EXAMPLE 2.3.2.1 - SELECTIVE MANUAL ACCESS TO PERMANENT FILE

1.0 This shows that the Editor wasn't given a copy of the user's file MANUAL, because he printed the file and it is empty.

2.0 The user talks to SERVICES to set up access to MANUAL.

2.1 This command sets up access to MANUAL in his PERMDIR under the name 'M' in TEMPDIR.

3.0 He calls BASIC, reads in his file MANUAL, alias M, and executes the program.

Example 2.3.2.2

```
         CAL TSS VERSION 2.Ø
         2Ø:46:Ø4 1Ø/21/71
         PERMANENT DIRECTORY?
         .USER:VV
         GIVE PASSWORD
         .QRBL
         TEMPORARY DIRECTORY?
         .V
         COMMAND PROCESSOR HERE
        {!SERVICES
        {SERVICES HERE
        {*CHAIN PERMDIR TEMPDIR    1.1
  1.0   {*UNCHAIN PERMDIR              1.2
        {*CHAIN TEMPDIR PERMDIR            1.3
        {*FIN
        {COMMAND PROCESSOR HERE
        {!BASIC
        {BASIC VERSION 2.Ø
        {-LOAD MANUAL
  2.0   {-RUN
        { 31.41593       62.83185
        { EXECUTION COMPLETE
        {-FIN
         COMMAND PROCESSOR HERE
         !LOGOUT
         2Ø:47:14 1Ø/21/71
         CONNECT TIME = 52511.
         CPU TIME = 7699295.
         FIXED ECS = 1851Ø48ØØ.
         MOT SLOTS = Ø.
         SWAPPED ECS = 247353344.
         TEMP DISK = Ø.
         MONEY =$.317
         GOOD DAY
```

EXAMPLE 2.3.2.2 - ACCESS FOR SUBSYSTEMS TO ALL YOUR PERMANENT FILES

1.0 This conversation with SERVICES makes the the user's PERMDIR look like part of his TEMPDIR and hence gives access to his permanent files to all subsystems which have access to the temporary files.

1.1 CHAIN causes the first directory, PERMDIR, to have the second directory, TEMPDIR, appended to it. Oops, that's backwards.

1.2 So UNCHAIN takes any appended directory out of PERMDIR.

1.3 Now CHAIN appends PERMDIR to TEMPDIR, which is what the user was trying to do. If he hadn't unchained PERMDIR from TEMPDIR back at step 1.2, the two directories would constitute a loop and the code which looks up names would get annoyed if it ever used them.

2.0 The same use of BASIC as in the previous example.

Example 2.4

```
CAL TSS VERSION 1.2
NO ROOM, SWPEGS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
NO ROOM, SWPECS
GOOD DAY
CAL TSS VERSION 1.2
PERMANENT DIRECTORY?
.USER:VV
GIVE PASS WORD
.QRBL
TEMPORARY DIRECTORY?
.V
        COMMAND PROCESSOR HERE
        !SERVICES
  1.0   SERVICES HERE
        *MCAP PERMDIR:TRIVIA TEMPDIR:INPUT
        *FIN
```

```
       COMMAND PROCESSOR HERE
       !EDITOR INPUT
       :T;P$                                 (2.1)
                PROGRAM TRIV(TTYIN,TTYOUT,TAPE2=TTYIN,TAPE1=TTYOUT)
                WRITE  (1,1ØØ)
(2.0)  1ØØ      FORMAT     (*TRIVIA SPEAKING, WHO'S THERE?*)
                READ  (2,2ØØ) NAME
       2ØØ      FORMAT     (A1Ø)
                WRITE  (1,3ØØ) NAME
       3ØØ      FORMAT     (*GOODBYE,*A1Ø)
                END
       :Q
       COMMAND PROCESSOR HERE
       !SCOPE 4ØØØØ                                            (3.1)
       15:42:35 Ø8/Ø6/71 SCOP32C OF Ø8/Ø1/71
       >RUN                                                    (3.2)
        WAITING AT TOP OF QUEUE FOR  SWAPPED ECS SPACE
                COMPILING  TRIV
       >LGO                                                    (3.3)
        WAITING AT TOP OF QUEUE FOR  SWAPPED ECS SPACE
(3.0)   WAITING FOR ACCESS TO  SWAPPED ECS SPACE
                          3 AHEAD IN QUEUE
        WAITING AT TOP OF QUEUE FOR  SWAPPED ECS SPACE
       BEGIN EXECUTION     TRIV
       TRIVIA SPEAKING, WHO'S THERE?
       ↑GEORGE                                                 (3.4)
       GOODBYE,GEORGE
       END        TRIV
       >FIN                                                    (3.5)
       COMMAND PROCESSOR HERE
       !LOGOUT
       GOOD DAY
```

EXAMPLE 2.4 - SCOPE SIMULATOR: A SIMPLE INTERACTIVE FORTRAN PROGRAM

This example was generated when the system was fairly busy. When the user tried to log on, he was refused access because there was no space to accomodate him. The space fluctuates on a short time scale, so the user just kept trying until he got on. Subsequently, the SCOPE subsystem requested additional space which was not immediately available and CAL TSS printed the messages saying 'waiting at top of queue...' and 'waiting for access to...' so that the user would be forewarned that processing his request might take longer than usual.

1.1 The reader has seen this before. The file TRIVIA in PERMDIR is made available in TEMPDIR as INPUT.

2.0 The file is printed with the Editor.

2.1 Notice the special file names used to talk to the console.

3.0 The user asks for the SCOPE Simulator. Characters typed by the user are underlined in this section.

3.1 SCOPE requests the SCOPE Simulator and the 40000 is an optional parameter which determines the initial FL in the Simulator. If it is omitted, a default value of 14000 is used. 40000 is required to use the RUN complier so that is why this value was chosen. SCOPE prints the time and date.

3.2 > is SCOPE's prompt character, signalling that it is ready to process a request. The user may type the same commands that he would have put on his control cards when using the batch system. In particular, RUN causes the FORTRAN compiler to compile statements from the file INPUT.

3.3 Another command causes the compiled program to be loaded and executed.

3.4 The previous line was printed by the user's program. The | is the prompt character which signals that a program running on the simulator is waiting for input, as opposed to the simulator itself. After the user responds 'GEORGE', (followed by RETURN, of course), the program grinds to its rather uninspiring conclusion and SCOPE starts watching the console again.

3.5 SCOPE prompts for another command and the user dismisses it. The Command Processor reappears.

Example 2.5

```
      CAL TSS VERSION 1.2
      PERMANENT DIRECTORY?
      .GUEST
      GIVE PASS WORD
      .GUEST
      TEMPORARY DIRECTORY?
      .VANCE
      COMMAND PROCESSOR HERE
1.0   !SERVICES
      SERVICES HERE
      *PCAP OWN.KEY
      77777777777777ØØ2737      1.1
      ØØØØØØØØØØØØØØ123401
      *FIN
      COMMAND PROCESSOR HERE
      !LOGOUT
      GOOD DAY
```

```
      CAL TSS VERSION 1.2
      PERMANENT DIRECTORY?
      .USER:VV
      GIVE PASS WORD
      .ORBL
      TEMPORARY DIRECTORY?
      .VANCE
      COMMAND PROCESSOR HERE
2.0   !SERVICES
      SERVICES HERE
      *ADDKEY 1234Ø1 7142Ø PERMDIR:REACT
      *ADDKEY 1234Ø1 7142Ø PERMDIR:DATA       2.1
      *FIN
      COMMAND PROCESSOR HERE
      !LOGOUT
      GOOD DAY
```

3.0

```
       CAL TSS VERSION 1.2
       PERMANENT DIRECTORY?
       .GUEST
       GIVE PASS WORD
       .GUEST
       TEMPORARY DIRECTORY?
       .VANCE
       COMMAND PROCESSOR HERE
       !SERVICES
       SERVICES HERE
       *MCAP VV:REACT;OWN.KEY PERMDIR:REACT
       UNEXPECTED FRETURN
       *MCAP USER:VV:REACT;OWN.KEY PERMDIR:REACT
3.1    UNEXPECTED FRETURN
       *FRIENTP USER:VV
       BAD SYNTAX
       *FRIENTP USER:VV TEMPDIR:VV
       BAD SYNTAX
3.2    *FRIENDP USER:VV TEMPDIR:VV
       *MCAP VV:REACT;OWN.KEY PERMDIR:REACT
3.3    *MCAP VV:DATA;OWN.KEY PERMDIR:DATA
       *ADDKEY 1234Ø1 777777777777777 PERMDIR:REACT
3.4    *ADDKEY 1234Ø1 777777777777777 PERMDIR:DATA
       *FIN
       COMMAND PROCESSOR HERE
       !LOGOUT
       GOOD DAY
       CAL TSS VERSION 1.2
       PERMANENT DIRECTORY?
       .GUEST
       GIVE PASS WORD
       .GUEST
4.0    TEMPORARY DIRECTORY?
       .VANCE
       COMMAND PROCESSOR HERE
       !SCOPE
       16:19:54 Ø8/Ø6/71 SCOP32C OF Ø8/Ø1/71
4.1    >SNOBOL,I=REACT
        SUCCESSFUL COMPILATION

       WOULD ANYONE OUT THERE LIKE TO HEAR SOME POEMS?

4.2    ↑SURE

       HELLO. WHAT IS YOUR NAME?

       ↑VANCE

       I WRITE POETRY. WOULD YOU CARE FOR A POEM, VANCE?

       ↑YES

       GOOD. I SPECIALIZE IN WRITING HAIKU. SHALL I EXPLAIN
       ABOUT THE FORM IN WHICH HAIKU ARE WRITTEN?

       ↑NO THANX

       VANCE, I ALWAYS FIND ONE'S PHONE NUMBER A KEY TO
       PERSONALITY.  WHAT IS YOUR PHONE NUMBER?

       ↑6425823

       NAME A SEASON--OR IF YOU PREFER I'LL CHOOSE ONE

       ↑SUMMER

       THANK YOU. SUCH A LOVELY SEASON. IT INSPIRES ME.
```

```
        FISHERMAN'S BOAT DRIFIS
        GLIMPSE OF YELLOW PINE POLLEN
        FIREFLIES WANDERING.


        WOULD YOU CARE FOR ANOTHER POEM?

        ↑NO


        I UNDERSTAND, VANCE. THE SOUL CAN TAKE ONLY
        SO MUCH POETRY AT ONE TIME.


        WOULD ANYONE OUT THERE LIKE TO HEAR SOME POEMS?

        ↑NO


        THAT'S ALL RIGHT.  I'M WRITING A SONNET CYCLE
        >FIN
  4.3   COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY
```

EXAMPLE 2.5 - SCOPE SIMULATOR: AN INTERACTIVE SNOBOL PROGRAM USING A FILE FROM A FRIEND'S DIRECTORY directory

This rather complicated example involves four separate console sessions.

1.0 The whole purpose of this session is to find out the number of the user's access key so that his friend can add it to the files she wants to let the user use.
1.1 The user tells SERVICES to print OWN.KEY so that he can see its number, which is 123401.
2.0 This session is done by the user's friend, in order to add locks matching the user's key to her files.
2.1 These commands to SERVICES add locks matching his key, which is 123401, to his friend's files REACT and DATA in her permanent directory. Only read access is allowed by the option lists 71420.
3.0 Now the user is going to make links in his own permanent directory to his friend's files.
3.1 This is an example of typing first and thinking later. None of these commands did anything except provoke nasty messages from SERVICES.
3.2 Finally, FRIENDP causes a search to be made for a permanent directory named 'USER:VV', and if one is found, a link to it named 'VV' will be placed in TEMPDIR. If a permanent directory USER:VV isn't found, the user will get some message like the ones printed above.
3.3 These commands make links in PERMDIR named 'REACT' and 'DATA' to files REACT and DATA in the directory VV. The meaning of 'VV:REACT;OWN.KEY' scans roughly as: find something named 'VV', (which will be the permanent directory of the user's friend USER:VV) and look up file REACT in that directory using the access key OWN.KEY.
3.4 These commands have been seen before. They give automatic access in the future to the files named by 'REACT' and 'DATA' in the user's permanent directory. Even though the locks added here would allow all kinds of access, read only access is all that is allowed because of the locks on REACT and DATA in USER:VV.
4.0 This session uses the files to which the user has laboriously gained access. It is program written in SNOBOL which interacts with the console and writes poetry.
4.1 The user calls SCOPE and invokes SNOBOL on his file REACT.
4.2 Most of the rest of this example is a conversation with the poet. Lines which start with the ' indicate that the poet is waiting for the user to say something and the characters after the ‖ are whatever the user chooses to respond.
4.3 When interest in poetry wanes, the poet goes away and SCOPE resumes watching the console. The user leaves much edified.

July 1971

Example 2.6

```
        CAL TSS VERSION 1.2
        NO ROOM, SWPECS
        GOOD DAY
        CAL TSS VERSION 1.2
        NO ROOM, SWPECS
        GOOD DAY
        CAL TSS VERSION 1.2
        NO ROOM, SWPECS
        GOOD DAY
 1.0    CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .QRBL
        TEMPORARY DIRECTORY?
        .V
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY


         CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .VV
        UNEXPECTED FRETURN           2.1
        PERMANENT DIRECTORY?
        .USER:VV;
        BAD SYNTAX                   2.2
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .PASS
 2.0    PASS WORD NOT CONFIRMED          2.3
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .QRBL
        TEMPORARY DIRECTORY?
        .PAUL
        DUPLICATE TEMPDIR                2.4
        TEMPORARY DIRECTORY?
        .VANCE
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY
```

EXAMPLE 2.6 - LOGIN PROBLEMS ILLUSTRATED

1.0 When the user sent his CTRL-SHIFT-P to CAL TSS, there wasn't enough space to accomodate him. The space in the system fluctuates on a fairly short time scale, so trying again every few seconds will generally get the user on before he can get annoyed.

2.0 This interaction illustrates the consequences of most of the mishaps that can occur during login.

2.1 'UNEXPECTED FRETURN' means that there is not a permanent directory named 'VV'.

2.2 'BAD SYNTAX' indicates that 'USER:VV;' is not even a possible name for a permanent directory.

2.3 Self-explanatory.

2.4 'DUPLICATE TEMPDIR' means that someone else has already named his TEMPDIR 'PAUL'. The user must keep choosing a new name until he gets one that does not conflict.

3.1 Summary of the Editor

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

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

EDIT *fname1* *fname2*

where *fname1* is the name of the file to be edited and *fname2* is the name of the file that the results are written on. *fname1* is the default value of *fname2*. All file names are specified by standard parameters. The Editor prompts by typing : and awaits 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 the 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,

or 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 (carriage return only).
D*sc*	Delete the specified lines.
T	Move to the top of the file (pseudo-line).
M*sc*	Move forward over the specified lines.
B*sc*	Move backward the specified number of lines. (NOTE: *sc* can only be a number.)
P*sc*	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 each of the specified lines.

Esc — Edit the specified lines using the Line Collector.[4]

R,fname[5] — Insert the contents of file fname after the current line.

W,fname[5],,sc — Write the specified lines, including the current line, into the file fname,

F,fname[5] — Finished - create the file fname from the latest version; simply entering 'F' causes the updated text to replace the original file fname2 specified when the Editor was called.

Q — Finished but do not save any file.

The Editor prompts with : and responds ???? to lines it does not understand.

[4] Each line being edited is made the old line in the line collection and may then be altered using the Line Collector. (See section 1.10 on the Line Collector.)

[5] If fname is null a CP name is requested on the next line.

see other 42

3.1 Summary of the Editor

The Editor subsystem enables the TSS user to construct and edit files of coded information. A file consists of lines, where a line is a string of coded characters ending with a carriage 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. All file names are looked up in the BEAD name space. The Editor prompts by typing : and awaits 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 cr, 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 the 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,

or 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 (carriage return only).
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

see other 43

	in the specified lines.
CG/str1/str2/sc	Replace every occurrence of str1 by str2 in each of the specified lines.
Esc	Edit the specified lines using the Line Collector.[*]
R,fname	Insert the contents of the file fname after the current line.
W,fname,,sc	Write the specified lines, including the current line, into the file fname,
F,fname	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 : and response ???? to lines it does not understand.

[*] Each line being edited is made the old line in the line collection and may then be altered using the Line Collector. (See section 1.10 on the Line Collector.)

3.2 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, available at the Computer Center Library.

BASIC accepts two 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).

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. Those without line numbers are assumed to be direct. Statements which may be indirect only are those that would only make sense in a program. Statements which may only be direct are usually for changing the program itself rather than the data it works on.

BASIC is called by typing a command of the form:

```
BASIC fname
```

where fname, if specified, is a file containing a BASIC program to be loaded. BASIC responds with BASIC VERSION ... after which either direct statements or a program of indirect statements may be entered.

BASIC prompts with -.

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

1. Pass BASIC a file fname as the first parameter when it is called; the file is loaded in the same manner as when a 'LOAD' command is given.

2. Use the 'LOAD' command to read in a program from a file. Lines containing errors will be typed out after an error message and are not included in the program.

3. Create a new program by typing it into BASIC. Lines with errors will not be saved.

Sample BASIC program starting from the Command Processor:

```
BASIC
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
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
EXECUTION COMPLETE
```

Now the user may:

1. Edit his program using direct statements and rerun it.
2. Quit (and return to the Command Processor) by typing FIN.
3. Save his program by typing SAVE fname.

List of Indirect or Direct Statements

LET var=[...var=]expr

Each variable[5] 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.
`20 DIM A(60),L(5,N,3*N)`

SIG expr

Number of significant digits printed for numbers is changed to the value of expr.
Example: `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

[5] 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.

in 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 to the statement with line number lnum if the logical expression is true.
Example: 105 IF A>B/SIN(X) GOTO 115

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

ON expr GCTO 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.

```
Example: 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 = CCN - Array c is set to all ones.

MAT c = a+b - Array c is set to the sum of a plus b.

MAT c = a-b - Array c is set to the difference between a and b.

MAT c = a*b - Array c is set to the product of 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.
Example: 12 DATA 5,7.3,30+52

PAUSE[str]
Execution pauses and str, if given, is printed. BASIC will accept direct statements or editing request; 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 var=expr TO expr[STEP expr]
NEXT var
Defines the limits of a loop. The three expressions give the initial values 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.

LIST [line number[-line number]]

Prints out the specified lines on the teletype. If the line numbers are omitted or are replaced by 'ALL', then the entire program is printed.

DELETE line number[-line number]

Deletes the specified lines from the program. If 'ALL' is typed instead of the line numbers, then the whole program is deleted. Note that this statement has no effect on the values that may have been stored into any variables.

EDIT line number [-line number]

The specified lines are passed one at a time to the line collector for editing. Note that if the line number is altered so that it is larger than what it was before but is still smaller than the number of the last line in the range specified, then the line will be edited again when the new line number's turn comes.

~~LOAD [line will be edited again when the new line numbers turn comes.~~

LOAD [fname]

Loads a program from a text file of the given name. (No lines which may have been entered into BASIC are deleted.) The name, if given, is a simple name which is looked up using the scan list SCANL, created by the system in the user's temporary directory. SCANL looks for the file in the user's temporary directory, his permanent directory (with the user's own access key) and then in the public directory. To type a more complicated name, fname is omitted and a prompt character quote (") will appear, after which any Command Processor name can be specified.

SAVE [fname]

Writes all the text onto a file of the given name, which must be in the same format as for load. However, if a name is given and no file by that name exists, then a new file is created in the user's temporary directory with that name.

WHO

Types out BASIC.

QUIT
FIN

Both of these statements return to the Command Processor after destroying any program that may have existed.

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
&	Logical AND
NOT	Logical NOT

Functions

ABS(X)	|X|	LGT(X)	log 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)	x
EXP(X)	e	TAN(X)	tan(x)
INT(X)	integer	TIM(X)	seconds used
LOG(X)	log x		

3.3 Summary of the SCOPE Simulator

SCOPE provides an operating environment fo many programs written for CAL's 6400 batch system (SCOPE 3.0 or CALIDOSCOPE), as well as real-time control over the construction and execution of such programs by a user at a console.

SCOPE is called with the following command:

```
          SCOPE fl
```

where fl is an optional parameter specifying the field length. When omitted, 14000 is the default value. SCOPE responds by typing the date and time and then awaits requests after typing >, which is its prompt character. Programs executing under SCOPE prompt with | when they want input frcm the console.

SCOPE creates several standard files necessary for its operation whenever it is called, notably a SYSTEXT file called 'OUTPUT'. Whenever it needs a file to process a request, it gets it from the BEAD NAME SPACE. If there is no file by the appropriate name available, one is created in TEMPDIR.

SCOPE Simulator Requests

Request	Meaning
TEXT,fname	Declare a new SYSTEXT file fname (will not change to SYSTEXT a file which already exists in another mode).
FILE,fname	Use the file fname as the source of SCOPE Simulator requests.
MSG,OFF or ON	Suppresses program messages to the console or restarts them.
GET,fname	Get the file fname from the BEAD NAME SPACE.
PUT,fname	Return fname tc its directory.
STEP	Trace calls made on the Simulator by code setting cell 1. SCOPE will print each cell 1 call in octal and then await a response: B call the debugger S perform the request E ignore the request and perform END instead G leave step mode and then perform the request
FIN	Exit from SCCPE Simulator.

Loading requests:

	Meaning
L,fname	Load and link the file fname
LGO,fname	Load and link the file fname and start the resulting code executing

LDCTL,TSS	Set TSS mode for the loader (load all common blocks after program blocks)
OVERLAY,fname	Contents of loaded and linked core (without banner words) are written onto file fname

CALIDOSCOPE Control Requests:

- CATALOGUE
- COMPARE
- COMPASS
- COPY
- COPYL
- COPYN
- COPYBSF
- CPC
- DMP
- REWIND
- RFL
- RUN
- SNOBOL
- UPDATE

Library Programs:

- CFIO
- DEBUG
- IO
- IORANDOM
- KOMMON
- MEMORY
- REGDUMP
- SETPRU
- TRACE

All RUN FORTRAN Library Routines

3.4 SERVICES and the BEAD GHOST

This section consists of a list of the ccmmands understood by SERVICES and the BEAD GHOST. An attempt has been made to indicate what sort of parameter(s) each command expects, and some examples of the different kinds of parameters are given below. A few of the commands are understood by only one or the other of the dynamic duo, and they are so marked. The commands are written in caps, the parameters are underlined. The command and the parameters are separated by one or more blanks.

FIN is the command which terminates SERVICES; it is not understood by the BEAD GHOST

PURGE (BEAD GHOST only) aborts the current subsystem and returns to the Command Processor

RETRY (BEAD GHOST only) resumes execution of the current subsystem right where it quit

RETURN (BEAD GHOST only) resumes execution of the current subsystem without re-executing the most recent system call, if that call provoked an error

NEWPSW password changes the user's password to password

NEWDF direct:fname creates a file fname in the directory direct

ADDKEY keynum obits dirloc adds a lock which can be opened by the access key keynum to directory entry dirloc; the kinds of access allowed to the wielder of keynum are defined by obits

DELKEY keynum dirloc revokes privileges of access to the directory entry dirloc for holders of access key keynum

FRIENDP direct objloc if there is a permanent directory named direct, access to it is placed in objloc; the access is highly restricted

FRIENDT direct objloc same as FRIENDP, except temporary directories

PCAP object prints the indicated object

PDATA datum prints the indicated datum

PDATA datumloc datum prints datum words of data, starting at datumloc

MCAP object objloc places a link to object at objloc

MDATA datum datumloc moves datum to datumloc

CHAIN direct1 direct2 makes direct2 look like an extension of direct1

UNCHAIN direct eliminates any extension of direct

NEWV ident creates a new variable ident

KILLV ident eliminates the variable ident

BLST
DLIST
DLIST direct prints the contents of the directory direct

SPACE datum1 datum2 datum3 datum4 resources are reserved for the user; see section on space control

MSPACE direct1 datum direct2 datum sectors of disk space are moved from direct1 to direct2. One must be the father of the other. (One sector=64 words)

NEWU ident datum — creates new user subordinate to the user (i.e., a new permanent directory named ident of size datum as a son of the user's permanent directory)

KILLU ident — the permanent directory ident is eliminated from the user's permanent directory and destroyed

NEWDR ident datum — creates a directory ident on the user's permanent directory of size datum

NEWBLK filadr — creates a new file block at filadr

KILLBLK filadr — deletes the file block at filadr

NEWKEY objloc — creates a new access key at objloc

KILLOBJ object — deletes the indicated object

DELLINK dirloc — removes the link at dirloc

DELOWN dirloc — the cwnership entry at dirloc is removed and the owned object is destroyed

P.FULL — sets the print mode to print 20 successive octal digits

P.ASCII — sets the print mode to print 60 bit words in groups of 4,7,7,7,7,7,7,7,7, useful for decoding text files which the user has somehow been reduced to inspecting in octal

P.INST — sets the print mode to print octal digits in groups of 15, useful for dumping code files (this is the default mode)

IN.OCT — the mode of numbers typed into the Command Processor complex is to be octal if not expressly marked otherwise (this is the default mode)

IN.DEC — the mode of numbers typed into the Command Processor complex is to be decimal if not expressly marked otherwise.

MAKSUBP <dir> <dirloc>

WHO

CHARGES

TIME

DISPLAY [illegible]

MDOLS

MUSERS

? ETELL [illegible]

SHAZAM

SOFTL dirloc dirloc

RENAME ident dirloc

VIEW

~~KILLOBJ~~

READSEMSG

CØMMAND PROCESSOR

SERV
SERVICES
WHO
LOGOUT
LOGOFF
CHARGES
TIME

BUG
CRUNCH
BEADSBUG
GETBDFILE?
STOP
GONE
SYSTEMP
BIGTOY
USERBUG
JPROC

BILL
BRUCE
PAUL
KICK
FORCEOUT
SYSDOWN
DECIML ?

Parameters

datum parameters are evaluated to 60-bit integers; notice that if the user gives the name of a datum, the datum is looked up for him. Examples:

7	represents	7
11	represents	9, if 'IN.OCT'
11	represents	11, if 'IN.DEC'
5+10-15D	represents	-2, if 'IN.OCT'
VARIABLE+4	represents	7, if VARIABLE contains 3
7CB+(#52B+4)	represents	56 plus the contents of cell 46 in the subsystem which just call the BEAD GHOST

datumloc parameters specify places where data can be kept.

NAME	A variable called 'NAME'
FILE#0	The first word of a file FILE in the Command Processor name space
#10	Cell 8 of the subprocess calling the BEAD GHOST

direct parameters specify a directory

PERMDIR	The user's permanent directory
TEMPDIR	The user's termporary directory
USER:VV	The directory name 'VV' in the USER directory
USER:VV:P	The directory named 'P' in the directory named 'VV' in the etc.

dirloc parameters specify names of files in directories

TEMPDIR:INPUT	A file in the user's TEMPDIR
TEMPDIR:VV:REACT	A file in the directory named VV in the user's TEMPDIR

filadr parameters specity addresses within files

INPUT#0	Word 0 of a file INPUT named in the Command Processor name space
TEMPDIR:VV:REACT#100	Word 64 of the file mentioned above

fname is any legal file name; here are mentioned only strings of alphanumeric characters

```
INPUT
MYFILE10
```

ident is again, any string of aplhanumeric characters, blanks excluded keynumb is just a datum with a different name

301	Access key number 301
VARIABLE	Same, if VARIABLE=301

object is a two-word set of information which is the internal form of stuff kept by the system, like files and directories and access keys; if the user specifies an objloc, the object will be fetched

OWN.KEY	The user's private access key
SCANL	The user's private name space

objloc parameters specify places where objects are kept, such as directories and variables

VARNAME	The user can create a variable VARNAME and move objects to it
PERMDIR:FNAME	A dirloc is a special form of objloc