

CAL TSS  
COOK BOOK

## PART THREE - CAL TSS, the CAL Time-sharing System

### Chapter 1 - What CAL TSS is and how it runs

- 1.1 Introduction to CAL TSS
- 1.2 Sample session at the console: Log on, creating a text file with the editor, printing it on the line printer, log off

### Chapter 2 - User Subsystem

- 2.1 Command processor
- 2.2 File editor
- 2.3 Line editor
- 2.4 SCOPE simulator
- 2.5 Debugger
- 2.6 BASIC
- 2.7 BCPL
- etc. printer, reader, gettape, dumptape, etc.

### Chapter 3 - System architecture

- 3.1 Files
- 3.2 Directories
- 3.3 Processes; Subprocesses (F-return)
- 3.4 Capabilities and C-lists
- 3.5 Operation/calling the system
- 3.6 Event channels
- 3.7 System resources: control and accounting
- 3.8 Disk processes

### Chapter 4 - System Actions

### Chapter 5 - I/O Interfaces

## Preface

### Use of this manual

CAL TSS is a time-sharing operating system available to users of the Computer Center. Chapter 1 contains folksy bits of information to help the novice get acquainted with CAL TSS and get a feeling for its capabilities and usefulness. Chapter 2 tells how to talk to the system via the command processor and various subsystems currently available; parts of it will be essential to every user. Chapter 3 contains sufficiently detailed information about system concepts and structure to be of interest to a system programmer and can probably be skipped by the casual user without dire consequences. Chapter 4 gives the details of system-implemented actions which a user may invoke in code he writes. These actions may be considered as extensions to the 6400 hardware and are of interest mainly to subsystem-implementors and machine-language programmers. Chapter 5 gives details on the I/O interfaces which would allow a user to establish his own printer driver, for example.

## CHAPTER 1 - WHAT CAL TSS IS AND HOW IT RUNS

### 1.1 INTRODUCTION TO CAL TIME-SHARING SYSTEM

CAL TSS is a large-scale, general-purpose time-sharing system written by the Computer Center staff to run on a CDC 6000 series machine with ECS. The broad design goals of the system are:

1. to support up to 256 simultaneous interactive users at teletype-compatible terminals with fast response times for simple interactions, low system overhead, and good access to various hardware facilities;
2. to provide a file system allowing many files to reside permanently in the machine and to provide a very general, powerful framework within which such files can be accessed, shared, and protected;
3. to provide a system environment in which a large number of user-oriented subsystems can be developed and run;
4. to make possible a guaranteed response time for some subset of the users of the system;
5. to utilize efficiently the hardware represented by the Computer Center's 64003 system.

CAL TSS is called "large-scale" because it is in primary control (i.e., does not run under another system) of the computer, which is a large-scale machine. It is a "time-sharing" system because a large number of users at terminals may have programs active simultaneously and may each command responses from their programs on a time-scale of a few seconds. Finally, it is called "general-purpose" because the terminal user is not restricted to some particular programming language or set of programming languages; he may, in fact, program in machine language if he so desires. In general terms, the system provides facilities for the interactive user to

1. create files, preserve them in the system, retrieve and destroy them;
2. manipulate text files with a text editor;
3. process files with a number of subsystems provided, including



- a. a SCOPE simulator, giving access to all the facilities of the SCOPE system, including RUN Fortran, COMPASS, SNO-BOL, etc.;
  - b. a BASIC processor;
  - c. BCPL (a low level systems programming language);
4. prepare and run his own subsystems which may interact with his teletype and other subsystems;
  5. access the card reader, line printer, tape drives, and display console;
  6. give access privileges for his objects selectively to other users if he so desires and obtain privileges of access to objects of other users who wish to grant it.

When the terminal load on CAL TSS is low, another system facility will process a subset of the batch jobs normally processed by the SCOPE system. Other facilities can be implemented as determined by the needs of the computing community, the programmer time available, and the capacity of the hardware.

The structure of CAL TSS and the methods of using it are extensively described in the subsequent pages of this document. Here are briefly described concepts which all users of the system will have to deal with, whether his aim is to run Fortran or to implement a language processing system, whether his aim is to run FORTRAN or to implement a language processing system of his own.

Figure 1-1. IAL ( ) Hardware Configuration

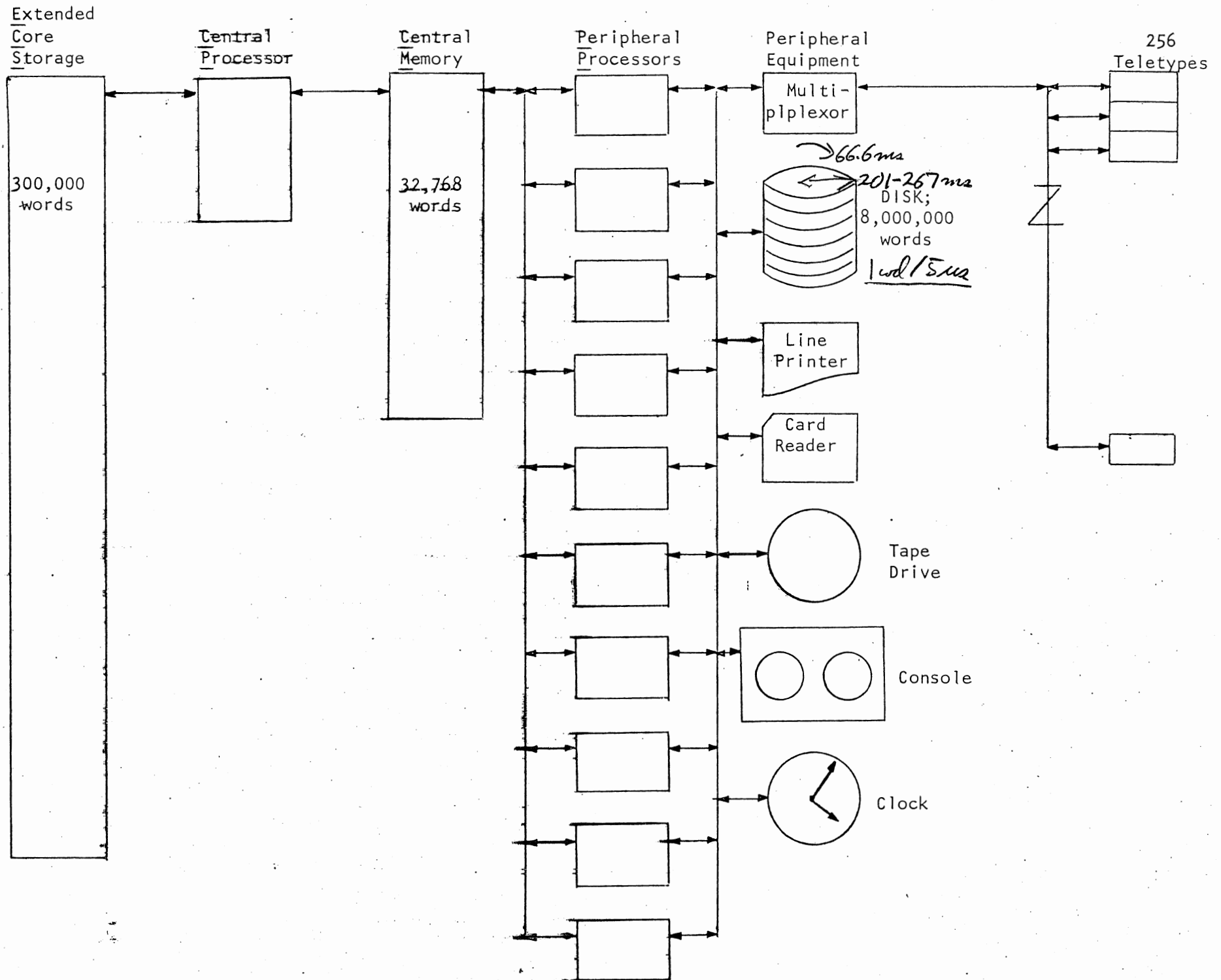
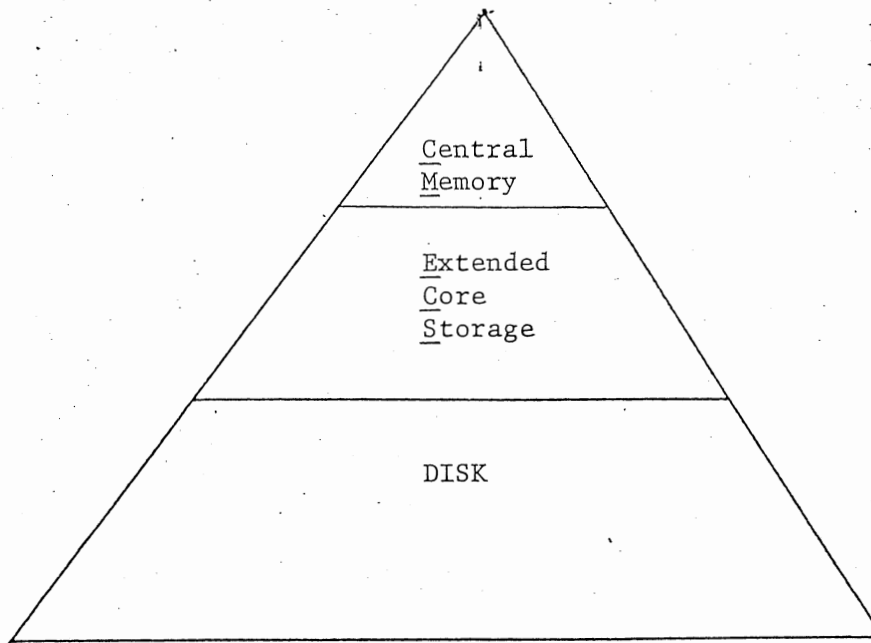


Figure 1-2. Storage Pyramid.



<u>Access Time</u> <u>For 1 word</u>	<u>Words</u> <u>Available</u>	<u>Cost per</u> <u>word</u>
1 $\mu$ s	32,768	\$1
4 $\mu$ s	300,000	\$1
depends on system load; many ms.	8,000,000	5¢

Figure 1.1 indicates the hardware configuration of the current 6400B system, on which CAL TSS is run. An exact understanding of all the boxes and their interconnections is, fortunately, unnecessary, but a brief description of the memory hierarchy will make dealing with the system more understandable. Note first that a file is the basic entity for storing information in the system; code ready for execution on the central processor and text ready to be fed to a language translator are both maintained in files.

Figure 1.2 represents as a pyramid the different storages present in the hardware. As the figure indicates, storage at the bottom is slow and cheap and big, while storage at the top is fast and expensive and, at present, small.

The disk, at the bottom of the pyramid, contains all the files which are accessible to the system without outside intervention such as mounting a tape or reading a card deck.

At the top, central memory contains the actively executing code of one process.<sup>1</sup> As the central processor is switched from one process to another, the code is swapped between CM and ECS. Thus, ECS must contain the code files for all processes currently active on the system.

Part of any file which is being manipulated by one of the active processes (an open file) may be in ECS or on the disk at the discretion of the process concerned. It may explicitly ask the system to maintain parts (blocks) of the file in ECS by attaching them and may dismiss blocks by detaching them. When a process accesses part of a file which is currently in ECS, the information is delivered immediately. Access to part of a file not in ECS causes the process to be blocked (stop running) until the system is able to bring the required information in from the disk. A process may, by attaching a block of a file in advance of its need to use it, improve its real-time processing speed at a cost of using more ECS.

Within this context, some of the terms with which every user will eventually become acquainted are now defined.

User profile: When someone makes arrangements to utilize CAL TSS, a body of information is recorded with the Computer Center business office which identifies him and describes his funding and access to

---

<sup>1</sup> The normal user at a terminal may consider all his interactions with the system to be carried out under the auspices of his own private process.

various system facilities. This information is called the user profile.

Log on/Log off: When a user attracts the attention of CAL TSS from a terminal, if his user profile and funding are in good shape, he is logged on. This is a procedure which gives him access to his own objects and such system objects and resources as his user profile allows. System resources, such as memory space, are reserved for his use at this point and may be charged against his account. When the user logs off, the resources are released and charging ceases.

File: As already noted, files are the basic entity for storing information in the system. Program code ready for execution as well as input to and output from language processors reside in files.

Directory: Directories are special objects which control access to other files, directories, and other system objects. Directories also provide the mechanism for associating symbolic names ("print names") with the objects controlled by the directory.

Permanent disk space: An amount of space determined by the user profile is permanently reserved for a user's files. It is the only system resource tied up by a user who is not currently logged on and is charged for continuously. It is controlled by the user's permanent directory.

Temporary disk space: When a user logs on, space on the disk, as determined by his user profile, is reserved to hold the temporary files he may need while running, such as output files and compiler scratch files. This space is controlled by his temporary directory.

Process: A process may be thought of as an organizational entity within the system which ties together certain code from files and other resources necessary to "carry out a task" or "run a program". CAL TSS creates a process for each user as he logs on the system. His process looks like a 6400 central processor with somewhat less than 32K of memory; the full range of 6400 CP instructions is available to it. In addition, the user's process is able to manipulate files and certain other system-defined objects in a general way. It is delivered already equipped with some code which can, for example, communicate with the user at his teletype. The private process created for the user is given access to his permanent directory (among others), thus giving him access to his files without giving access to other users.

Fixed ECS space: Various data relevant to the state of a process are kept in ECS by the system, as are certain other objects germane to its functioning. Also, the control information for open files is kept in ECS. Because the system has no facility for keeping such information on the disk, it is kept in what is called fixed ECS space. The amount

of fixed ECS which is set aside for a given user's process is determined by his profile when he logs on.

Swapped ECS space: Files and directories which are currently in ECS at the request of a user's process are kept in what is called swapped ECS, so that the system can free ECS space if it needs to by swapping the files out to the disk. The amount of swapped ECS available to each user is also determined by his profile at log in time.



# desiderata

- 1) don't echo passwords
- 2) squelch "waiting for space" messages
- 3) inter teletype communication
- 4) 'space obtained message'
- 5) change addkey to addlock
- 6) fix ETELL ("test of ETEL" !! ?!)
- 7) units on Charge, Logout



26 Aug - SHAZAM file1 file2 } add to services  
SOFTL dirle dirle }  
VIEW, KILL OBJ, DISPLAY

26 Aug - change name to CAL TSS cookbook?

" " prompt

" PURGE works in services



## CAL Time-Sharing System

Status and Information, 13 August 1971

### Availability

CAL TSS is currently available weekdays from 2-6 PM. There are 8 teletypes available for general use during these hours in Rooms 225 and 227 Campbell Hall. For information about connection of additional teletypes, contact Vance Vaughan (see below).

### Documentation

The fundamental document for users is the Introduction to CAL TSS, available from the Computer Center librarian. Other documentation is also available at the library, but it is spotty and users should consult with someone on the TSS staff before acquiring any.

### TSS Consultant

A member of the TSS staff is available in Room 225 Campbell Hall, ext. 2-5008, from 2-3 PM every weekday except Wednesday. He will answer questions, demonstrate the system, help new users through initial sessions, etc. Users unable to reach the consultant should contact Vance Vaughan, 207 Evans Hall, ext. 2-5823. He is there Thursdays from 1-2 PM, or by accident, or by appointment. Leave a message in the main Computer Center office, 239 Evans Hall, ext. 2-0851 to arrange an appointment.

### Getting help, reporting problems, etc.

Sections 1.7 and 1.9 of the Introduction to CAL TSS give procedures and information for diagnosing and understanding problems encountered when using the system. If the user's teletype is dead, or has gone crazy, he should first consult those sections. They may solve the problem, or be irrelevant, or give some such helpful advice as 'call a system programmer' or 'the teletype is down or the system is down'. If they are irrelevant, or say to contact a system programmer, or something like that, contact the TSS consultant (not the regular programming consultant). When the diagnosis is that the teletype is down or the system is down, the user should call the shift supervisor, (64)2-3043, and explain the problem. If the system is down, he will give information about when it will be up. If the system is up, there is some problem with the teletype or the line. The user should contact the person responsible for the maintenance of the teletype (Computer Center teletypes are maintained by Charles Cuffel, ext. 2-4403).

### Complaints and suggestions:

These should be made to the TSS consultant. The TSS staff is especially anxious to get feedback on the documentation. Corrections to content and suggested style modifications are both welcome.

July 1971

## INTRODUCTION TO CAL TSS

## Preface

## 1. General concepts

- 1.1 Access to CAL TSS
- 1.2 Files, directories
- 1.3 Login, logout
- 1.4 Command Processor, subsystems
- 1.5 Names, objects, name spaces, access locks, access keys
- 1.6 Command processor name space, BEAD name space, SCANL, PERMDIR, TEMPDIR, OWN.KEY, null key, PUB.KEY
- 1.7 SERVICES, BEAD GHOST, errors
- 1.8 Space control (what to do about 6,?,? errors)
- 1.9 'WHO' and PANICS, or how to untangle a console and how the user stops something he wishes he hadn't started
- 1.10 A note on the Line Collector (how to erase mistakes)

## 2. Examples

- 2.1 Use of BASIC, not keeping permanent files
- 2.2 Creation of a permanent disk file to be kept for future sessions:
  - 2.2.1 future access 'automatic'
  - 2.2.2 future access 'manual'
- 2.3 Access to permanent disk files
  - 2.3.1 Using EASIC on the file from example 2.2.1
  - 2.3.2.1 Selective access to permanent files
  - 2.3.2.2 Making all the user's permanent files available to all subsystems
- 2.4 SCOPE Simulator: a simple interactive FORTRAN program
- 2.5 SCOPE Simulator: an interactive SNOBOL program using a file from a friend's directory
- 2.6 Login problems illustrated

## 3. Subsystem summaries

- 3.1 EDITOR
- 3.2 BASIC
- 3.3 SCOPE
- 3.4 SERVICES and the BEAD GHOST

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 of 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.

July 1971

### 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

July 1971

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

July 1971

# !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.

July 1971

### 1.6 Command Processor name space, BEAD name space, SCANL name space, PERMDIR, TEMPDIR, PUBLIC, CWN.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 name 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 proceeds 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 logs out. 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

July 1971

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 to 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 from wreaking havoc. The existence of more than two 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.



July 1971

Table 2 - Name Spaces

COMMAND PROCESSOR NAME SPACE		SCANL <sup>1</sup> NAME SPACE		BEAD NAME SPACE <sup>2</sup>	
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 Processor proper is to minimize the number of reserved words which may not be used as names of user subsystem ('SERVICES', 'LOGOUT', etc.).

Unlike SERVICES, which is troublesome 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, error processing is initiated. Some errors are handled automatically by various subsystems along the way,

---

<sup>1</sup> methods for altering SCANL from the console are available.

<sup>2</sup> 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.

July 1971

and the user isn'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 for 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 to prevent space hogging. First, whenever a user returns to the Command Processor, he is automatically

July 1971

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.<sup>3</sup>  
 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.<sup>3</sup>

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

---

<sup>3</sup> 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.

July 1971

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.

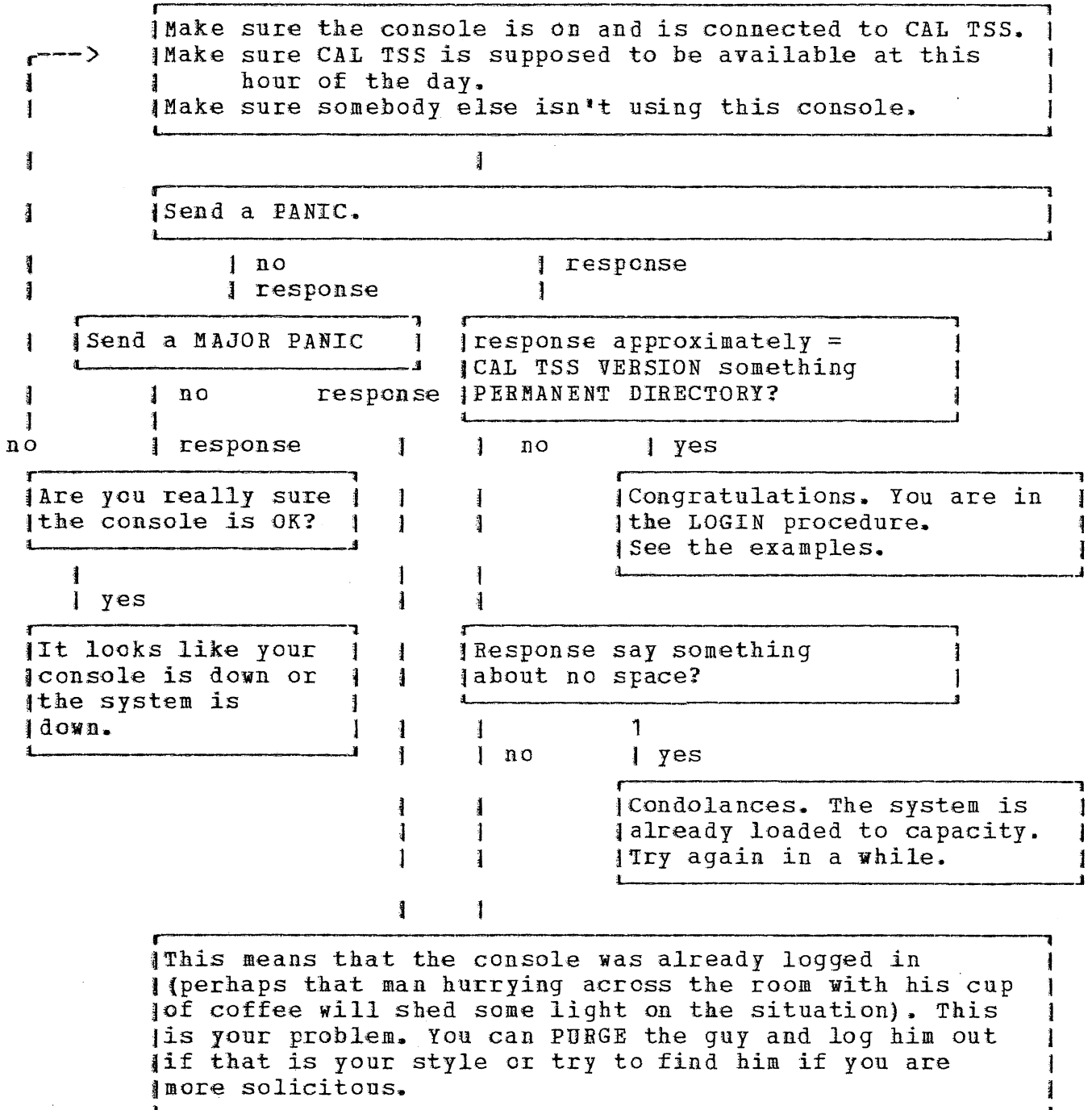
PROCEDURE II tells what the user does if he is already logged in and using the console but has either forgotten what he was doing or the console stopped responding the way he expects it to.

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 cminous 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.

July 1971

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



July 1971

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.)

|

no | If you have just typed something, did the characters echo  
r--< | (print)?

| yes |

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).

	no	response
	response	

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

no	response
----	----------

July 1971

| 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

| 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 this way. Either |

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

July 1971

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

Send a PANIC
--------------

no	response
response	

<p>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 REPLY.</p>
---

Send a MAJOR PANIC
--------------------

no	response
response	

<p>It looks like your your console or the system is down.</p>
---

<p>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.</p>
---



July 1971

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 suc- cessfully finish the login (see examples)
SERVICES	*	same as COMMAND PROCESSOR	'FIN'
BEAD GHOST (debugger)	@	same as COMMAND PROCESSOR	'PURGE' will return to the COMMAND PRO- CESSOR; 'RETRY' or 'RETURN' will return to the currently active subsystem.
EDITOR	:	????	'F' or 'Q' (see EDITOR document)
BASIC	: or ?	???? or miscellaneous diagnostics relevant to erroneous BASIC statements	same as EDITOR
SCOPE	(see SCOPE)	??NO??	'FIN'

July 1971

## 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 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 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

<u>type</u>	<u>meaning</u>	<u>and the teletype responds</u>
<u>CTRL-I</u>	make an insert at the beginning of the old line	<
10_	this is what is to be inserted	10_
<u>CTRL-O</u>	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

<u>type</u>	<u>meaning</u>	<u>and the teletype responds</u>
<u>CTRL-D</u>	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
<u>CTRL-Q</u>	erase the M	<-
N		N
<u>CTRL-H</u>	copy the rest of the old line into the new line	T_X

July 1971

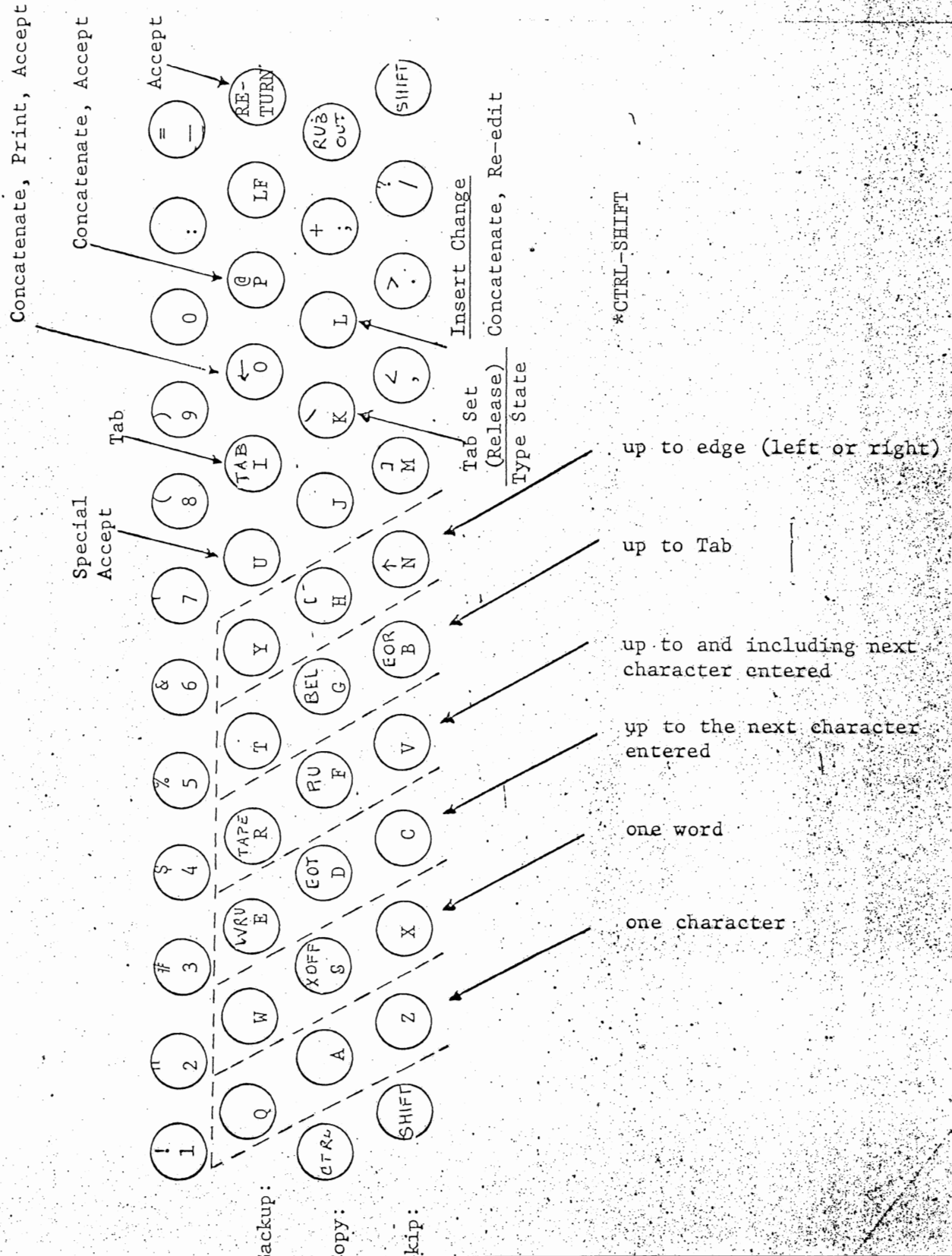
,Y        you remembered to print Y  
RETURN    you are satisfied with your  
            new line

,Y  
            and the car-  
            riage will  
            return

BASIC should accept this line, which is  
old line:    10 PRINT X,Y

July 1971

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



July 1971

## 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.

October 1971

## Example 2.1

	CAL TSS VERSION 2.0		
	20:35:14 10/21/71		
(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 2.0	(3.1)	
	-PRINT PI	(3.2)	
	3.141593		
	-10 LET X = 13		
	-20 LET Y = 19+8	(3.4)	
(3.3)	-30 PRINT X,YX*Y	(3.5)	
	ERROR OPERATOR MISSING		
	-30 PRINT X,Y,X*Y		
	-40 END		
	-RUN	(3.6)	
(3.0)	13 18	234	
	EXECUTION COMPLETE		
	-LIST 30	(3.7)	
	30 PRINT X,Y,X*Y		
	-EDIT 30	(3.8)	
	30 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		
	ILOGOUT	(4.1)	
	20:37:10 10/21/71		
	CONNECT TIME = 97782.		
	CPU TIME = 6311602.		
(4.0)	FIXED ECS = 344681550.		
	NOT SLOTS = 0.		
	SWAPPED ECS = 407565312.		
	TEMP DISK = 0.		
	MONEY = \$.297		
	GOOD DAY		

July 1971

## 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.

July 1971

This page no longer contains information.



July 1971

## Example 2.2

```

(1.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .ORBL
        TEMPORARY DIRECTORY?
        .V
(2.0) { COMMAND PROCESSOR HERE
        !SERVICES
        SERVICES HERE
        *NEWDF PERMDIR: AUTO (3.1)
        *PCAP OWN.KEY
        77777777777777777777 (3.2)
(3.0) { 0000000000000000000053002
        *ADDKEY 53002 77777777777777777777 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
        10 PRINT 10*PI
        20 PRINT 20*PI
        30 END
(4.0) { :F
        COMMAND PROCESSOR HERE
        !EDITOR M
        :I
        10 LET X = 10
        20 LET Y = 20
        30 PRINT X*PI, Y*PI
        40 END
        :F
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```

July 1971

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 2.0

20:40:39 10/21/71

PERMANENT DIRECTORY?

.USER:VV

GIVE PASSWORD

.QRBL

TEMPORARY DIRECTORY?

.V

COMMAND PROCESSOR HERE

IEASIC

BASIC VERSION 2.0

-LOAD AUTO

ERROR OPERATOR MISSING

20 PRINT 20\*PI

-LIST

10 PRINT 10\*PI

30 END

-20 PRINT 20\*PI

-RUN

31.41593

62.83185

EXECUTION COMPLETE

-SAVE AUTO

-FIN

COMMAND PROCESSOR HERE

!LOGOUT

20:41:43 10/21/71

CONNECT TIME = 47156.

CPU TIME = 7245765.

FIXED ECS = 166224900.

MOT SLOTS = 0.

SWAPPED ECS = 224351232.

TEMP DISK = 0.

MONEY = \$.296

GOOD DAY

1.0

1.1

1.2

1.3

1.4

1.5

1.6

1.7

July 1971

## 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 PIN leaves BASIC destroying the program in it.

October 1971

## Example 2.3.2.1

CAL TSS VERSION 2.0  
 20:42:27 10/21/71  
 PERMANENT DIRECTORY?  
 .USER:VV  
 GIVE PASSWORD  
 .QRBL  
 TEMPORARY DIRECTORY?  
 .V

①.0

COMMAND PROCESSOR HERE  
 !EDITOR MANUAL  
 :T;P\$

②.0

:Q  
 COMMAND PROCESSOR HERE  
 !SERVICES  
 SERVICES HERE  
 \*MCAP PERMDIR:MANUAL TEMPDIR:M  
 \*FIN

②.1

③.0

COMMAND PROCESSOR HERE  
 !BASIC  
 BASIC VERSION 2.0  
 -LOAD M  
 -RUN  
 31.41593            62.83185  
 EXECUTION COMPLETE  
 -FIN  
 COMMAND PROCESSOR HERE  
 !LOGOUT  
 20:43:58 10/21/71  
 CONNECT TIME = 71783.  
 CPU TIME = 10849976.  
 FIXED ECS = 253038600.  
 MOT SLOTS = 0.  
 SWAPPED ECS = 319674880.  
 TEMP DISK = 0.  
 MONEY = \$.443  
 GOOD DAY

July 1971

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.

October 1971

## Example 2.3.2.2

```

CAL TSS VERSION 2.0
20:46:04 10/21/71
PERMANENT DIRECTORY?
.USER:VV
GIVE PASSWORD
.QRBL
TEMPORARY DIRECTORY?
.V
COMMAND PROCESSOR HERE
!SERVICES
SERVICES HERE
*CHAIN PERMDIR TEMPDIR
*UNCHAIN PERMDIR
*CHAIN TEMPDIR PERMDIR
*FIN
COMMAND PROCESSOR HERE
!BASIC
BASIC VERSION 2.0
-LOAD MANUAL
-RUN
  31.41593      62.83185
EXECUTION COMPLETE
-FIN
COMMAND PROCESSOR HERE
!LOGOUT
20:47:14 10/21/71
CONNECT TIME = 52511.
CPU TIME = 7699295.
FIXED ECS = 185104800.
MOT SLOTS = 0.
SWAPPED ECS = 247353344.
TEMP DISK = 0.
MONEY = $.317.
GOOD DAY

```

(1.0)

(1.1)

(1.2)

(1.3)

(2.0)

July 1971

## 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 TMPDIR 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, TMPDIR, 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 TMPDIR, which is what the user was trying to do. If he hadn't unchained PERMDIR from TMPDIR 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.



July 1971

## Example 2.4

```
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
  ISERVICES
  SERVICES HERE
  *CAP PERMDIR:TRIVIA TEMPDIR:INPUT
  *FIN
```

(1.0)

July 1971

```

(2.0) { COMMAND PROCESSOR HERE
        !EDITOR INPUT
        :T;P$
        (2.1) PROGRAM TRIV(TTYIN,TTYOUT,TAPE2=TTYIN,TAPE1=TTYOUT)
        WRITE (1,100)
        100  FORMAT (*TRIVIA SPEAKING, WHO'S THERE?*)
        READ (2,200) NAME
        200  FORMAT (A10)
        WRITE (1,300) NAME
        300  FORMAT (*GOODBYE,*A10)
        END
        :Q
        COMMAND PROCESSOR HERE
        !SCOPE 40000 (3.1)
        15:42:35 08/06/71 SCOP32C OF 08/01/71
        >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
        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

```

July 1971

## 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 TMPDIR 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 compiler 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 ccmmands 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.

July 1971

## Example 2.5

```

(1.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .GUEST
        GIVE PASS WORD
        .GUEST
        TEMPORARY DIRECTORY?
        .VANCE
        COMMAND PROCESSOR HERE
        ISERVICES
        SERVICES HERE
        *PCAP OWN.KEY
        7777777777777777002737
        0000000000000000123401 } (1.1)
        *FIN
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```

```

(2.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .ORBL
        TEMPORARY DIRECTORY?
        .VANCE
        COMMAND PROCESSOR HERE
        ISERVICES
        SERVICES HERE
        *ADDKEY 123401 71420 PERMDIR:REACT
        *ADDKEY 123401 71420 PERMDIR:DATA } (2.1)
        *FIN
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```

July 1971

(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

UNEXPECTED FRETURN

\*FRIENDP USER:VV

BAD SYNTAX

\*FRIENDP USER:VV TEMPDIR:VV

BAD SYNTAX

\*FRIENDP USER:VV TEMPDIR:VV

\*MCAP VV:REACT;OWN.KEY PERMDIR:REACT

\*MCAP VV:DATA;OWN.KEY PERMDIR:DATA

\*ADDKEY 123401 7777777777777777 PERMDIR:REACT

\*ADDKEY 123401 7777777777777777 PERMDIR:DATA

\*FIN

COMMAND PROCESSOR HERE

!LOGOUT

GOOD DAY

(3.1)

(3.2)

(3.3)

(3.4)

July, 1971

CAL TSS VERSION 1.2  
PERMANENT DIRECTORY?

.GUEST

GIVE PASS WORD

.GUEST

(4.0) TEMPORARY DIRECTORY?

.VANCE

{ COMMAND PROCESSOR HERE

ISCOPE

(4.1) { 16:19:54 08/06/71 SCOP32C OF 08/01/71

>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 THANK

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

↑6425 823

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

↑SUMMER

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

July 1971

FISHERMAN'S BOAT DRIFTS  
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

{COMMAND PROCESSOR HERE

{!LOGOUT

{GOOD DAY

(4.3)

July 1971

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

(1.0) { 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  
 !LOGOUT  
 GOOD DAY

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

July 1971

EXAMPLE 2.6 - LOGIN PROBLEMS ILLUSTRATED

- 1.0 When the user sent his CTRL-SHIFT-R 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 RETURN' 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.

November 1971

### 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 <u>sc</u>	Delete the specified lines.
T	Move to the top of the file (pseudo-line).
M <u>sc</u>	Move forward over the specified lines.
B <u>sc</u>	Move backward the specified number of lines. (NOTE: <u>sc</u> can only be a number.)
P <u>sc</u>	Print the specified lines.

November 1971

<u>C/str1/str2/sc</u>	Replace the first occurrence of <u>str1</u> by <u>str2</u> in the specified lines.
<u>CG/str1/str2/sc</u>	Replace every occurrence of <u>str1</u> by <u>str2</u> in each of the specified lines.
<u>Esc</u>	<u>Edit</u> the specified lines using the Line Collector.*
<u>R,fname</u> <sup>s</sup>	Insert the contents of file <u>fname</u> after the current line.
<u>W,fname</u> <sup>s</sup> , <u>,sc</u>	Write the specified lines, including the current line, into the file <u>fname</u> .
<u>F,fname</u> <sup>s</sup>	Finished - create the file <u>fname</u> from the latest version; simply entering 'F' causes the updated text to replace the original file <u>fname2</u> specified when the Editor was called.
<u>Q</u>	Finished but do not save any file.

The Editor prompts with : and responds ???? 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.)

<sup>s</sup> If fname is null a CP name is requested on the next line.

July 1971

*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).
D <u>sc</u>	Delete the specified lines.
T	Move to the top of the file (pseudo-line).
M <u>sc</u>	Move forward over the specified lines.
B <u>sc</u>	Move backward the specified number of lines. (NOTE: <u>sc</u> can only be a number.)
P <u>sc</u>	Print the specified lines.
C/ <u>str1</u> / <u>str2</u> / <u>sc</u>	Replace the first occurrence of <u>str1</u> by <u>str2</u>

July 1971

*see other 43*

<u>CG/str1/str2/sc</u>	in the specified lines. Replace every occurrence of <u>str1</u> by <u>str2</u> in each of the specified lines.
<u>Esc</u>	<u>Edit</u> the specified lines using the Line Collector.*
<u>R,fname</u>	Insert the contents of the file <u>fname</u> after the current line.
<u>W,fname,,sc</u>	Write the specified lines, including the current line, into the file <u>fname</u> .
<u>F,fname</u>	Finished - create the file <u>fname</u> from the latest version; simply entering 'F' causes the updated text to replace the original file specified when the Editor was called.
<u>Q</u>	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.)

July 1971

### 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 'LCAD' 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
```

July 1971

```

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<sup>5</sup> 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

---

<sup>5</sup> 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.



July 1971

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 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.

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.

July 1971

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.

July 1971

# 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 number] [line number] [line number]~~  
~~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.

July 1971

OperatorsArithmetic

| 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|  
 ACS(X) arcos(x)  
 ASN(X) arcsin(x)  
 ATN(X) arctan(x)  
 COS(X) cos(x)  
 EXP(X) e  
 INT(X) integer  
 LOG(X) log x

LGT(X) log x  
 RND(X) random num  
 SGN(X) sign(x)  
 SIN(X) sin(x)  
 SQR(X) x  
 TAN(X) tan(x)  
 TIM(X) seconds used

July 1971

### 3.3 Summary of the SCOPE Simulator

SCOPE provides an operating environment for 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 f1

where f1 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 from 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

<u>Request</u>	<u>Meaning</u>
TEXT, <u>fname</u>	Declare a new SYSTEXT file <u>fname</u> (will not change to SYSTEXT a file which already exists in another mode).
FILE, <u>fname</u>	Use the file <u>fname</u> as the source of SCOPE Simulator requests.
MSG,OFF or ON	Suppresses program messages to the console or restarts them.
GET, <u>fname</u>	Get the file <u>fname</u> from the BEAD NAME SPACE.
PUT, <u>fname</u>	Return <u>fname</u> to 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 SCOPE Simulator.

#### Loading requests:

	<u>Meaning</u>
L, <u>fname</u>	Load and link the file <u>fname</u>
LGO, <u>fname</u>	Load and link the file <u>fname</u> and start the resulting code executing

July 1971

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

July 1971

3.4 SERVICES and the BEAD GHOST

This section consists of a list of the commands 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
<u>NEWPSW password</u>	changes the user's password to <u>password</u>
<u>NEWDF direct:fname</u>	creates a file <u>fname</u> in the directory <u>direct</u>
<u>ADDKEY keynum obits dirloc</u>	adds a lock which can be opened by the access key <u>keynum</u> to directory entry <u>dirloc</u> ; the kinds of access allowed to the wielder of <u>keynum</u> are defined by <u>obits</u>
<u>DELKEY keynum dirloc</u>	revokes privileges of access to the directory entry <u>dirloc</u> for holders of access key <u>keynum</u>
<u>FRIENDP direct objloc</u>	if there is a permanent directory named <u>direct</u> , access to it is placed in <u>objloc</u> ; the access is highly restricted
<u>FRIENDT direct objloc</u>	same as FRIENDP, except temporary directories
<u>PCAP object</u>	prints the indicated <u>object</u>
<u>PDATA datum</u>	prints the indicated <u>datum</u>
<u>PDATA datumloc datum</u>	prints <u>datum</u> words of data, starting at <u>datumloc</u>
<u>MCAP object objloc</u>	places a link to <u>object</u> at <u>objloc</u>
<u>MDATA datum datumloc</u>	moves <u>datum</u> to <u>datumloc</u>
<u>CHAIN direct1 direct2</u>	makes <u>direct2</u> look like an extension of <u>direct1</u>
<u>UNCHAIN direct</u>	eliminates any extension of <u>direct</u>
<u>NEWV ident</u>	creates a new variable <u>ident</u>
<u>KILLV ident</u>	eliminates the variable <u>ident</u>
<u>DLIST direct</u>	prints the contents of the directory <u>direct</u>
<u>SPACE datum1 datum2 datum3 datum4</u>	resources are reserved for the user; see section on space control
<u>MSPACE direct1 datum direct2 datum</u>	<u>datum</u> sectors of disk space are moved from

B4BT  
DLIST



July 1971

<u>NEWU ident datum</u>	<u>direct1</u> to <u>direct2</u> . One must be the father of the other. (One sector=64 words) creates new user subordinate to the user (i.e., a new permanent directory named <u>ident</u> of size <u>datum</u> as a son of the user's permanent directory)
<u>KILLU ident</u>	the permanent directory <u>ident</u> is eliminated from the user's permanent directory and destroyed
<u>NEWDR ident datum</u>	creates a directory <u>ident</u> on the user's permanent directory of size <u>datum</u>
<u>NEWBLK filadr</u>	creates a new file block at <u>filadr</u>
<u>KILLBLK filadr</u>	deletes the file block at <u>filadr</u>
<u>NEWKEY objloc</u>	creates a new access key at <u>objloc</u>
<u>KILLOBJ object</u>	deletes the indicated <u>object</u>
<u>DELLINK dirloc</u>	removes the link at <u>dirloc</u>
<u>DELOWN dirloc</u>	the ownership entry at <u>dirloc</u> is removed and the owned object is destroyed
<u>P.FULL</u>	sets the print mode to print 20 successive octal digits
<u>P.ASCII</u>	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
<u>P.INST</u>	sets the print mode to print octal digits in groups of 15, useful for dumping code files (this is the default mode)
<u>IN.OCT</u>	the mode of numbers typed into the Command Processor ccomplex is to be octal if not expressly marked otherwise (this is the default mode)
<u>IN.DEC</u>	the mode of numbers typed into the Command Processor complex is to be decimal if not expressly marked otherwise.

MAKSUBP &lt;dir&gt; &lt;dirloc&gt;

WHO

CHANGES

TIME

DISPLAY permanent directory name

MDOLS

MUSERS

P ETELL dirloc

SHAZAM!

SOFTL dirloc dirloc

RENAME ident dirloc

VIEW

KILLOBJ

BEADSEMSG

COMMAND PROCESSOR

SERV

SERVICES

WHO

LOGOUT

LOGOFF

CHARGES

TIME

BUG

BILL

CRUNCH

BRUCE

BEADBUG

PAUL

GETBDFILE?

KICK

STOP

FORCEOUT

GONE

SYSDOWN

SYSTEMP

DECIML ?

BIGTOY

USERBUG

JPROC

July 1971

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 temporary 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 specify 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 alphanumeric characters, blanks excluded

July 1971

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

effective Nov 25 1971

PROPOSED CHARGES FOR TSS 10/8/71

- I) A basic machine rate of \$200 per hour, divided up as proposed in "A note on charging" of 8/17/71. This results in the following rates: (rounded to 2 figures.)

CPU	\$130.00 per hour
ECS	\$ .29 per $10^3$ words-hour
Temporary disk	\$ .047 per $10^3$ -sector-hour

- II) A connect charge rate so set that the effective connect charge for no activity is

Direct wired TTY	\$2.00 per hour
Dial up lines	\$2.50 per hour

- III) Permanent disk space charged at the effective rate of temporary disk space for 200 hours per month. This rate will not depend on the number of hours the system is actually in service.

Permanent disk space \$5.40 per  $10^3$  -sector-month

ABORT  
>FIN  
COMMAND PROCESSOR HERE  
!PRINTER  
TSS DISK SYSTEM PRINTER DRIVER VER 4.1  
ENTER TITLE LINE  
FOR VANCE  
SCOPE FORMAT CONVENTIONS ?  
Y  
EVCH CLEARED  
PRINTING ...  
COMMAND PROCESSOR HERE  
!LOGOUT  
16:20:13  
08/24/71  
42607433  
2023714345  
421550564744  
0  
0  
0  
46205  
GOOD DAY

*connect  
CPU*

*Fixed ECS  
Swapped ECS  
?  
?*

*dollars in  
mils in total ~\$16*

*logged in for several  
hours, <sup>edited,</sup> assembled  
SUBPROC trace,  
listed it once*



Cookbook ~ \$1

User's Guide, Vol IV, Part 3 ~ \$8-10

Permanent disk/month: 30¢ small (800 words, ~12,13 <sup>sectors</sup> ~~tracks~~)  
~ 3¢/sector/month 7.40 med (20K words, ~313 ~~tracks~~ <sup>sectors</sup>)  
44.00 large (120K words, ~2000 sectors)

Connect time \$3.00/hr

Scope Simulator/hr \$210

null cell .70

null Kumpare 1.17

large " 10.50

\$1/month 2500 words  
20,000 characters

A NOTE ON CHARGINGINTRODUCTION

Within the next few weeks it will be possible to begin charging for the use of TSS. In fact we will probably begin testing the charging procedures on live customers within a week. This note will explore various rationales for setting rates and the resulting charges for a few standard tasks.

RATIONALES

There are a number of different ways to set rates. The method used in this note is to set a nominal rate for the machine as a whole. This rate is then partitioned among the various resources for which we can account. The actual rate for the machine as a whole will then depend on the percentage of utilization of each of these accountable resources. The nominal rate for the machine as a whole is then adjusted on an ad hoc basis to obtain the desired income.

Another method, not used in this note, is to attempt to set the rate for each piece of equipment so as to pay its costs. Thus, for example, if disk storage becomes full there would be enough income from the disk itself to pay for another. This method has a number of problems. Not all components can be extended at will and all components are not accounted. Moreover, this method makes no allowance for various overhead costs, such as system programmers. It might be worthwhile to examine this method later.

Once a nominal overall rate has been set, it must be partitioned among the accountable components. The obvious method that presents itself is to divide the rate proportional to the cost of each component. This leads to similar problems to the other method, i.e., not all components can be accounted, and not all of each accounted component can be accounted. Another problem is that one of the accountable items, connect time, is not a component.

We can charge for part of each of the major components: CPU, ECS and the disk. One procedure is to just ignore all other components, and that is what is done in this note. Another question is whether to weight each component by its full cost and require the accountable portion to pay for the full weight, or to weight each component by the cost of its accountable portion. This is a real problem for ECS since we can only account for half of it. Finally, there is the question of how to determine the cost for a component, by the original cost to the Computer Center, or by a probable replacement cost. These last two questions lead to four possibilities, and in later computations all of them are examined.



STANDARD TASKS

There are a number of standard tasks considered in this note. The first is the null task. The cost for just logging in to the system and doing nothing, the so-called connect charge. This must be distinguished from a charge for connect time alone, since while a TTY is logged in it is sitting on some resources that are accounted. The total connect charge would be the sum of the charge for connect time alone and the charge for the resources reserved to the TTY.

It will turn out later, under the charging assumptions made in this note, that the charge for the resources reserved to a logged in TTY are quite high, due to a number of inefficiencies in the current system. It is hoped that they will be reduced in the next six months. In the meantime we propose a negative charge for connect time, so as to reduce the total connect charge to a reasonable amount. This amounts to simply permitting the free use of a minimum amount of ECS, and then charging for any additional.

The total resources used by a logged in TTY are .7.5K (decimal) ECS and some disk space. The ECS space comes from two types of overhead. The first is system overhead of 4K of fixed ECS space, which should reduce over the next six months. The second is due to a crude algorithm for the control of swapped ECS space, which also should reduce to near zero under the forced swapped procedures to be installed late this year. The cost of the disk space is ignored, since it is at most \$.10.

The next tasks considered will involve the use of the SCOPE Simulator. All estimates of charging assume that ECS will be used for the same length of time as is the CPU. This would be approximately true if only one user is on the system. Since ECS costs will be low compared with CPU costs, the errors can be ignored under low load. Under high load the increase in cost for ECS will become significant, which may lead users to avoid the system during high load.

All of these SCOPE tasks will assume a field length of 45K (octal), sufficient for large assemblies. Under these conditions, the SCOPE Simulator requires 100K (octal) of swapped ECS. In order to run large assemblies more disk space than the nominal amount for a logged in TTY must be reserved. Compared to the cost for CPU this will be small, on the order of \$10.00 an hour, and therefore will be neglected.

The particular SCOPE tasks will be: null, null assembly, and large assembly. Under the current system the null use of the SCOPE Simulator, call and return, takes about 12 seconds. Running a null NOMPASS assembly raises the total to about 20 seconds.

The large assembly is one of the decks used in the system itself. The cost on the A machine for assembling the deck under MOMPASS is 1 minute of CPU time and 80 seconds of PPU time. At \$400 per hour on the A machine the charge would be about \$6.70. Under the current TSS system the cost seems to be about 3 minutes, including the overhead for calling the SCOPE Simulator.

The final task to be considered is that of permanent file storage. We will attempt to compute the costs on a per month basis. Since the system accumulates charges for disk space only while the system is actually running, we need an estimate of the number of hours the system will be up in a month. Since a month has about 700 hours, it is unlikely that the system will be up more than 350 hours.

Next we need estimates of the amount of permanent disk space needed. These are given for three classes of users: small, medium and large. The small user is a student with a very small program to save, say about 1 or 2 pages. We estimate that about 800 words or 12 sectors will be sufficient. The medium user has maybe a 50 page program, about 20 thousand words or 300 sectors. Finally, the large user has maybe 120 thousand words, or about 1800 sectors. (The system cannot support many large users.)

#### RATE SETTING

In order to set a rate by the method chosen, we need a nominal rate for the whole machine. For this purpose we have chosen \$400 per hour so as to compare with the A machine, which is slightly more than \$400 an hour. The B machine is probably somewhat cheaper than the A machine since it does not have as much central memory, nor as many printers or tape drives. The operator overhead for running TSS is considerably less than the SCOPE system, at present requiring an operator only at start up and shutdown, plus an occasional tape mount. The programming staff for TSS is also somewhat smaller than for the A machine.

The TSS should give a much higher rate of utilization for the accounted components than does the SCOPE system, thus inherently producing a higher rate of income for a fully loaded system. (\$400 per hour, 24 hours per day, 365 days per year amounts to  $3.5 \times 10^6$ .)

In view of the results that follow, we suggest starting with a basic rate of less than \$400 per hour if the desire is to produce a competitively priced system.

Having chosen a basic rate, we need the comparative costs of the various components in order to divide up the rate. The following table contains the basic information used in the subsequent calculations. These values were obtained from Ken Hebert on August 12, 1971.

<u>component</u>	<u>original cost</u>	<u>probable replacement cost</u>	<u>standard unit</u>	<u>total standard units (NSU)</u>	<u>accountable units (NASU)</u>
CPU + 32K CM	.69*10 <sup>6</sup>	.5*10 <sup>6</sup>	1 hour	1	.9
300K ECS	.48*10 <sup>6</sup>	.6*10 <sup>6</sup>	1 K-words-hr	300	160
1/2 disk	.26*10 <sup>6</sup>	.10*10 <sup>6</sup>	1 K-sector- hr	1020	765
other	.08*10 <sup>6</sup>	.10*10 <sup>6</sup>	--	--	--

The following tables contain the computation of the rates per standard unit under different assumptions. The following symbols are used:

R = total rate to be distributed (using \$400 per hour)  
 cc = component cost  
 tc = total cost for all components  
 nsu = number of standard units in the component  
 nasu = number of accountable standard units in the component  
 ac = cost of accountable portion of a component =  $cc \times nasu/nsu$   
 tac = total cost of accountable portions of components

Next we give the formulas for the two methods:

#### Method A

$$\text{rate/s.u.} = R \times (cc/tc) \times (1/nasu)$$

#### Method B

$$\text{rate/s.u.} = R \times (ac/tac) \times (1/nasu)$$

Method A

component	(original costs)			(replacement costs)		
	cc	cc/tc	rate/su	cc	cc/tc	rate/su
CPU + 32K CM	.69	.46	\$ 205.	.5	.38	\$ 170.
300K ECS	.48	.32	.80	.6	.46	1.15
1/2 disk	.26	.17	.089	.1	.08	.042
other	.08	.05		.1	.08	
tc =	1.51			1.3		

Method B

component	ac	ac/tac	rate/su	ac	ac/tac	rate/su
CPU + 32K CM	.62	.59	\$ 263.	.45	.53	\$ 236.
300K ECS	.25	.23	.58	.32	.38	.95
1/2 disk	.19	.18	.094	.08	.09	.047
	1.06			.85		

COSTS PER TASK

The following table gives the charges for the tasks described above in the 4 cases of rate division considered.

	<u>original costs</u>		<u>replacement costs</u>	
	<u>Method A</u>	<u>Method B</u>	<u>Method A</u>	<u>Method B</u>
connect time per hour	\$ 5.24	\$ 4.05	\$ 10.50	\$ 6.75
SCOPE Simulator per hour for CPU and ECS	230.00	281.00	207.00	266.00
null call	.77	.94	.69	.89
null NOMPASS assembly	1.28	1.56	1.15	1.48
large NOMPASS assembly (\$6.70 on A machine)	11.50	14.00	10.04	13.30
Disk storage/month (at 350 hours/month)				
small (800 wds) = 10000	.37	.40	.18	.20
medium (20K wds)	9.40	9.90	4.40	5.00
large (120K wds)	56.00	59.00	26.00	30.00

FINAL

After consideration of the above results, we propose that:

- 1) the basic charge rate be \$300 per hour.
- 2) Method B, based on original costs, be used to calculate the rates.
- 3) a negative connect charge be used so that the basic charge for connect time is about 1 or 2 dollars an hour.

These proposals result in the following rates:

CPU	\$200.00 per hour	5.864/100
ECS	0.43 per K-hour	
disk	0.07 per K-sector-hour	

and the following charges for the tasks considered above:

connect time per hour	\$ 3.01 (not including negative connect fee)
SCOPE Simulator per hour	210.00
null call	.70
null NOMPASS assembly	1.17
large NOMPASS assembly	10.50
Disk storage/month	
small	.30
medium	7.40
large	44.00

BEADS & ERRORS  
ALL THINGS LIKE CLASS 17<sup>4</sup><sub>8</sub>

NUMBER 1 - building subprocess

MOD

- 00 - error when reading subprocess description
- 05 - error when getting space
- 10 - error when looking up class code  
(could be bad name)
- 20 - error when creating list
- 30 - error when creating the subprocess
- 40 - can not find scratch file - should  
not even happen
- 45 - error on opening scratch file
- 50 - error when reading shape of scratch file
- 60 - too many blocks in scratch file
- 70 - error while probing for end  
of scratch file

- 100 - error while creating block or scratch file
- 110 - error while reading map description
- 120 - can not find file to put in map
- 124 - error while opening file read only to put in map
- 125 - error while opening file read write to put in map
- 130 - error while making map entry -  
BEHDSMSG is very helpful
- 140 - error while reading chit description
- 150 - can not find object requested in C list
- 160 - error while moving csp into chit
- 170 - error while mapping call operation  
(old style call operation)

- 171 - error while creating new style calloperata  
with black cap
- 172 - error while making parameter block  
datum on cap
- 200 - error while putting in fixed C list  
entry

NUMBER 4 - can not create or find directory  
entry for capability to be put in  
directory

NUMBER 5 - can not do a bend locate  
on an object. most common  
reason is that user name is bad



NUMBER 6 - error while closing down  
after subprocess return

MOD

10 - can't find scratch file

15 - error closing scratch file

20 - error reading map description

30 - can't find file in map entry

40 - error closing file in map

NUMBER 7 - error killing an object  
see BEADSEMSG for details

NUMBER 10 - error while cleaning out  
a directory

MOD

50 - ownership entry not removable

10 - error while deleting a link

NUMBER 11 - BEADS crashed - got  
an error when it shouldn't - see BEADSEMSG

NUMBER 12 - first logon call

MODIFIER

00 - logon has already been done -  
doubtful this will ever happen

10 - permanent dirctry is not an accounting  
dirctry

20 - the dirctory does not have a  
profile ~~on~~ on it but is an  
accounting dirctory

40 - profile is in a bad state

50 - password does not match

NUMBER 13 - second login call

MOD

00 - already logged on (same as 17, 12, 0)

10 - profile is out of money

---

NUMBER 14 - removing a profile from  
a directory - can be caused by  
KILLJOB or DELOWN of directory

MOD

00 - attempt to delete root profile  
(ie father pointer is 0)

10 - profile still has sons

20 - someone is currently logged on  
to that profile

NUMBER 15 - change a users password

MOD

00 - directory passed is not an accounting directory

10 - directory passed does not have a profile

---

NUMBER 16 - error when trying to create a subuser

MOD

00 - not the perm-dir of the currently logged on user.

03 - block parameter is not big enough

05 - directory size is negative

06 - directory size is too small

- MOD

- 10 - account number is negative
- 12 - account number  $\geq 2^{15}$
- 15 - attempt to create user with new account number without flag F.NACCT on ~~in~~ users own profile
- 20 - one of the limits for the new user is greater than the fathers
- 25 - new user has more flags than his father
- 27 - negative amount of perm disk space passed
- 30 - tried to give user more disk space than there is available
- 35 - negative money to be given

- 40 - attempt to give new user more money than is available
- 50 - negative number of subusers passed
- 60 - no subusers available and/or attempt to give away more than are available
- 70 - error on the operation to create the new users directory. BEADS EMSG will give the error class and number of the error that occurred likely errors are:  
duplicate name  
directory full  
directory size too big  
identifier bad

NUMBERS 17

20

moving dollars  
moving subusers

MOD

00 - donor directory not accounting

10 - donor directory does not have profile

20 - receiving directory not accounting

30 - receiving directory does not have profile

40 - negative amount given

50 - users are not in a father son relationship

---

NUMBER 21 - display profile

MOD

00 - directory not accounting

10 - directory does not have a profile

List corrections, suggestions, etc., here.



July 1971

## INTRODUCTION TO CAL TSS

### Preface

#### 1. General concepts

- 1.1 Access to CAL TSS
- 1.2 Files, directories
- 1.3 Login, logout
- 1.4 Command Processor, subsystems
- 1.5 Names, objects, name spaces, access locks, access keys
- 1.6 Command processor name space, BEAD name space, SCANL, PERMDIR, TEMPDIR, OWN.KEY, null key, PUB.KEY
- 1.7 SERVICES, BEAD GHOST, errors
- 1.8 Space control (what to do about 6,?,? errors)
- 1.9 'WHO' and PANICS, or how to untangle a console and how the user stops something he wishes he hadn't started
- 1.10 A note on the Line Collector (how to erase mistakes)

#### 2. Examples

- 2.1 Use of BASIC, not keeping permanent files
- 2.2 Creation of a permanent disk file to be kept for future sessions:
  - 2.2.1 future access 'automatic'
  - 2.2.2 future access 'manual'
- 2.3 Access to permanent disk files
  - 2.3.1 Using BASIC on the file from example 2.2.1
  - 2.3.2.1 Selective access to permanent files
  - 2.3.2.2 Making all the user's permanent files available to all subsystems
- 2.4 SCOPE Simulator: a simple interactive FORTRAN program
- 2.5 SCOPE Simulator: an interactive SNOBOL program using a file from a friend's directory
- 2.6 Login problems illustrated

#### 3. Subsystem summaries

- 3.1 EDITOR
- 3.2 BASIC
- 3.3 SCOPE
- 3.4 SERVICES and the BEAD GHOST

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 of 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.

July 1971

### 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

July 1971

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

July 1971

**!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.

July 1971

### 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 name 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 proceeds 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 name 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

July 1971

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 to 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 from wreaking havoc. The existence of more than two 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.

July 1971

Table 2 - Name Spaces

COMMAND PROCESSOR NAME SPACE		SCANL <sup>1</sup> NAME SPACE		BEAD NAME SPACE <sup>2</sup>	
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 Processor proper is to minimize the number of reserved words which may not be used as names of user subsystem ('SERVICES', 'LOGOUT', etc.).

Unlike SERVICES, which is troublesome 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, error processing is initiated. Some errors are handled automatically by various subsystems along the way,

-----  
<sup>1</sup> methods for altering SCANL from the console are available.

<sup>2</sup> 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.



July 1971

and the user isn'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) MCT 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 for 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 to prevent space hogging. First, whenever a user returns to the Command Processor, he is automatically

July 1971

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.<sup>3</sup>
  - 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.<sup>3</sup>

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

---

<sup>3</sup> 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.

July 1971

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.

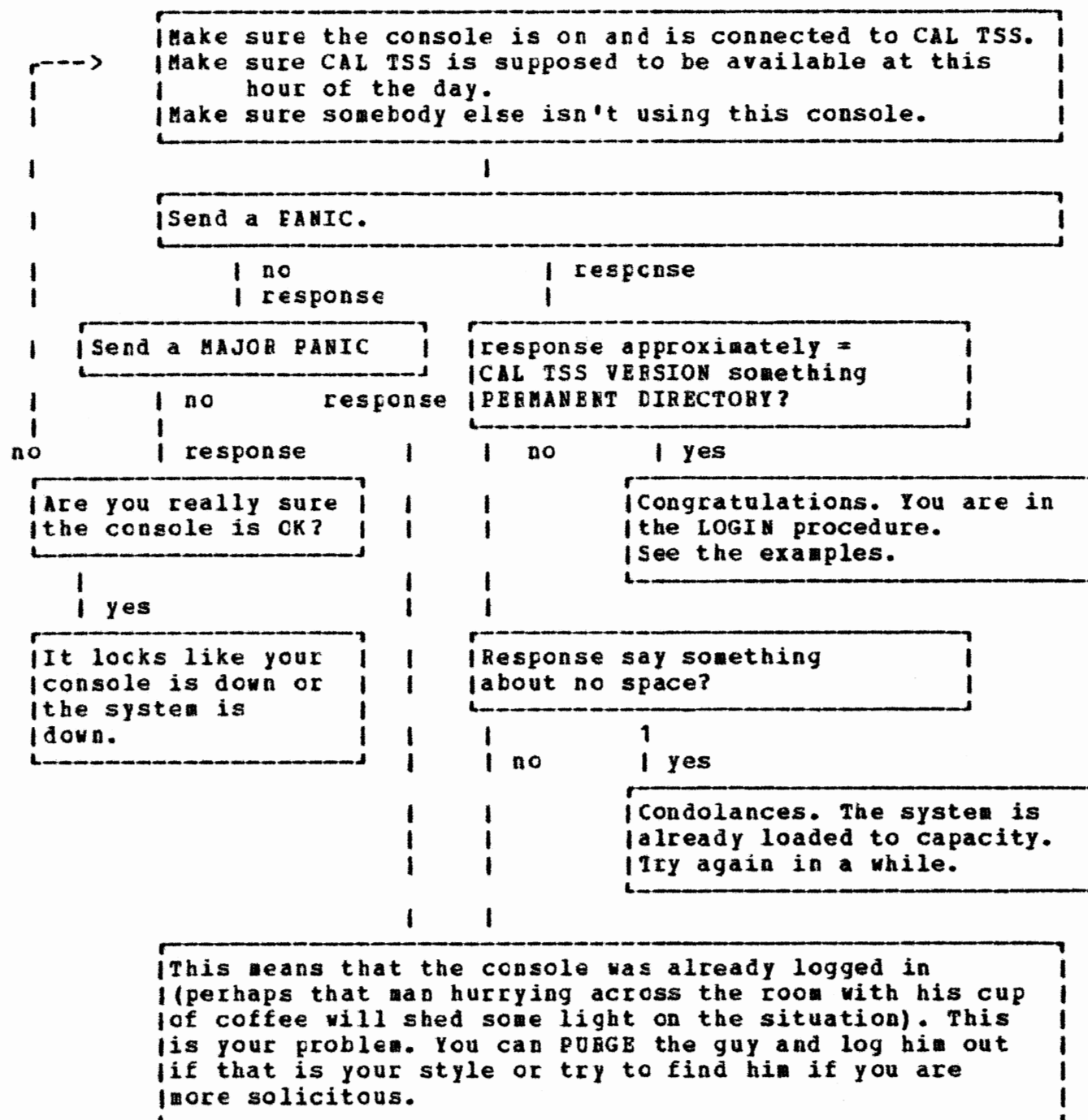
PROCEDURE II tells what the user does if he is already logged in and using the console but has either forgotten what he was doing or the console stopped responding the way he expects it to.

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.

July 1971

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



July 1971

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.)

|

no | If you have just typed something, did the characters echo  
r--< | (print)?

| yes |

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).

| no | response  
| response |

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

| no | response

July 1971

| 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

| 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 this way. Either  
| it was a non-standard subsystem,  
| or CAL TSS has a bug. |

July 1971

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

Send a PANIC
--------------

no response	response
----------------	----------

<p>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.</p>
---

Send a MAJOR PANIC
--------------------

no response	response
----------------	----------

<p>It looks like your your console or the system is down.</p>
---

<p>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.</p>
---

July 1971

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

SUBSYSTEM	PROMET	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 CMDS	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 suc- cessfully finish the login (see examples)
SERVICES	*	same as COMMAND PROCESSOR	'FIN'
BEAD GHST (debugger)	@	same as COMMAND PROCESSOR	'PURGE' will return to the COMMAND PRO- CESSOR; 'RETRY' or 'RETURN' will return to the currently active subsystem.
EDITOR	:	????	'P' or 'Q' (see EDITOR document)
BASIC	: or ?	???? or miscellaneous diagnostics relevant to erroneous BASIC statements	same as EDITOR
SCOPE	(see SCOPE)	??NO??	'FIN'



July 1971

## 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 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 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

<u>type</u>	<u>meaning</u>	<u>and the teletype responds</u>
<u>CTRL-I</u>	make an insert at the beginning of the old line	<
10_	this is what is to be inserted	10_
<u>CTRL-O</u>	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

<u>type</u>	<u>meaning</u>	<u>and the teletype responds</u>
<u>CTRL-D</u>	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
<u>CTRL-Q</u>	erase the M	<-
N		N
<u>CTRL-H</u>	copy the rest of the old line into the new line	T_X

July 1971

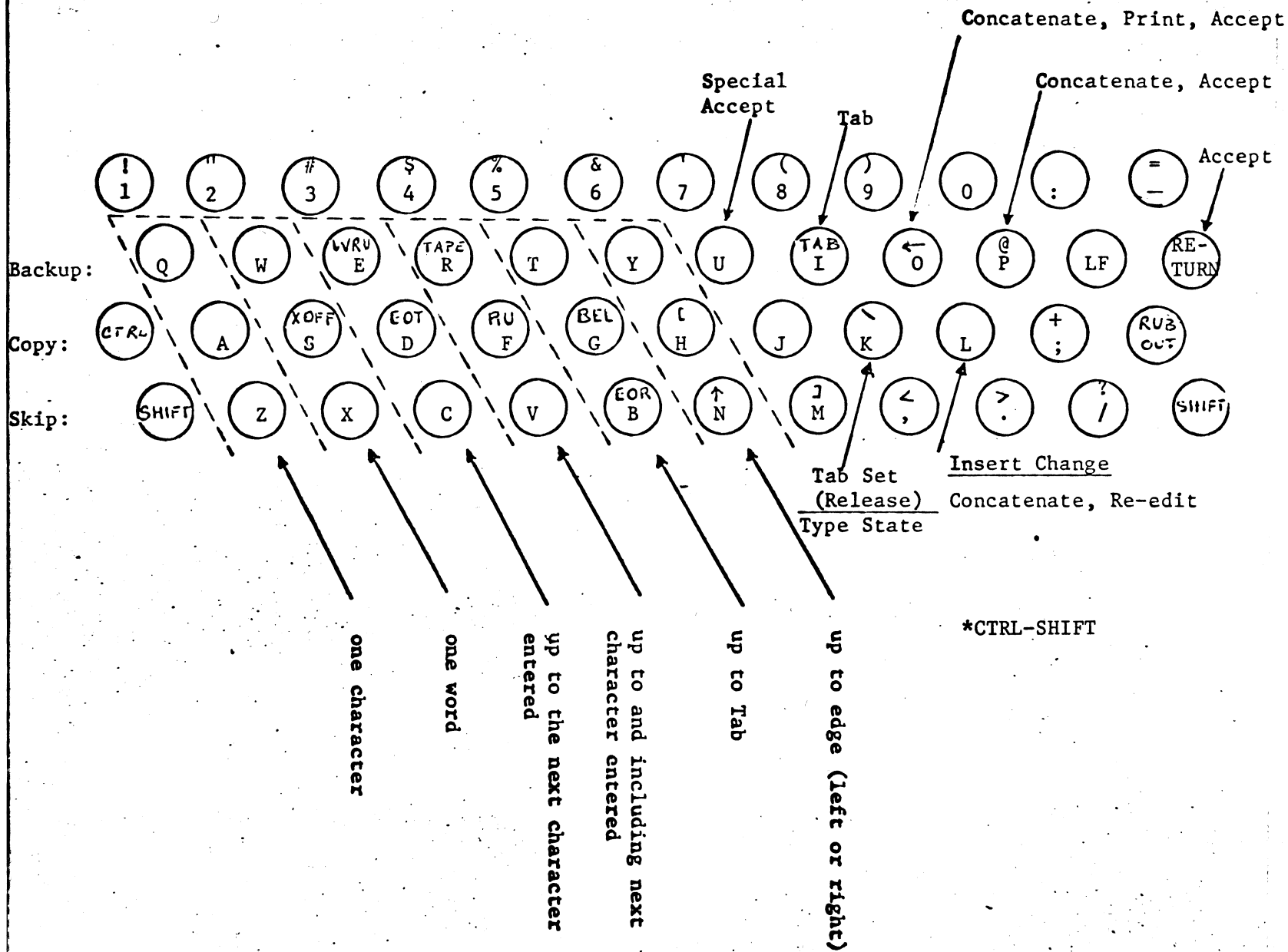
,Y      you remembered to print Y  
RETURN    you are satisfied with your  
         new line

,Y  
         and the car-  
         riage will  
         return

BASIC should accept this line, which is  
old line:    10 PRINT X,Y

JULY 1971

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



July 1971

## 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.

October 1971

## Example 2.1

CAL TSS VERSION 2.0

20:35:14 10/21/71

PERMANENT DIRECTORY?

.GUEST

GIVE PASSWORD

.GUEST

TEMPORARY DIRECTORY?

.JOHN

COMMAND PROCESSOR HERE

IBASIC

BASIC VERSION 2.0

-PRINT PI

3.141593

-10 LET X = 13

-20 LET Y = 19+3

-30 PRINT X,YX\*Y

ERROR OPERATOR MISSING

-30 PRINT X,Y,X\*Y

-40 END

-RUN

13

13

234

EXECUTION COMPLETE

-LIST 30

30 PRINT X,Y,X\*Y

-EDIT 30

30 PRINT X,Y,X\*Y,X/Y

-RUN

13

13

234

EXECUTION COMPLETE

-FIN

CHANGES NOT SAVED

-FIN

COMMAND PROCESSOR HERE

ILOGOUT

20:37:10 10/21/71

CONNECT TIME = 97782.

CPU TIME = 6311602.

FIXED ECS = 344681550.

NOT SLOTS = 0.

SWAPPED ECS = 407565312.

TEMP DISK = 0.

MONEY = \$.297

GOOD DAY

(1.1)

(1.2)

(1.3)

(2.1)

(3.1)

(3.2)

(3.4)

(3.5)

(3.6)

(3.7)

(3.8)

(3.9)

(3.10)

(3.11)

(4.1)

.7222222

July 1971

## 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.

July 1971

This page no longer contains information.

July 1971

## Example 2.2

```

(1.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .USER: VV
        GIVE PASS WORD
        .ORBL
        TEMPORARY DIRECTORY?
        .V
(2.0) { COMMAND PROCESSOR HERE
        !SERVICES
        SERVICES HERE
        *NEWDF PERMDIR: AUTO (3.1)
        *PCAP OWN.KEY
        7777777777777777002737 } (3.2)
(3.0) { 00000000000000000053002
        *ADDKEY 53002 7777777777777777 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
        10 PRINT 10*PI
        20 PRINT 20*PI
        30 END
(4.0) { : F
        COMMAND PROCESSOR HERE
        !EDITOR M
        : I
        10 LET X = 10
        20 LET Y = 20
        30 PRINT X*PI, Y*PI
        40 END
        : F
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```



July 1971

**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 NEWDP. He has asked that it be created in his permanent directory and named AUTO.
- 3.2 The command 'PCAP CWN.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 'NEWDP 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 2.0  
20:40:39 10/21/71  
PERMANENT DIRECTORY?  
.USER:VV  
GIVE PASSWORD  
.QRBL  
TEMPORARY DIRECTORY?  
.V

COMMAND PROCESSOR HERE

IRASIC

BASIC VERSION 2.0

-LOAD AUTO

ERROR OPERATOR MISSING

20 PRINT 20\*PI

-LIST

10 PRINT 10\*PI

30 END

-20 PRINT 20\*PI

-RUN

31.41593

62.83185

EXECUTION COMPLETE

-SAVE AUTO

-FIN

COMMAND PROCESSOR HERE

!LOGOUT

20:41:43 10/21/71

CONNECT TIME = 47156.

CPU TIME = 7245765.

FIXED ECS = 166224900.

NOT SLOTS = 0.

SWAPPED ECS = 224351232.

TEMP DISK = 0.

MONEY = \$.296

GOOD DAY

1.0

1.1

1.2

1.3

1.4

1.5

1.6

1.7

July 1971

## 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.

October 1971

## Example 2.3.2.1

CAL TSS VERSION 2.0  
 20:42:27 10/21/71  
 PERMANENT DIRECTORY?  
 .USER:VV  
 GIVE PASSWORD  
 .QRBL  
 TEMPORARY DIRECTORY?

.V

COMMAND PROCESSOR HERE  
 EDITOR MANUAL  
 :T;P\$

:Q

COMMAND PROCESSOR HERE  
 SERVICES  
 SERVICES HERE  
 \*MCAP PERMDIR:MANUAL TEMPDIR:M  
 \*FIN

COMMAND PROCESSOR HERE  
 IBASIC  
 BASIC VERSION 2.0

-LOAD M

-RUN

31.41593 62.83185

EXECUTION COMPLETE

-FIN

COMMAND PROCESSOR HERE  
 !LOGOUT

20:43:58 10/21/71

CONNECT TIME = 71783.

CPU TIME = 10849976.

FIXED ECS = 253038600.

NOT SLOTS = 0.

SWAPPED ECS = 319674880.

TEMP DISK = 0.

MONEY = \$.443

GOOD DAY

(2.1)

July 1971

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.

October 1971

## Example 2.3.2.2

```

CAL TSS VERSION 2.0
20:46:04 10/21/71
PERMANENT DIRECTORY?
.USER:VV
GIVE PASSWORD
.QRBL
TEMPORARY DIRECTORY?
.V
COMMAND PROCESSOR HERE
ISERVICES
SERVICES HERE
*CHAIN PERMDIR TEMPDIR
*UNCHAIN PERMDIR
*CHAIN TEMPDIR PERMDIR
*FIN
COMMAND PROCESSOR HERE
IBASIC
BASIC VERSION 2.0
-LOAD MANUAL
-RUN
  31.41593      62.83185
EXECUTION COMPLETE
-FIN
COMMAND PROCESSOR HERE
ILOGOUT
20:47:14 10/21/71
CONNECT TIME = 52511.
CPU TIME = 7699295.
FIXED ECS = 185104800.
NOT SLOTS = 0.
SWAPPED ECS = 247353344.
TEMP DISK = 0.
MONEY = $.317
GOOD DAY

```

(1.0)

(1.1)

(1.2)

(1.3)

(2.0)

July 1971

## 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.

July 1971

## Example 2.4

```

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
  SERVICES HERE
  *MCAP PERMDIR:TRIVIA TEMPDIR:INPUT
  *FIN

```

(1.0)



July 1971

```

(2.0) { COMMAND PROCESSOR HERE
        !EDITOR INPUT
        :T;P$
        (2.1)
        PROGRAM TRIV(TTYIN,TTYOUT,TAPE2=TTYIN,TAPE1=TTYOUT)
        WRITE (1,100)
        100  FORMAT (*TRIVIA SPEAKING, WHO'S THERE?*)
        READ (2,200) NAME
        200  FORMAT (A10)
        WRITE (1,300) NAME
        300  FORMAT (*GOODBYE,*A10)
        END
        :Q
        COMMAND PROCESSOR HERE
        !SCOPE 40000 (3.1)
        15:42:35 08/06/71 SCOP32C OF 08/01/71
        >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
        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

```

July 1971

## 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 accommodate 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 compiler 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.

July 1971

## Example 2.5

```

(1.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .GUEST
        GIVE PASS WORD
        .GUEST
        TEMPORARY DIRECTORY?
        .VANCE
        COMMAND PROCESSOR HERE
        !SERVICES
        SERVICES HERE
        *PCAP OWN.KEY
        77777777777777002737 } (1.1)
        00000000000000123401
        *FIN
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```

```

(2.0) { CAL TSS VERSION 1.2
        PERMANENT DIRECTORY?
        .USER:VV
        GIVE PASS WORD
        .ORBL
        TEMPORARY DIRECTORY?
        .VANCE
        COMMAND PROCESSOR HERE
        !SERVICES
        SERVICES HERE
        *ADDKEY 123401 71420 PERMDIR:REACT (2.1)
        *ADDKEY 123401 71420 PERMDIR:DATA
        *FIN
        COMMAND PROCESSOR HERE
        !LOGOUT
        GOOD DAY

```

July 1971

(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

UNEXPECTED FRETURN

\*FRIENDP USER:VV

BAD SYNTAX

\*FRIENDP USER:VV TEMPDIR:VV

BAD SYNTAX

\*FRIENDP USER:VV TEMPDIR:VV

\*MCAP VV:REACT;OWN.KEY PERMDIR:REACT

\*MCAP VV:DATA;OWN.KEY PERMDIR:DATA

\*ADDKEY 123401 77777777777777 PERMDIR:REACT

\*ADDKEY 123401 77777777777777 PERMDIR:DATA

\*FIN

COMMAND PROCESSOR HERE

!LOGOUT

GOOD DAY

(3.1)

(3.2)

(3.3)

(3.4)

July 1971

CAL TSS VERSION 1.2  
PERMANENT DIRECTORY?

.GUEST

GIVE PASS WORD

.GUEST

(4.0)

TEMPORARY DIRECTORY?

.VANCE

COMMAND PROCESSOR HERE

ISCOPE

(4.1)

16:19:54 08/06/71 SCOP32C OF 08/01/71

>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 THANK

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

↑6425 823

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

↑SUMMER

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

July 1971

FISHERMAN'S BOAT DRIFTS  
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

(4.3)

{>FIN  
COMMAND PROCESSOR HERE  
!LOGOUT  
GOOD DAY

July 1971

**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

(1.0) { 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  
 !LOGOUT  
 GOOD DAY

(2.0) { 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  
 PASS WORD NOT CONFIRMED (2.3)  
 PERMANENT DIRECTORY?  
 .USER:VV  
 GIVE PASS WORD  
 .QFBL  
 TEMPORARY DIRECTORY?  
 .PAUL  
 DUPLICATE TEMPDIR (2.4)  
 TEMPORARY DIRECTORY?  
 .VANCE  
 COMMAND PROCESSOR HERE  
 !LOGOUT  
 GOOD DAY



July 1971

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.

July 1971

### 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).
D <u>sc</u>	Delete the specified lines.
T	Move to the top of the file (pseudo-line).
M <u>sc</u>	Move forward over the specified lines.
B <u>sc</u>	Move backward the specified number of lines. (NOTE: <u>sc</u> can only be a number.)
P <u>sc</u>	Print the specified lines.
C/ <u>str1</u> / <u>str2</u> / <u>sc</u>	Replace the first occurrence of <u>str1</u> by <u>str2</u>

July 1971

CG/ <u>str1</u> / <u>str2</u> / <u>sc</u>	in the specified lines. Replace every occurrence of <u>str1</u> by <u>str2</u> in each of the specified lines.
<u>Esc</u>	<u>Edit</u> the specified lines using the Line Collector.*
R, <u>fname</u>	Insert the contents of the file <u>fname</u> after the current line.
W, <u>fname</u> , <u>sc</u>	Write the specified lines, including the current line, into the file <u>fname</u> .
F, <u>fname</u>	Finished - create the file <u>fname</u> 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 cld line in the line collection and may then be altered using the Line Collector. (See section 1.10 on the Line Collector.)

July 1971

### 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
```

July 1971

```

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<sup>5</sup> 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

---

<sup>5</sup> 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.

July 1971

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 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.

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.

July 1971

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.

July 1971

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.



July 1971

OperatorsArithmetic

↑ 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) $\sqrt{x}$
EXP(X) $e^x$	TAN(X) tan(x)
INT(X) integer	TIM(X) secnds used
LOG(X) $\log_{10} x$	

July 1971

### 3.3 Summary of the SCOPE Simulator

SCOPE provides an operating environment for 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 from 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

<u>Request</u>	<u>Meaning</u>
TEXT, <u>fname</u>	Declare a new SYSTEXT file <u>fname</u> (will not change to SYSTEXT a file which already exists in another mode).
FILE, <u>fname</u>	Use the file <u>fname</u> as the source of SCOPE Simulator requests.
MSG, OFF or ON	Suppresses program messages to the console or restarts them.
GET, <u>fname</u>	Get the file <u>fname</u> from the BEAD NAME SPACE.
PUT, <u>fname</u>	Return <u>fname</u> to 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 SCOPE Simulator.

#### Loading requests:

	<u>Meaning</u>
L, <u>fname</u>	Load and link the file <u>fname</u>
LGO, <u>fname</u>	Load and link the file <u>fname</u> and start the resulting code executing

July 1971

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

CALIDGSCOPE Control Requests:

CATALOGUE  
CCMFABE  
CCMPASS  
COPY  
COPYL  
COPYN  
COPYBSF  
CPC  
DMP  
REWIND  
RFL  
RUN  
SNOBCL  
UPDATE

Library Programs:

CFIO  
DEBUG  
IO  
IORANDOM  
KOMMON  
MEMORY  
REGDUMP  
SETPRU  
TRACE

All RUN FORTRAN Library Routines

July 1971

3.4 SERVICES and the BEAD GHOST

This section consists of a list of the commands 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 <u>password</u>	changes the user's password to <u>password</u>
NEWDF <u>direct:fname</u>	creates a file <u>fname</u> in the directory <u>direct</u>
ADDKEY <u>keynum</u> <u>obits</u> <u>dirloc</u>	adds a lock which can be opened by the access key <u>keynum</u> to directory entry <u>dirloc</u> ; the kinds of access allowed to wielder of <u>keynum</u> are defined by <u>obits</u>
DELKEY <u>keynum</u> <u>dirloc</u>	revokes privileges of access to the directory entry <u>dirloc</u> for holders of access key <u>keynum</u>
FRIENDP <u>direct</u> <u>objloc</u>	if there is a permanent directory named <u>direct</u> , access to it is placed in <u>objloc</u> ; the access is highly restricted
FRIENDT <u>direct</u> <u>objloc</u>	same as FRIENDP, except temporary directories
PCAP <u>object</u>	prints the indicated <u>object</u>
PDATA <u>datum</u>	prints the indicated <u>datum</u>
PDATA <u>datumloc</u> <u>datum</u>	prints <u>datum</u> words of data, starting at <u>datumloc</u>
MCAP <u>object</u> <u>objloc</u>	places a link to <u>object</u> at <u>objloc</u>
MDATA <u>datum</u> <u>datumloc</u>	moves <u>datum</u> to <u>datumloc</u>
CHAIN <u>direct1</u> <u>direct2</u>	makes <u>direct2</u> look like an extension of <u>direct1</u>
UNCHAIN <u>direct</u>	eliminates any extension of <u>direct</u>
NEWV <u>ident</u>	creates a new variable <u>ident</u>
KILLV <u>ident</u>	eliminates the variable <u>ident</u>
DLIST <u>direct</u>	prints the contents of the directory <u>direct</u>
SPACE <u>datum1</u> <u>datum2</u> <u>datum3</u> <u>datum4</u>	resources are reserved for the user; see section on space control
MSPACE <u>direct1</u> <u>datum</u> <u>direct2</u>	<u>datum</u> sectors of disk space are moved from

July 1971

<u>NEWU ident datum</u>	<u>direct1</u> to <u>direct2</u> . One must be the father of the other. (One sector=64 words) creates new user subordinate to the user (i.e., a new permanent directory named <u>ident</u> of size <u>datum</u> as a son of the user's permanent directory)
<u>KILLU ident</u>	the permanent directory <u>ident</u> is eliminated from the user's permanent directory and destroyed
<u>NEWDR ident datum</u>	creates a directory <u>ident</u> on the user's permanent directory of size <u>datum</u>
<u>NEWELK filadr</u>	creates a new file block at <u>filadr</u>
<u>KILLBLK filadr</u>	deletes the file block at <u>filadr</u>
<u>NEWKEY obilloc</u>	creates a new access key at <u>obilloc</u>
<u>KILLOBJ object</u>	deletes the indicated <u>object</u>
<u>DELLINK dirloc</u>	removes the link at <u>dirloc</u>
<u>DELCWN dirloc</u>	the ownership entry at <u>dirloc</u> 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.

July 1971

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
70B+(#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 temporary 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 specify 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 alphanumeric characters, blanks excluded

July 1971

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 VAR-NAME and move objects to it
PERMDIR:PNAME	A <u>dirloc</u> is a special form of <u>objloc</u>



SERVICES- HERE

\*BLIST TSS

NAME	TYP	OBJ	FILES	PC	RSVD	SPC	USED	SPC	DESC	SPC	RSVD\$	USED\$
OWN.KEY	HRD	AKY										
PERMDIR	HRD	DIR										
PASSWORD	HRD	AKY										
BILL	OWN	DIR			2364		2033		13		1536.000	1156.897
→ CODE	OWN	DIR			3270		3201				NO PRF	
CMMD.S	OWN	DIR			2430		2404				NO PRF	
→ ECS.S	OWN	DIR			3000		2376				NO PRF	
→ BIN	OWN	DIR			2645		2265				NO PRF	
XTEXT	OWN	DIR			434		344				NO PRF	
KEITH	OWN	DIR			3255		3065				512.000	426.230
STURGIS	OWN	DIR			3712		3454		44		1856.000	1570.705
DAVE	OWN	DIR			253		204				512.000	318.544
LDR.S	OWN	DIR			1606		1566				NO PRF	
GENE	OWN	DIR			5261		5074				2984.199	2885.238
DIRECTORY.S												
OWN	DIR				1640		1051				512.000	62.305
OPERATIONS.S												
OWN	DIR				200		154				NO PRF	
→ VV	OWN	DIR			3610		3303				1536.000	1161.450
PPU.S	OWN	DIR			764		757				NO PRF	
DISK.S	OWN	DIR			3110		2315				NO PRF	
BRUCE	OWN	DIR			3400		3237				1024.000	478.296
KARL	OWN	DIR			1000		405				512.000	95.665
WILLIESUE												
OWN	DIR				300		234				512.000	89.675

DIRECT	RSVRD	IN-USE	DSCNDTS	RSVD\$	USED\$
--------	-------	--------	---------	--------	--------

HERE	54367	54367	54202	14964.000	11648.685
SUBDIRS	54202	47042	57		

FILE SPACE=



## BLIST \*ROOTD: BLIST

NAME	TYP	OBJ	FILES	SPC	RSVDSPC	USEDSPC	DESCSPC	RSVD\$	USED\$
PERMDIR	HRD	DIR							
PDLIST	HRD	DIR							
USER	OWN	DIR			3440	3330	3230	45876.000	5529.040
OWN.KEY	HRD	AKY							
PASSWORD	HRD	AKY							
OLD	HRD	DIR							
TSS	OWN	DIR			54367	54367	54202	14964.000	11648.685
STAFF	OWN	DIR			46522	46522	46377	45984.000	45550.849
GUEST	OWN	DIR			11	11		37697.000	1773.130
OPERATOR	OWN	DIR						512.000	36.781
FOR	OWN	DIR			372	372	367	550.000	550.000
CS	OWN	DIR			5631	5630	5606	2430.000	2431.040
NUTS	OWN	DIR			226	226	225	200.000	200.000
ERL	OWN	DIR			764	764	762	300.000	300.000
CRDHE	OWN	DIR			764	764	762	400.000	400.000
ME	OWN	DIR			454	454	447	816.000	816.000
GSPP	OWN	DIR			144	144	142	400.000	400.000
BA	OWN	DIR			144	144	142	105.000	105.000
USFS	OWN	DIR			144	144	142	100.000	106.892
EECS	OWN	DIR			62	62	61	41.203	41.203
TE	OWN	DIR			754	440		1000.000	529.933

DIRECT	RSVRD	IN-USE	DSCNDTS	RSVD\$	USED\$
--------	-------	--------	---------	--------	--------

HERE	145051	141471	141371	*****	.000
SUBDIRS	141371	140744	137615		

## FILE SPACE=

\*DELOWN P:VV

\*BLIST P:BIN

NAME	TYP	OBJ	FILES	SPC	RSVDSPC	USEDSPC	DESCSPC	RSVD\$	USED\$
OPERCL	OWN	DFL			13				
DISKSYS	OWN	DFL			621				
LDR.BINARY									
	OWN	DFL			157				
CMMDBIN	OWN	DFL			303				
DIRBIN	OWN	DFL			125				
DIRIBIN	OWN	DFL			67				
DRCTCL	OWN	DFL			7				
ECSBIN	OWN	DFL			337				
BEADS	OWN	DFL			147				
CLNPROF	OWN	DFL			23				
TTYWBIN	OWN	DFL			1				

DIRECT	RSVRD	IN-USE	DSCNDTS	RSVD\$	USED\$
--------	-------	--------	---------	--------	--------

HERE	2645	2265			
SUBDIRS					



\*BLIST TSS:ECS.S

NAME TYP OBJ FILESPC RSVDSPC USEDSPC DESCSPC

USED\$

SYS PAR	OWN	DFL	7		
SUBPROC	OWN	DFL	307		
LOWCM	OWN	DFL	7		
SYSYMB	OWN	DFL	7		
INITL	OWN	DFL	73		
IPROC	OWN	DFL	47		
SYS ENT	OWN	DFL	57		
OPERAT	OWN	DFL	103		
EVENT	OWN	DFL	73		
CAPAB	OWN	DFL	37		
MISC	OWN	DFL	43		
SHCED	OWN	DFL	17		
ALLOC	OWN	DFL	247		
SWAP	OWN	DFL	53		
MAPS	OWN	DFL	111		
FILES	OWN	DFL	77		
PROCESS	OWN	DFL	103		
SYSFXT	OWN	DFL	7		
SYSERR	OWN	DFL	23		
ECSINIT	OWN	DFL	27		
INTINIT	OWN	DFL	63		
GENLINT	OWN	DFL	17		
MUXINT	OWN	DFL	47		
SDVCINT	OWN	DFL	27		
DSKINT	OWN	DFL	67		
DSPINT	OWN	DFL	57		
SCHED	OWN	DFL	33		
DEADST	OWN	DFL	13		

DIRECT RSVRD IN-USE DSCNDTS RSVD\$ USED\$

HERE 3000 2376  
SUBDIRS

FILE SPACE= 2376

\*DELOWN TSS:ECS.S

ERROR: CMMDS

000017, 000007, 0000000000000000 ERROR

ADDITIONAL INFORMATION FROM CMMDS IS

000000000000000000000006

000000000000000000000026

000000000000000000000037

000000000000000000000022

\*BLIST TSS:ECS.S



\*BLIST TSS:VV

NAME	TYP	OBJ	FILES	PC	RSVD	SPC	USED	SPC	DESC	SPC	RSVD	USED
OWN.KEY	HRD	AKY										
PERMDIR	HRD	DIR										
TSS	HRD	DIR										
PASSWORD	HRD	AKY										
P	HRD	DIR										
BINARY	OWN	DFL	347									
TEVCH1	OWN	DFL	37									
LAST	OWN	DFL	7									
CLIST	OWN	DFL	7									
SUBPROC	OWN	DFL	163									
TALOC1	OWN	DFL	43									
ISUBPS	OWN	DFL	27									
TCLST1	OWN	DFL	23									
TMISC1	OWN	DFL	13									
ALLOC	OWN	DFL	243									
TSUBP1	OWN	DFL	13									
TFILE1	OWN	DFL	17									
TMACROS	OWN	DFL	27									
ECSINIT	OWN	DFL	27									
ECSMAC	OWN	DFL	7									
TTIME1	OWN	DFL	13									
TCOMP1	OWN	DFL	17									
MAPS	OWN	DFL	123									
SWAP	OWN	DFL	43									
SYSENT	OWN	DFL	153									
PROCESS	OWN	DFL	127									
SNO	OWN	DFL	13									
ECSACT	OWN	DFL	17									
INITL	OWN	DFL	73									
PROCSYM	OWN	DFL	17									
ERRNUMS	OWN	DFL	27									
SUBP	OWN	DFL	307									
EVENT	OWN	DFL	67									
LOG	OWN	DFL	33									
ERRCODE	OWN	DFL	7									
TTT	OWN	DFL	7									
DUMPX	OWN	DFL	257									
XCHED	OWN	DFL	33									
CMMD2	OWN	DFL	57									
MISC	OWN	DFL	43									

DIRECT	RSVRD	IN-USE	DSCNDTS	RSVD	USED
--------	-------	--------	---------	------	------

HERE	3610	3303		1536.000	1161.450
SUBDIRS					

FILE SPACE= 3303



\*QLIST TSS:CODE

BEADS OWN DFL  
 USRDSK OWN DFL  
 FRETRW OWN DFL  
 TERMIN OWN DFL  
 HELPER OWN DFL  
 DSKLDR OWN DFL  
 DSKLDR1 OWN DFL  
 CMMD OWN DFL  
 CMMD1 OWN DFL  
 BEAD6 OWN DFL  
 CMMD2 OWN DFL  
 PPU OWN DFL  
 CPU OWN DFL  
 OPER1 OWN DFL  
 DISK1 OWN DFL  
 DISK2 OWN DFL  
 DISK3 OWN DFL  
 DISK4 OWN DFL  
 DIR1 OWN DFL  
 TTYW1 OWN DFL  
 JANITOR OWN DFL  
 USERN OWN DFL  
 TERMINN OWN DFL  
 HELPERN OWN DFL  
 RETRYN OWN DFL  
 TTYWTCH OWN DFL  
 JPROCN OWN DFL  
 ACCTP OWN DFL  
 RETRY OWN DFL  
 DIRSYS OWN DFL  
 TLINE OWN DFL  
 NULLSUB OWN DFL  
 REALUSERBUG  
     OWN DFL  
 USERBUG HRD DFL  
 NEWSYS OWN DFL  
 NAMES OWN DFL  
 MKNAMES OWN DFL  
 CLNPROF OWN DFL  
     FULL OWN DFL  
 ACCTPN OWN DFL  
 SYSTAPE OWN DFL  
 BUILDER OWN DFL  
     DISK5 OWN DFL  
     FAKE6 OWN DFL  
     OPER2 OWN DFL

Tentative CAL TSS consulting schedule, effective 20 Oct '71:

Monday	Keith Standiford
Tuesday	Bruce Lindsay
Wednesday	Dave Redell
Thursday	Vance Vaughan
Friday	Gene McDaniel

Time: 2-3PM

Place: 201 Evans Hall

Phone: (64)2-5817, to be changed to (64)2-5008 sometime soon

For the time being, I hope you will try to find out who you are talking to and how to contact him, as we don't know some of the people who are using the system. Please note each (coherent) contact with a customer in the 'Consultant's Log'. Who knows? We may identify some common problems that are correctable before the bookkeeping gets out of hand.

The only entertainment currently available on the system is ELIZA and REACT, both available in the directory USER:VV, password ~~CRST~~. Somebody is adapting his learning cubic program to the system, hopefully available soon.

AAARG



18 NOV 71 : 12:30 pm

CHRIS GRANGER CS1

problem of understanding how to  
use  $RND(x)$  in BASIC.  
introduced him to use of  $RND(-1)$  to  
cause REAL TIME clock to be used as  
random number generator seed. problem  
solved. Here



## CAL Time-Sharing System

Status and Information, 19 October 1971

### Schedule

CAL TSS will be run from 10:00 a.m. to 10:00 p.m. on weekdays. The hours from 12 to 6:00 will be reserved for customers. Customers who successfully login at other times may use the system at their own risk - they may be chased off on 5 minutes notice.

### Consulting

A member of the CAL TSS staff is available from 2:00 to 3:00 p.m. on weekdays to answer questions, demonstrate the system, help new users through initial sessions, etc. The consultant will temporarily be located in Room 201, Evans Hall, (64)2-5817; to be changed to (64)2-5008.

### Documentation, Information

The Introduction to CAL TSS, available from the Computer Center Library, hopefully provides a means of getting new users acquainted with CAL TSS. It also provides a good deal of information that may interest experienced users.

### Charging

Free use of CAL TSS will end effective 25 October 1971. Arrangements can be made now with the Computer Center Accounting Office; (64)2-7355, 237 Evans Hall to have TSS funds available on the 25th, thus avoiding any interruption in your service. Please note that even if you have a valid Computer Center job number, you must still file another form to make those funds available on TSS.

Details of the rate structure are appended.

### Teletypes

All teletypes connected to the system as of 18 September should be functioning as before. Report all line problems and problems with Computer Center teletypes to Gil Costa, (64)2-4775. Some details specifying the problem will be required.

### Access to Teletypes

The TTYs which were available for general use in 225 Campbell has disappeared in the machine shuffle. Some TTYs are being installed in B30 Evans Hall, but the use of these TTYs will be subject to some control not as yet specified. It will also be a week or more before these TTYs are connected at all. If this interferes with your access to the system, please contact the CAL TSS consultant and make the details of your problem known to him.

old

## CAL Time-Sharing System

Status and Information, 13 August 1971

### Availability

CAL TSS is currently available weekdays from 2-6 PM. There are 8 teletypes available for general use during these hours in Rooms 225 and <sup>223</sup> ~~223~~ Campbell Hall. For information about connection of additional teletypes, contact Vance Vaughan (see below).

### Documentation

The fundamental document for users is the Introduction to CAL TSS, available from the Computer Center librarian. Other documentation is also available at the library, but it is spotty and users should consult with someone on the TSS staff before acquiring any.

### TSS Consultant

A member of the TSS staff is available in Room 225 Campbell Hall, ext. 2-5008, from 2-3 PM every weekday except Wednesday. He will answer questions, demonstrate the system, help new users through initial sessions, etc. Users unable to reach the consultant should contact Vance Vaughan, 207 Evans Hall, ext. 2-5823. He is there Thursdays from 1-2 PM, or by accident, or by appointment. Leave a message in the main Computer Center office, 239 Evans Hall, ext. 2-0851 to arrange an appointment.

### Getting help, reporting problems, etc.

Sections 1.7 and 1.9 of the Introduction to CAL TSS give procedures and information for diagnosing and understanding problems encountered when using the system. If the user's teletype is dead, or has gone crazy, he should first consult those sections. They may solve the problem, or be irrelevant, or give some such helpful advice as 'call a system programmer' or 'the teletype is down or the system is down'. If they are irrelevant, or say to contact a system programmer, or something like that, contact the TSS consultant (not the regular programming consultant). When the diagnosis is that the teletype is down or the system is down, the user should call the shift supervisor, (64)2-3043, and explain the problem. If the system is down, he will give information about when it will be up. If the system is up, there is some problem with the teletype or the line. The user should contact the person responsible for the maintenance of the teletype (Computer Center teletypes are maintained by Charles Cuffel, ext. 2-4403).

### Complaints and suggestions:

These should be made to the TSS consultant. The TSS staff is especially anxious to get feedback on the documentation. Corrections to content and suggested style modifications are both welcome.



How to find the teletype watcher in a PM dump.

This involves finding the special event channel that the watcher hangs on and looking at its queueing word. The MOT of the watcher doesn't change unless initialization changes, so noting the most recent MOT will short-cut this procedure in most cases.

1. Find the MOT and print the first few cells. The beginning of the MOT is given by the symbol EC.MOT, defined in the deck ECSINIT in the ECS system. It hardly ever changes. Currently = 411B
2. Two items are of interest: unique name 2 is the master C-list  
unique name 3 is the interrupt object file.  
Print the address specified for the interrupt object file. The very first word contains a pointer to the data block, usually just a few cells down from the file header itself. Print the first word of the data block. This is the index in the master c-list of the event channel where the watcher hangs.
3. Take the address of the master c-list from the MOT, add 1 and add the index of the watcher's event channel twice to find the address of the ~~xxxxxxcapabilityxxx~~  
~~Print the two words of the capabilityxxx. The second word is the MOT index of the~~  
~~evch; print itxxx~~ watchers event channel stuff. Print 3 capabilities (6 words) starting at that address. You should find a file followed by two evchs. The ~~first~~ *second* evch is where the watcher hangs. Print its MOT slot.
4. Use the MOT pointer you just found to print the first two words of the ~~evch~~. The second word is the process queueing word. If it starts with 1776, the watcher wasn't hanging when the dump was taken, sorry. Otherwise, the low 6 digits of the word are the MOT of the watcher. Congratulations.
5. A second page on how to find the user processes, given the ~~xx~~ location of the watcher, is forthcoming. That may be even more fun.

current origin of MOT

current MOT of watcher

origin      date

MOT          date

411<sub>8</sub>      280d'71

28 Oct 171

CP doesn't handle stack full properly.