GE-625/635
GECOS-III
Time-Sharing System

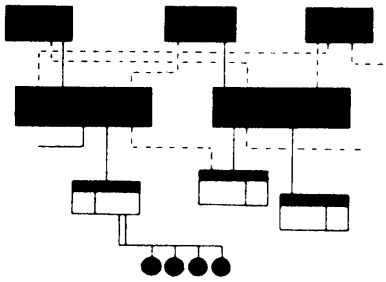
Betty J. Colaluca

Information Systems

Information Systems Equipment

TIS 600-213;

PROGRAMMING REFERENCE MANUAL



GENERAL ELECTRIC

CPB-1514

CPB-1514

GE-625/635 GECOS-III Time-Sharing System

PROGRAMMING REFERENCE MANUAL

PROGRAM NUMBER

CD600T1.001

March 1968

INFORMATION SYSTEMS

GENERAL ELECTRIC

PREFACE

The Programming Reference Manual provides methods for the experienced programmer to further develop and extend the time-sharing capabilities of the system via subsystems.

The text material includes subsystem organization, information required to program for the 600TSS, and how to place a program in the System.

Individual chapters are devoted to command languages and primitives, file formats, and the subsystems supplied by the General Electric Company.

This manual is one in a series of time-sharing manuals. The others are:

GE-625/635 GECOS III Time-Sharing System BASIC Language, CPB-1510

GE-625/635 GECOS III Time-Sharing System Text Editor, CPB-1515

This manual was produced using the General Electric Remote Access Editing System (RAES). RAES is a time-shared disc-resident storage and retrieval system with text-editing and manuscript formatting capabilities. The contents of the manual were entered into RAES from a remote terminal keyboard, edited using the system editing language, and formatted by RAES on reproduction masters.

The index was produced using a computer-assisted remote access indexing system. This system produces an index using source strings delimited at manuscript input time.

Suggestions and criticisms relative to form, content, purpose, or use of this manual are invited. Comments may be sent on the Document Review Sheet in the back of this manual or may be addressed directly to Documentation, B-90, Processor Equipment Department, General Electric Company, 13430 North Black Canyon Highway, Phoenix, Arizona 85029.

© 1968 by General Electric Company

(5M 6-68)

CPB-1514

CONTENTS

1. INTRODUCTION

The GE-600 Time-Sharing System (600TSS) is a feature of an integrated batch/remote batch/time-sharing system (GECOS III). The time sharing portion of GECOS III is organized as a privileged slave program and has a dynamically variable but contiguous block of memory allocated to it. Thus, the time-sharing function can be carried on in conjunction with the normal batch load. The Time-Sharing System does not occupy core if it is not in use; it does take a variable amount of core if the system is being used. The primary design objectives and features of the systems follow.

SIMULTANEOUS BATCH AND TIME-SHARING

This feature allows the user to develop time-sharing applications without dedicating a complete GE-600 (or other system) computer to this function. In many cases the initial time-sharing load is small and would not justify committing a large system solely to this function.

EASE OF EXTENSION BY USER

As in the batch system, it is necessary that the user be allowed to write programs for his unique applications. The 600 TSS is designed as an executive, or monitor, servicing generic subsystems. The subsystems are analogous to slave programs in the batch environment. BASIC and EDITOR are two of the most notable subsystems within 600TSS. It is expected that users will add capabilities via new subsystems to suit their local installation requirements. The 600TSS has been designed to minimize the effort required to generate and install subsystems.

2. SUBSYSTEM ORGANIZATION

A subsystem program consists of two logically and physically separate parts:

1. Program
2. Program descriptor

The program is the block of code to be executed. Its organization is similar to that of a batch-environment slave program. It can be written in any language whose object code is loadable by GELOAD at system editor time. As many subroutines as desired may be used, as well as SYMREF,SYMDEF, BLOCK statements, and library subroutines. The subsystem executes within the 600TSS core area with the base register set around the code as loaded so the subsystem is unaware of its relative position in core at anytime. The following restrictions apply:

1. A data area of 100_{10} words must precede the subsystem coding. TSS uses this for bookkeeping during program swap, register storage, and other items.

2. The derail (DRL) instruction is used to request 600TSS service functions, which are analogous to the MME functions provided for the batch environment. The DRL instruction may only be used to communicate with the 600TSS Executive. No MME instructions are permitted.

The program descriptor consists of several blocks of data that must be incorporated into the executive portion of 600TSS. This data serves two purposes:

1. Defines the name of the subsystem, its size, etc.

2. Defines command language and primitives.

This descriptor is generated by adding GMAP statements to a block (.TPRGD) within the communication region of 600TSS. The general layout of the program descriptor is shown in Figure 2 (Chapter 4).

The program descriptor is itself physically separated into two portions: (1) the program-descriptor proper, or primary portion, and (2) the command-language/primitive list. Both portions are assembled into the .TPRGD block.

The primary portion of the program descriptor is a block of data needed by 600TSS to identify the subsystem, determine its size and file location, and to locate its command-language/primitive list. The only part of this required from the subsystem designer is the subsystem name (in ASCII lower-case alphanumerics), a pointer to the command-language/primitive list for the subsystem, and the number of command language words (which may be zero).

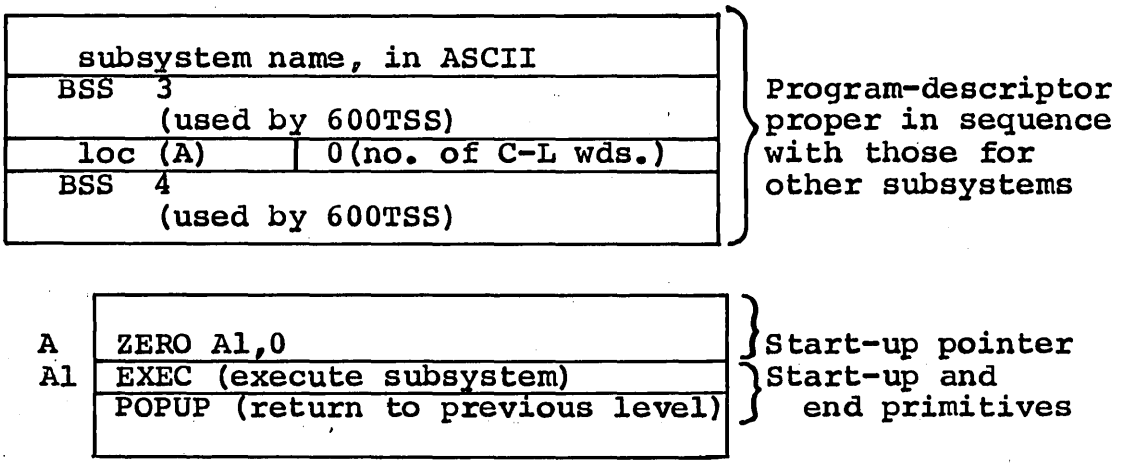
This primary portion of the program descriptor is fixed in length, and must be placed in sequence with the descriptors for other subsystems (at the beginning of block .TPRGD).

One or more command-language words for a given subsystem may be recognized by the system. The command-language list for the subsystem is pointed to by the program descriptor proper. This list consists of the text of each command-language word, an associated mask word, and a pointer to the corresponding primitive(s) for each. The command-language list always ends with a pointer to the "start-up" primitive(s). The list may be null, i.e., for a subsystem with no build-mode command language. Such a "degenerate" command-language/primitive list would consist, therefore, of only the start-up pointer and the corresponding primitive(s) for the start-up procedure.

The start-up procedure allows the subsystem designer either to specify a direct execution of the subsystem selected by a user, or, optionally, to select other subsystems or perform other preliminary operations before loading and executing the selected subsystem. For example, the BASIC subsystem's start-up procedure initially calls another subsystem (OLDN) to assign a source file and provide the OLD-NEW file-request sequence. The second start-up primitive then places the system in build-mode, to accumulate terminal input until a command-language word is recognized.

The actual flow of control to or from a subsystem is directed by the primitives that are specified for each command-language word in a subsystem, and for start-up. There are nine individual one-word primitives defined by the system; they are described in detail in Chapter 4, "Command Language and Primitives". Primitives may be considered as a high-level instruction sequence that direct 600TSS in the execution of sequences of subsystems, either when command language is encountered or when a subsystem is initially selected. The primary use of the primitives is to logically combine several related subsystems into a larger system, thus simplifying the system for the ultimate user at the terminal. As in BASIC or EDITOR, this logical combining of subsystems is apparent to the user.

For the simple case of a subsystem with no command language and a direct execute at start-up, the program descriptor layout would be as follows:



Note that the primitives need not be contiguous with the command-language list, as they are shown (the start-up pointer constituting the null list), since they are pointed to by the latter. However, both must be contained in the .TPRGD block of the TSS Communication Region.

The program-descriptor layout for the more complex situation of a subsystem with command language is described in Chapter 4.

3. PROGRAMMING FOR 600TSS

Writing a subsystem program for 600TSS is not significantly different than programming in slave mode for the batch system. The primary difference is that all MME functions provided by GECOS are eliminated, and a similar set of functions provided for time-sharing needs are supplied by TSS Executive, via the derail (DRL) instruction.

BASE REGISTER PROTECTION

The subsystem program, while in execution, has the base register set by 600TSS such that the subsystem cannot reference any memory area not assigned to it. The base register is always set to the user's current origin in memory, so that he is not aware of any changes in absolute memory area due to program swapping.

SUBSYSTEM DATA AREA AND FAULT VECTOR

The Time-Sharing system requires a data area of 100_{10} words in each subsystem for bookkeeping. This area must be at the beginning of the program when execution begins, although it may be moved during execution as explained in the service functions "Release Memory" and "Add Memory", in this chapter. The loader normally reserves 64_{10} words at the beginning of a program and this area is usable for a portion of the first 100_{10} words required. Thus, a subsystem program would normally start with a BSS 36 in order to reserve the additional space required.

The first 14 words of this data area represent the fault vector and are defined as for a GECOS slave, but the last four words may not be used. A subsystem fault vector, therefore, comprises words 0 to 11 of the data area. These words are used in pairs, one pair for each type of fault which can be returned to the subsystem program:

Words	0,1	Illegal op-code fault Zero-op-code fault Command fault Connect fault
	2,3	Memory fault
	4,5	Fault tag
	6,7	Divide check
	10,11 (octal)	Overflow fault

Format of each word-pair:

First word	C(IC)	C(IR)
Second word	TRA Instruction	

If a subsystem program causes one of the defined optional-recovery faults, 600TSS checks the second word of the corresponding fault-vector pair. If the second word is not zero, 600TSS stores the IC and indicators in the first word of the pair. If the second word is zero, 600TSS aborts the subsystem program giving a message to the terminal. For example, if a subsystem program wishes to recover from divide check and overflow faults the vector would be set up as follows:

0 1		No recovery
2 3		No recovery
4 5		No recovery
6 7	TRA DVCK	Transfer to recovery routine
8 9	TRA OVFLO	Transfer to recovery routine

The transfers, if any, must be stored in the fault-vector by coding within the subsystem. They cannot be loaded. The rest of the 100_{10} words in the subsystem data area are reserved exclusively for 600TSS usage.

SUBSYSTEM SWITCH WORD

A switch word is provided for each user to maintain status or pass information from one subsystem to another during execution. The 36 bit settings may be modified and/or tested by subsystem programs using the derails SETSWH (8) and RSTSWH (9). The switch settings may also be set, reset, and tested by the primitives defining the flow of control of the user-selected system. This joint accessibility of the switch word provides the time-sharing subsystem designer with the capability of having conditional execution of primitives based on conditions set by a subsystem program. To avoid interference with user-generated systems, those systems issued by GE are restricted to bits 0-17, and users are expected to restrict the additional usage to bits 18-35.

Present usage is:

0-14	not defined	
15	OLD - NEW sequence	(BASIC)
16	EDITOR	
17	Time-Sharing System	0 = no data on user's SY** file 1 = valid data on user's SY** file

SYSTEM MACROS

A set of TSS macros are available to facilitate the writing of 600TSS modules and subsystems. Two of these that would be of use to the implementor of a normal subsystem are .SSDRL and PRNTTY. The .SSDRL macro provides the symbolic address-value equivalences for the service-function DRL instructions (see the next subsection); PRNTTY provides a convenient facility for issuing short messages to the terminal.

The TSS macro library is loaded, at assembly time, by the call LODM. A description of the function of and calling sequence for each macro is described in Appendix A, "System Macros".

SERVICE FUNCTIONS

A subsystem program may request any one of many available services from 600TSS Executive. Since it is prevented from referencing any memory area outside that allocated and protected by the base register, this request must be made by an intentional fault. The DRL functions and their associated addresses follow:

Name	Function	Address octal	Address decimal
DIO	Do I/O on User's File	1	1
KOUT	Keyboard Output	2	2
KOUTN	Keyboard Output Then Input	3	3
KIN	Keyboard Input Last Line	4	4
RETURN	Return to Primitive List	5	5
DEFIL	Define and Access Temporary File	6	6
ABORT	Abort	7	7
SETSWH	Set Switch Word	10	8
RSTSWH	Reset Switch Word	11	9
REW	Rewind a Linked File	12	10
FILSP	Space a Linked File	13	11
RETFIL	Return a File	14	12
RELMEM	Release Memory	15	13
ADDMEM	Add Memory	16	14
CORFIL	Data From/To Core File	17	15
SNUMB	Obtain SNUMB	20	16
TIME	Obtain Processor Time and Time of Day	21	17
PASAFT	Pass List of Files to Subsystem	22	18
TERMTP	Terminal Type	23	19
PDIO	*Do I/O On a System File	24	20
SPAWN	Pass Job to Batch Processor	26	22
CALLSS	Internal Call to Another Subsystem	30	24
USERID	*Pass User-ID and Priority to Executive	31	25
TERMPG	*Clean Up UST After User Termination	32	26
PASUST	Pass UST to Subsystem	33	27
MORLNK	Add Links to Temporary File	34	28
NEWUSR	*Log-on New User Without Disconnect	35	29
FILACT	Permanent File Requests	36	30
SYSRET	Return to System	40	32
STPSYS	*Stop Execution of Master Subsystem	41	33

*This DRL function is privileged, or otherwise restricted, and is protected against execution by a normal subsystem.

Table 1. DRL Service Functions

Upon return to the subsystem all registers not specifically modified by the DRL function are restored to their original value and execution resumes at the first location after the DRL or the calling sequence parameters.

A detailed description of the non-file-I/O service function follows. However, those marked as restricted, in Table 1, are not described here. The file-I/O service functions are described in detail in Chapter 3 "File-I/O Derails". (A 600TSS system macro, .SSDRL, is available to provide the equivalence between the service-function name and the corresponding DRL address value. See Appendix A, "System Macros".)

Keyboard Output

```
DRL     KOUT    [2]
ZERO    L(tally), L(char)
```

The field (L(tally)) points to a driver tally word pointing in turn to a list of line TALLYB words which define each line of output of ASCII characters to be sent to the keyboard. The driver tally has the count of the line tallies in the list. This procedure allows the user to define scattered lines, not necessarily starting at word boundaries.

It should be noted that the Derail Processor utilizes the tally words and that they are modified on return to the subsystem.

The optional field L(char) points to a word containing up to four characters that will be appended to the end of the output defined by each line tally. These characters could be line feed, carriage return, etc. If this field is not present in the calling sequence, characters are not added. If the field is present, the first character of zero terminates the appending of characters. In any case, no more than four characters will be appended

The method used is to accumulate the user's output in a buffer for eventual output to the keyboard. When the buffer is full, output to the keyboard will be initiated. At this point, execution of the subsystem will be inhibited and the subsystem made eligible for swap. When the output is complete, the program is eligible for execution again.

Example:

```
      DRL     KOUT
      ZERO    DTAL,ENDC
       .
       .
       .
       .
DTAL  TALLY   LTAL,2
       .
       .
LTAL  TALLYB  LINE1,15
      TALLYB  LINE2,20
       .
       .
       .
LINE1  ASCII  4, THIS IS LINE 1
LINE2  ASCII  5,THIS IS SECOND LINE
       .
       .
ENDC   OCT    012015177000
STAT   BSS    1
```

This sequence will print two lines with three characters -- line feed, carriage return, and rubout -- appended to the end of each line.

Keyboard Output Then Input

```
      DRL     KOUTN      [3]
      ZERO    L(tally),L(char)
```

This sequence sends output to the keyboard device with an anticipated reply. The L(tally) and L(char) fields are identical to those for the KOUT sequence. In this case, however, the Executive adds this output message to any data that has accumulated in the keyboard output buffer and sends the data directly to the keyboard device. The transfer of data differs from the case of output only in that the line is left open for a response. The response can be retrieved by means of DRL KIN, described next.

Keyboard Input Last Line

```
DRL    KIN [4]
ZERO   L(dat),L(count)
ZERO   L(status)
```

This sequence retrieves the last line of input. Normally this sequence would follow immediately the KOUTN sequence; however, this is not necessary, and the DRL KIN may be repeated to retrieve the same line of input as many times as desired. The last line will remain in the buffer until some output or additional input destroys it. Dat is the location at which the string of input characters is to be stored. Count is a word in which the count of characters moved will be stored, in the lower part of the word. A zero character count will be returned if there is no data in the input buffer. The parameter L(status) is provided for future capabilities. The purpose of the status word is to receive the status of the line when it is passed back to the subsystem for certain conditions, such as BREAK, DISCONNECT, etc. In the present implementation the handling of these conditions is done by the executive and the status word in the subsystem is not altered.

Return to Primitive List

```
DRL     RETURN     [5]
```

This sequence indicates to the Executive that this subsystem process has reached a normal termination. The Control module selects the next primitive in the sequence defined within the program descriptor and, based on this primitive, initiates the next process. (Refer to the description of primitives in Chapter 4.)

Abort

```
DRL     ABORT     [7]
```

This sequence indicates to Executive that an abnormal event has occurred in a process. If the user has previously defined a file with the name ABRT, a core dump of the subsystem will be written to that file. In any case, a message is sent to the terminal stating that an abort has occurred and the user is free to select a new system.

Set Switch Word

DRL SETSWH [8]

A 36-bit switch word is provided for each user. This derail provides a method of setting individual bits of this word. The value of the Q register is ORed into the switch word. Thus, any bit that is on or true is set true in the switch word. Other bits are not disturbed. The value returned in the Q register is the resultant setting of the switch word. This provides a method of reading the switch word and determining a course of logic based on events in preceding subsystem processes. Thus, if one subsystem process encounters an abnormal situation, a prearranged bit may be set and subsequent subsystems may interrogate the switch word and take appropriate action. To obtain some measure of discipline in the use of this feature, the first 18 bits (0-17) are reserved for use of subsystems issued with the Time-Sharing System. The lower 18 bits (18-35) are kept free for user generated subsystems.

Reset Switch Word

DRL RSTSWH [9]

This function is similar to the SET SWITCH WORD except that each bit in the Q register that is on will be turned off in the switch word. The resultant value will be returned in the Q register.

Release Memory

DRL RELMEM [13]

C(A)	return location	data area location
C(Q)	no. words low	no. words high

This derail reduces the amount of memory assigned to a subsystem program during execution. The number of words to be released from the lower portion of the subsystem is in the left-half of the Q register and the number of words to be released from the upper portion of the subsystem is in the right-half of the Q register. Memory is released only in blocks of 1024 words. If the number of words specified (either high or low) is not a multiple of 1024, the number will be rounded. The address at which execution is to be resumed is in the left-half of the A register and the new location of the subsystem data area is in the right-half of the A register. This last entry must be modulo 8 and both addresses will be taken relative to the new base. The data area location must be given, even if it is not to change. (This derail may be used to change the data-area location only, without releasing any memory. This is the block of 100 words initially starting at subsystem zero.)

Add Memory

DRL ADDMEM [14]

C(A)	return location	data area location
C(Q)	----------	no. words high

This derail is the same as RELMEM except that the value in Q is interpreted as a request for additional memory at the upper boundary. A subsystem may not obtain memory at the low boundary. The number of words specified must be modulo 1024.

Data from/to Core File

DRL CORFIL [15]

C(A)	data loc.	n
C(Q)	i	k

A short block of core storage, called the core file, is maintained for each user. It allows one subsystem to pass data to another without accessing a mass storage device. This block of core is 10 words in length and may be "written" or "read" by a subsystem using the CORFIL derail.

The left-half of A contains the location within the subsystem that the data is to be read into or written from. The right-half of A contains the number of words (n) to be transmitted. The value of n must be $\leq$ 10. The left-half of Q contains the number of the core-file cell (i) at which transmission begins. The core-file cells are numbered 1 through 10. The right-half of Q(k) indicates the type of operation desired, i.e., "read" or "write":

k = 0 - transfer data from subsystem to core file
k = 1 - transfer data from core file to subsystem

Obtain SNUMB

```
DRL     SNUMB     [16]
```

Where a subsystem wishes to spawn a batch job, a unique SNUMB is required. The Time-Sharing Executive issues one of a sequence of numbers from 1 to 7777_8 and returns it in the following format:

C(A)

n	n	n	n	T	∅

(in BCI)

The trailing character T in the BCI form serves to distinguish this job and its output as a Time-Sharing initiated job. The subsystem program uses this number to generate the input file for the batch job, using this same number when the request is made to pass the file to the batch processor, i.e., DRL SPAWN (possibly notifying the user of the assigned SNUMB).

Obtain Processor Time and Time of Day

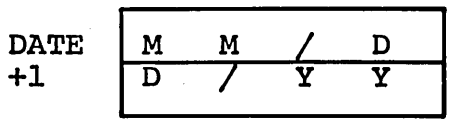
```
DRL     TIME     [17]
ZERO    L(date)
```

Returning from this derail, the processor time used by the current user, the time of day, and (optionally) the date are returned, in the following form:

C(A)	processor time
C(Q)	time of day

The unit of time is 1/64 of a millisecond.

At location date, the date is placed, in ASCII, with slants inserted between the values in the following form:



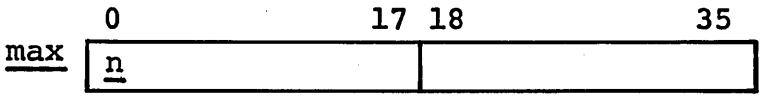
where MM is the month
DD is the day
YY is the year

If the value of L(date) is zero, the date is not stored.

Pass List of Files to User

```
DRL     PASAFT      [18]

ZERO    L(table),L(max)
```

where



n = maximum number of file names to be passed (if 0, all file names pass to the requestor)

This sequence places either the first n, or all, of the user's file names in the area specified. The format of the table passed back, at location table is:

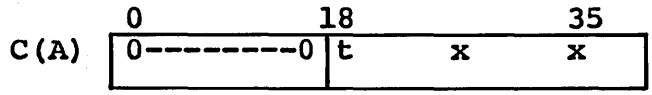
Word 1	No. of active files
2	file 1 - char 1-4
3	file 1 - char 5-8
	•
	•
N*2	file n - char 1-4
N*2+1	file n - char 5-8

Note: If the address value L(max) is equal to zero, all file names will be passed (as well as if n equals zero).

Terminal Type and Line Number

```
DRL     TERMTP    [19]
```

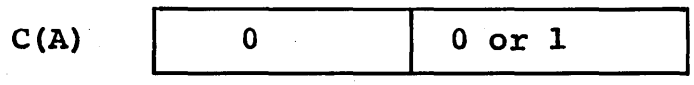
For some situations, it is necessary that the subsystem be aware of the type of terminal that is connected. It would be desirable to assume that all terminals could be made to look the same by use of the ASCII character set. While this is generally true, there are some essential features available on terminals that would be utilized by different procedures. This derail will return the line number (xx), in BCI, in the right-most 12 bits of the A register and the terminal type code (t) in bit positions 18-23. The codes for terminal type will be the same as defined by the GERTS system.*



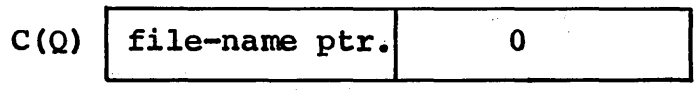
Code	Terminal Type
1	(not presently defined for TSS)
2	(not presently defined for TSS)
3	(not presently defined for TSS)
4	Teletype

Pass Job to Batch Processor

```
DRL     SPAWN     [22]
ZERO    L(SNUMB),L(buffer)
```



0 = immediate return
1 = return after batch job completes



The subsystem writes the file as an input job deck on a linked file and QU points to the name. The file must have an empty 320-word first block. The input deck (beginning with the second block) must have SNUMB and ENDJOB as in a normal card-image input deck, and must end with a GEFRC end-of-file (control record of zero). The file is in BCI format.

---

*See GERTS Manuals CPB-1416, 1417, 1418

The SNUMB-pointer specifies the location of the SNUMB obtained via DRL SNUMB. The buffer-pointer points to a 325-word work area provided by the subsystem. SPAWN generates the information required by GEIN in this area and writes it into the 320-word first block of the file.

Upon return to the subsystem, any error condition is indicated in the Q-register. The one-digit error code, right-justified in QU, is as follows:

0 - no error
1 - undefined file
2 - system loaded
3 - duplicate SNUMB
4 - SNUMB not given

Internal Call to Another Subsystem

```
DRL       CALSS       [24]
ASCII     1,NAME
```

The current (calling) subsystem is swapped out to the swap file, to be resumed later when a POPUP primitive of the called subsystem is executed. Internal calls may not be more than three deep, i.e., nesting to more than two levels is not allowed.

Pass UST to Subsystem

```
DRL   PASUST      [27]
ZERO  L(buffer)
```

This derail copies the user's status table (UST), maintained within the TSS Executive, to the buffer provided by the subsystem. The TSS Communication-Region equivalence .LLNUE defines the size of the UST. The user must be aware of the format and content of the UST as currently defined by 600TSS. This DRL should be used carefully since this UST definition may change occasionally.

Return to System

```
DRL   SYSRET      [32]
```

This derail causes the subsystem to be killed; control returns to the system-selection point, thereby bypassing the normal return via the primitives.

FILE I/O DERAILS

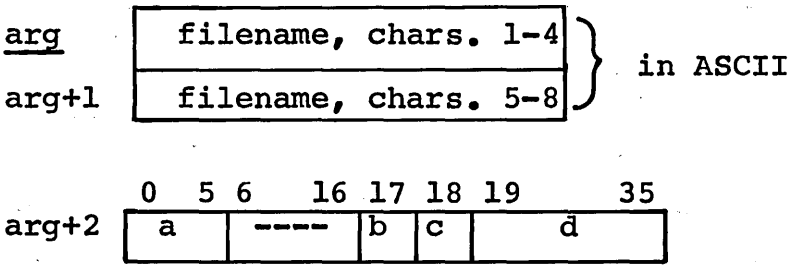
Time-Sharing subsystem programs perform disc-or-drum file activities using the following derail service functions:

1. Define and access a temporary file
2. Return a file
3. Space a linked file
4. Rewind a linked file
5. Do I/O on a user's file
6. Add more links to a temporary file
7. Permanent file activities

These functions are similar to and replace the MME functions in the GECOS III batch system.

Define and Access a Temporary File

```
DRL     DEFIL      [6]
ZERO    L(arg),L(stat)
```



where

a = device type as follows --
- 00 - DSU 270 (Disc)
- 01 - DSU 200 (Disc)
- 03 - MDU 200 (UNIVAC Drum)
- 04 - MDU 300 (Fairchild Drum)

b = 0 - use the standard TSS temporary-file device
b = 1 - use the type of device defined in a

c = 0 for linked file
c = 1 for random file

d = number of links desired, in binary

stat (2 words) -- status return in binary, right-justified in word 1:

0 = successful
3 = no room in AFT
4 = temporary file not available
5 = duplicate file name
6 = no room in PAT

A temporary file can be requested either on a specific type of device or on the standard TSS system device for temporary files (bit 17, arg + 2).

If a specific device type is requested but there is not enough space on that device, the request will be satisfied from the standard device.

After the file space is obtained, the DEFIL function builds the PAT entry and enters the file in the user's AFT.

Upon a successful return from this function, the A-register contains the following information:

Bits	Meaning
0-5	Device type
6-17	Number of words per physical block

- if bit 18 = 0, then

24-35	Number of links in the file

- if bit 18 = 1, then

24-35	Number of blocks in the file

- if bit 19 = 0, linked file

- if bit 19 = 1, random file

20-23	Unused

This information will also be returned for an already-defined file (status = 5).

Return a File

```
DRL     RETFIL      [12]
ZERO    L(fileid),L(buff)
```

where

fileid (2 words) contains the file name in ASCII, or a right-justified 777 in word 1 if all files (except SY**) are to be returned

buff (BSS 380) is a work area used by the system

When a temporary file is returned, the file space and PAT-entry space are released and the file deleted from the AFT. When a permanent file is returned, the file system is notified to deaccess the file, the PAT-entry space is released and the file deleted from the AFT.

NOTE: A file that cannot be found in the AFT is considered by this function to be already released.

Space a Linked File

```
DRL     FILSP       [12]
ZERO    L(fileid),L(n)
ZERO    L(stat),0
```

where

fileid (2 words) contains the filename in ASCII

n contains the number of 320-word blocks, to be spaced (a negative value of n denotes backspacing)

stat (2-words) is the status-return location

This function spaces a linked file forward or backward n 320-word blocks, depending upon whether n is positive or negative, respectively. Loadpoint and end-of-file status indications are returned. An undefined-file condition is returned as device-busy status (major status 01).

If a request is made to space a random file, the requesting system is aborted and an error message is sent to the terminal.

Rewind a Linked File

```
DRL     REW      [10]
ZERO    L(fileid),L(stat)
```

where

fileid (2 words) contains the filename in ASCII

stat (2 words) is the status-return location

This function rewinds the linked file specified. The loadpoint status indication is returned. An undefined-file condition is returned as device-busy status (major status 01).

If a request is made to rewind a random file, the requesting subsystem is aborted and an error message is sent to the terminal.

Do I/O on User's File

```
DRL     DIO      [01]
Seek    command
ZERO    L(fileid),L(dcw1)
Read/Write command
ZERO    L(fileid),L(dcw2)
ZERO    L(stat),0
```

where

fileid (2 words) contains the filename in ASCII - 1 to 8 characters

dcw1 IOTD L(rbn),1

where rbn contains, for random files, the relative block number (set by user) (This word is always altered by I/O routines.)

dcw2 IOTD L(data),n

where data contains the starting address at which data is to be read/written, and n is the number of words to be transferred

stat (2 words) is the status-return location (See the GECOS manual for the status codes.)*

*See GECOS III Manual, CPB-1518.

I/O commands — the user need not be concerned about giving commands for a specific device type because the seek (34) command, write (31) command, and read (25) command will be accepted for all devices, and the actual commands used will be acquired for the particular device

This function is for files that appear in a user's list of files (AFT). It performs the equivalent of the MME GEINOS and performs the indicated seek, read, or write, using the master-mode routines. The subsystem is not eligible for execution until the I/O is complete.

Normal status is returned except device-busy, as such. This status is used to notify the user that his file name is not yet defined. Logical end-of-file is returned as major status 17. (True device-busy status is never returned by the GECOS file system.)

If the terminal user does not have the necessary permissions, or if an invalid relative block number is given for a random file, the requesting subsystem is aborted and an error message is sent to the terminal.

Add Links to Temporary File

```
DRL   MORLNK          [28]
ZERO  L(links),L(fileid)
```

where

links — (0-17) contains the number of additional links desired

links — (18-35) contain, on return, the number of links obtained (number requested or zero)

fileid — (2 words) contains the filename in ASCII

This function will acquire the additional number of links requested if possible, and update the users entry for the file.

Permanent-File Activity

Grouped under DRL FILACT 30 are the following permanent file functions:

Create Catalog (CC)	Purge Catalog (PC)
Create File (CF)	Purge File (PF)
Access File (AF)	Modify Catalog (MC)
	Modify File (MF)

They are differentiated by a function number which is passed in the upper-half of word 3 of the calling sequence. The DRL FILACT handles all permanent file requests with the exception of file deaccesses. Permanent-file deaccesses are handled by DRL RETFIL.

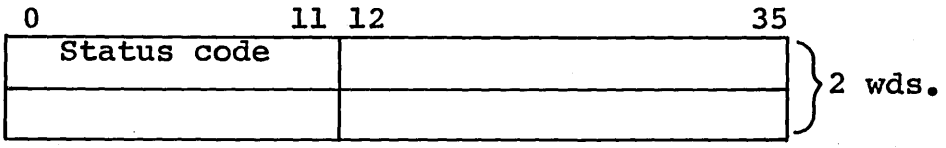
The following parameter descriptions are common to most of the DRL FILACT calling sequences. Following a calling sequence in which one or more of these parameters differ from the common layout, the description of such parameter(s) is given.

(1) buffer

```
buffer   BSS   380
```

This buffer is required in all cases as file system working storage.

(2) status return



Upon return, a status code of other than 4000_8 indicates that the request was denied.

Status codes:

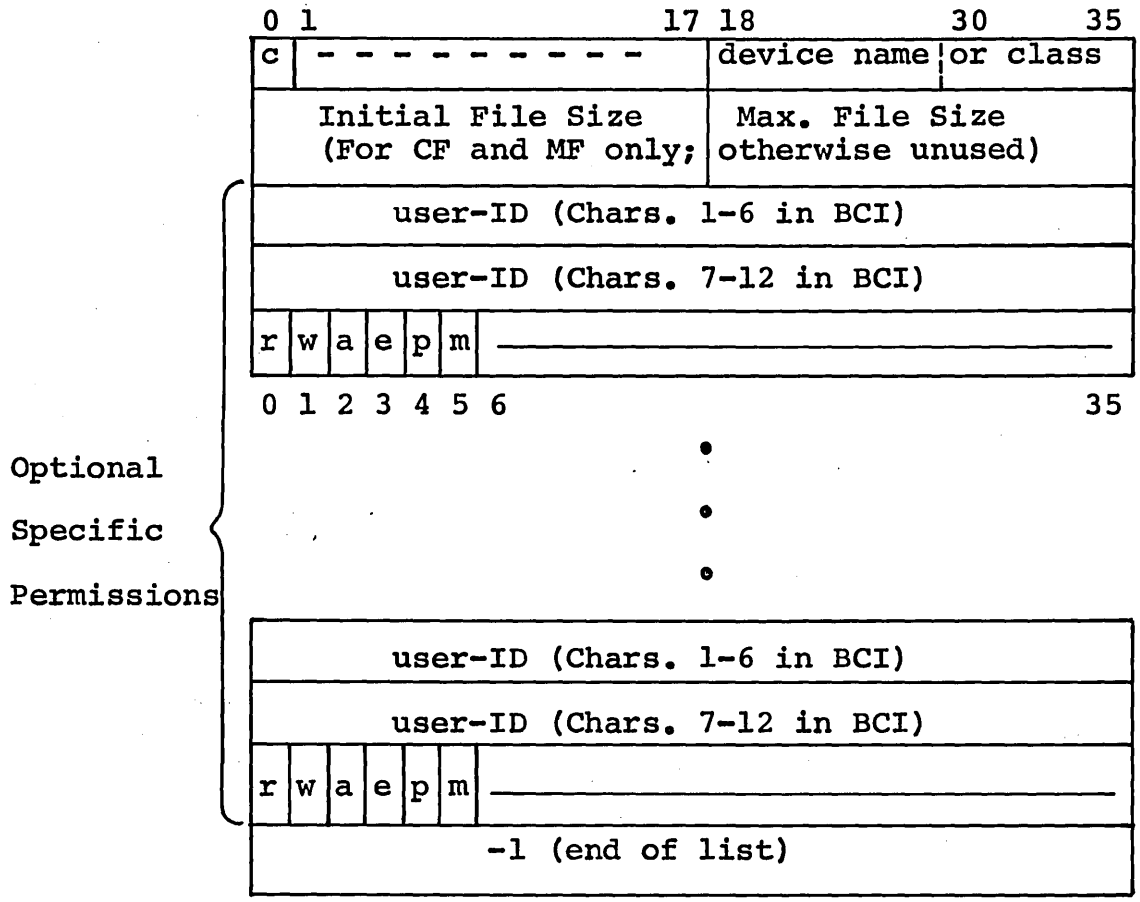
4000	No Errors
4001	Name Not In System Master Catalog
4002	I/O Error - Cannot Proceed
4003	Permission Denied
4004	File Busy
4005	Incorrect Catalog/File Description
4006	LLINK Space Exhausted
4007	Device Type Undefined
4010	LINK Space Exhausted
4011	Non-Unique Name
4012	Size Requested Less Than Current Size
4013	Requested Space Exceeds That Allowed
4014	Illegal Password
4036	AFT Is Full
4037	Duplicate Name In AFT
4040	No PAT Space Available

(3) permissions (assigned or requested)

If corresponding bit is set:

r = General Read permission
w = General Write permission
a = General Append permission
e = General Execute permission

(4) options



If corresponding bit is set:

c = Contiguous allocation desired
(CREATE FILE only)
r = Specific Read permission
w = Specific Write permission
a = Specific Append permission
e = Specific Execute permission
p = Specific Purge permission
m = Specific Modify permission

and device name or class is defined as:

Specific (named) device, in BCI -
e.g., DS1, where "DS1" would have been assigned as a specific device name in the installation's GECOS-startup deck.

Class of device (in bits 30-35) -

00 = DSU270 (large disc)
01 = DSU200 (standard disc)
03 = MDU200 (UNIVAC drum)
04 = MDU300 (Fairchild drum)

or where

-1 (bits 18-35) denotes the file with the most available space.

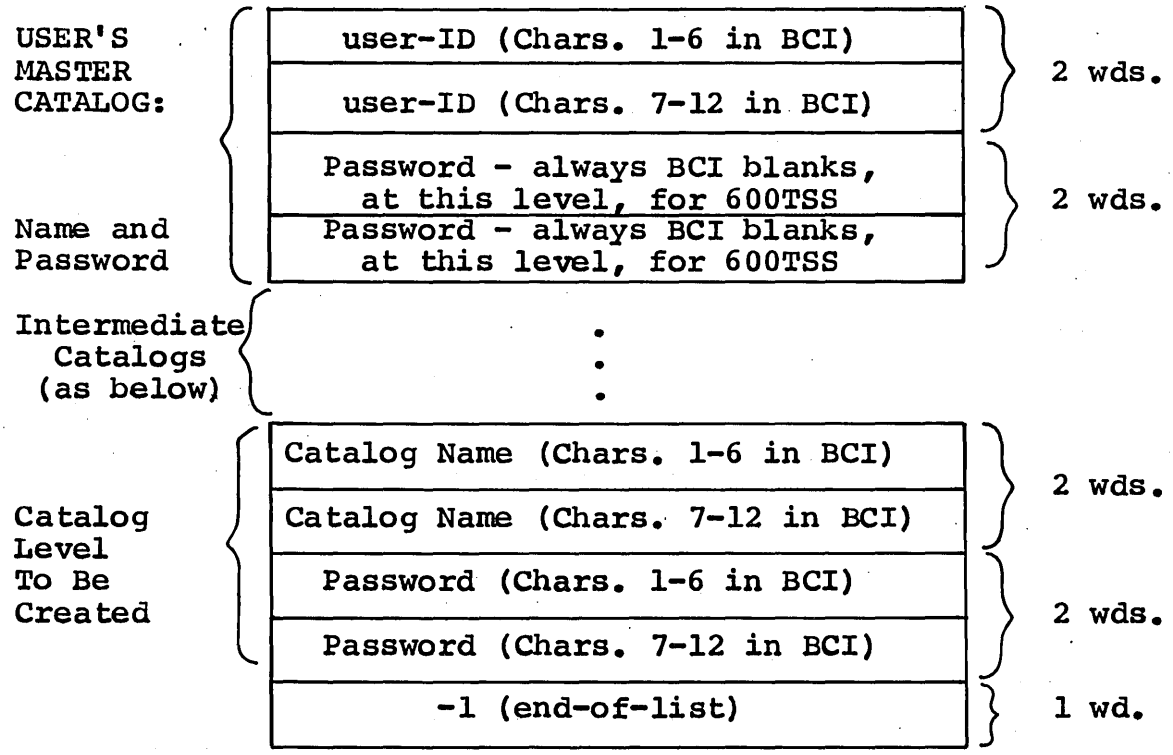
Create-Catalog Function

```
DRL     FILACT      [30]
ZERO    0,L(arglist)
ZERO    2,L(buffer)
```

where

```
arglist  ZERO  L(status-return),0
         ZERO  L(cat/file desc),L(permissions)

         ZERO  L(options)
```

cat/file desc



CPB-1514

(1) All names and passwords are left-justified with trailing blanks.

(2) -1 in place of the user's-master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

This FILACT function, identified by the function-number "2", creates the specified new catalog at the level indicated. All existing intermediate catalogs must be specified in the cat/file desc table (i.e., the complete catalog string).

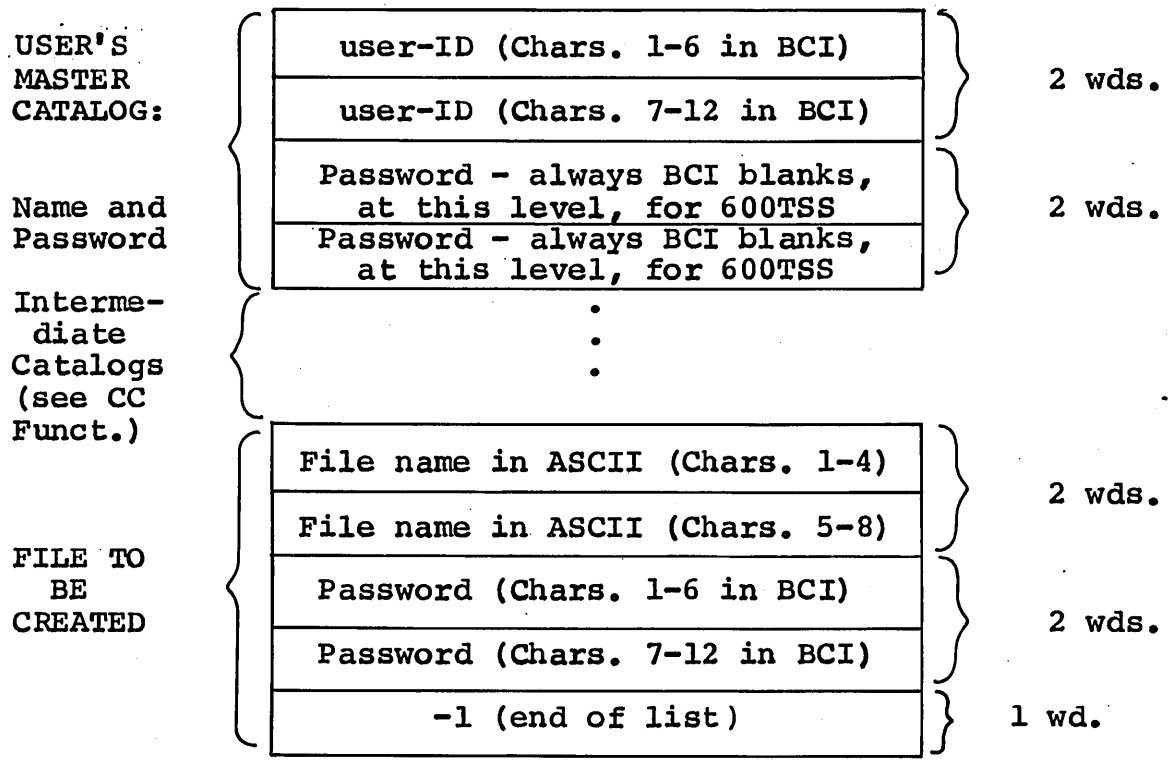
Create-File Function

```
DRL     FILACT
ZERO    0,L(arglist)
ZERO    3,L(buffer)
```

where

```
arglist  ZERO  L(status return),0
         ZERO  L(cat/file desc),L(permissions)
         ZERO  L(options)
```

cat/file desc



(1) All names and passwords are left-justified with trailing blanks.

(2) All entries are in BCI, except for the file name.

(3) -1 in place of the user's master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

The Create-File function creates a permanent-file description from the information specified in both the cat/file desc and options parameters and will acquire the necessary file space. The file name is not entered in the user's AFT (see "Access-File Function").

Access-File Function

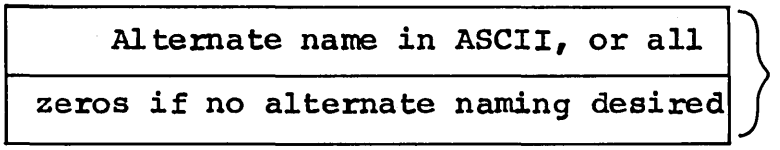
```
DRL     FILACT
ZERO    L(alternate name),L(arglist)
ZERO    4,L(buffer)
```

where

arglist ZERO L(status-return),1 for random/0 for linked file

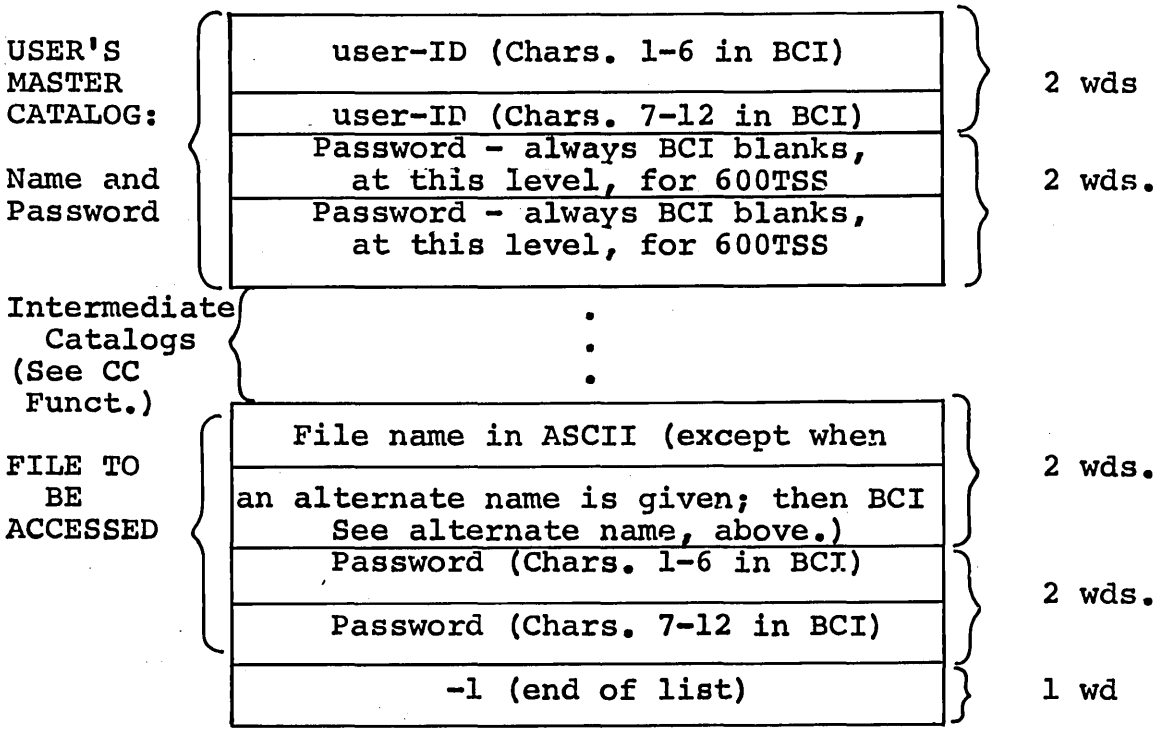
ZERO L(cat/file desc),L(permissions)

alternate name



This two-word entry is used when a file is to be accessed by a name other than that by which it was created. That is, a file created in the batch environment with a name of more than 8 characters, or a file whose name is the same as one already in the user's AFT.

NOTE: When an alternate name is used, the defined file name in the cat/file description must be in BCI and the alternate name in ASCII.



(1) All names and passwords are left-justified with trailing blanks.

(2) All entries are in BCI, except for the file name or alternate name.

(3) -1 in place of the user's-master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

The Access-File function places the specified file in the user's AFT and sets the file busy consistent with the permission(s) requested.

The file is placed in the AFT under its actual file name or under an alternate name, as indicated. (The effect of alternate naming is restricted to the AFT associated with the current user, and does not in any way change the file's definition in the file system.) An alternately-named file, if returned (released from the AFT -- see DRL RETFIL), must be returned under its alternate name.

Note that files created through the DRL FILACT Create-File function may be treated as either linked or random files. Their intended use must, however, be specified in each file-access request, as previously indicated in word 1 of arglist.

Upon a successful return from this function, the A-register contains the following information:

Bits	Meaning
0-5	Device type
6-17	Number of words per physical block

- if bit 18 = 0, then

24-35	Number of links in the file

- if bit 18 = 1, then

24-35	Number of blocks in the file

- if bit 19 = 0, linked file

- if bit 19 = 1, random file

20-23	Unused

Purge Catalog/File Function

```
DRL     FILACT      30
ZERO    0,L(arglist)
ZERO    n,L(buffer)
```

where

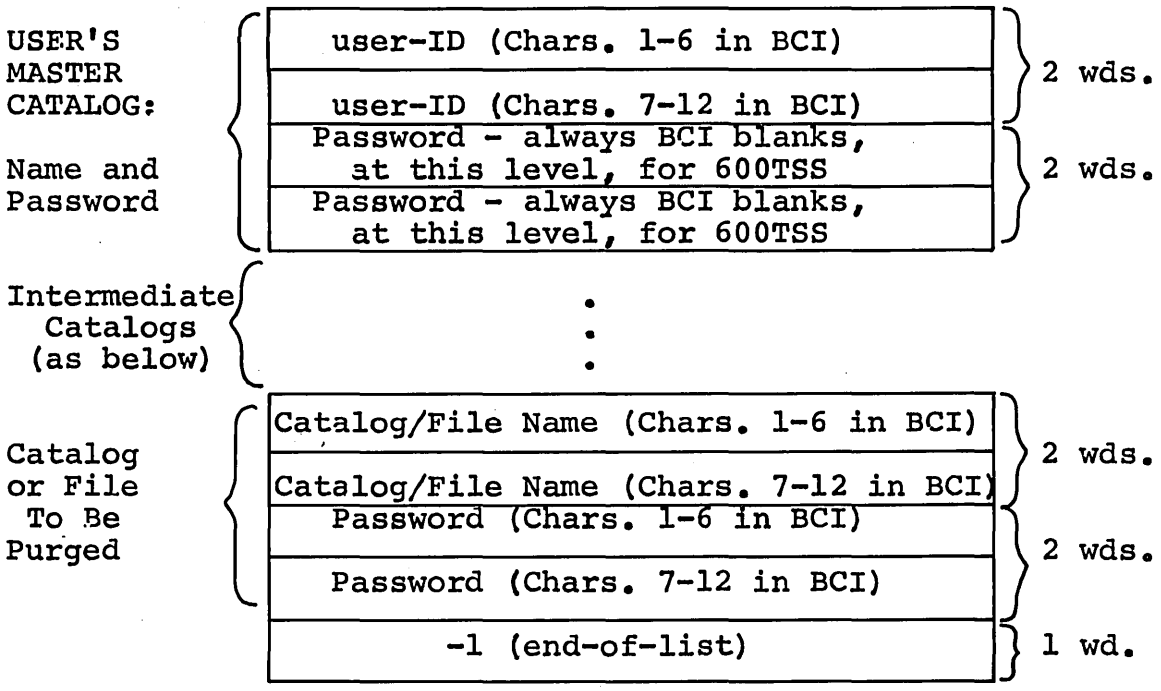
arglist

```
ZERO  L(status return),0
ZERO  L(cat/file desc),0
```

n (function number)

```
n = 8 - Purge Catalog
  = 9 - Purge File
```

cat/file desc



(1) All names and passwords are left-justified with trailing blanks.

(2) -1 in place of the user's-master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

At the file level, this function deletes the file description from the file system, and releases the corresponding file space. At the catalog level, this function deletes the named catalog and all catalogs and files subordinate to it. Any corresponding file space is released.

NOTE: User's master catalogs cannot be purged under 600TSS.

Modify Catalog/File Function

```
DRL     FILACT
ZERO    0,L(arglist)
ZERO    n,L(buffer)
```

where

arglist

```
ZERO  L(status return),0
ZERO  L(cat/file desc),L(permissions)
ZERO  L(options),L(newname)
```

n (function number)

n = 10 - Modify Catalog
n = 11 - Modify File

permissions follows the common layout, except as indicated under (3) below, and is used as follows:

(1) change the assigned general permissions - the permission bits (0-3) must specify the new set of permissions (i.e., not just additions)

(2) delete all general permissions - the permissions word must contain all zeros

(3) indicate no change of general permissions - the permissions word must contain a -1 (i.e., all 1 bits)

options follows the common layout, and is used as follows:

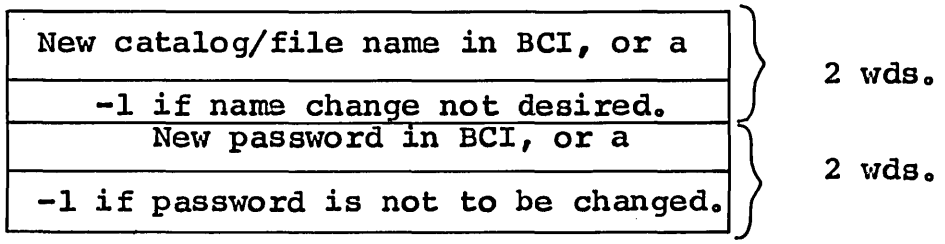
(1) change the assigned specific permissions - the permission bits must specify the new set of permissions (i.e., not just additions)

(2) delete all specific permission (for one user) - all permission bits must be zero

(3) indicate no change - omit entry for the user whose specific permissions are to remain unchanged

(4) change size in Modify File - the new maximum size (only) is indicated in lower-half of options +1

new name



This function will modify the description of a catalog or file, depending upon the function number specified. Unlike alternate naming in Access File, the changes made by this function are permanent.

600TSS FILE USAGE

Temporary User Files Assigned by 600TSS

The usage of standard temporary-user-files is described here on the basis of what is done by the GE-supplied 600TSS systems, primarily BASIC and EDITOR. The designer of a new subsystem which requires a source file for each user may select this usage, both for overall system consistency and to take advantage of facilities already provided in 600TSS.

There are two standard temporary files for each terminal user: (1) the input-collector file, SY**, and (2) the source file, *SRC.

Input-Collector File (SY**)

This file is automatically assigned to each terminal user by the TSS Executive. All terminal input except command language is collected on this file while the system is in build-mode (see Build-Input primitive in Chapter 4). This is the 'raw' data just as it is received from the terminal. The collection of input is performed by the Line-Service portion of the TSS Executive, that is, no subsystem is in execution. Thus, the assignment of SY**, the collection of input data on it, and the scanning of the input for command language are automatic functions of 600TSS, provided that the selected subsystem uses build-mode for the collection of new or additional input destined for a source file. Examples of the input are language statements in the case of BASIC, or text in the case of EDITOR. The format of this file is described in Chapter 6.

Source File (*SRC)

A source file (*SRC) is assigned to a user by the OLDN (Old-New) subsystem. The OLDN subsystem is called by the first primitive in the startup procedure of both BASIC and EDITOR. OLDN produces the old-new file request sequence -- OLD or NEW-, and conditionally, OLD NAME-.

The source file receives the 'edited' and/or merged version of the file with which the user is currently working. For example, if the user is writing a 'NEW' BASIC program, the input collector (SY**) file contains all of the raw input, including any mistakes and corrections, other than keying errors corrected by "@" or "CTRL/X".

When the user gives one of the BASIC commands, this causes BSED to edit the data on SY** -- all corrections applied, duplications removed, etc. SY** is then written to the source file, *SRC, which is the copy that is listed, run, and/or saved.

In the case of an 'OLD' BASIC program, the OLD file is copied directly to the user's *SRC file. Any changes that are typed are collected on SY** until a BASIC command is given. This causes the SY** file to be edited and then merged with the data on *SRC, and the new, merged copy written to *SRC. Again, it is this new copy of the program that is run, listed, and/or saved.

In the case of an OLD file, the user is always working with a copy of that file on *SRC -- either as-is, or modified by SY** data -- and not the original. This provides an automatic backup copy, i.e., the OLD (permanent) file.

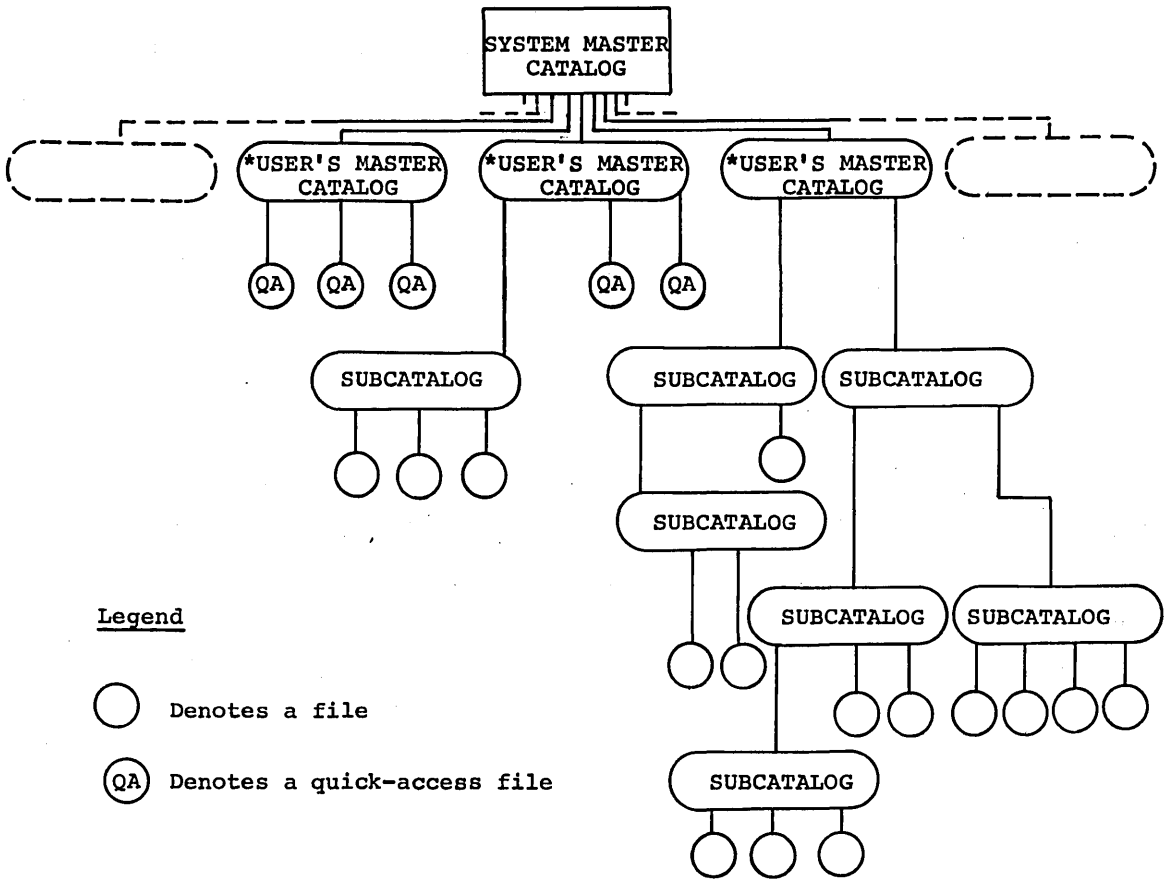
Permanent Files Assigned by User

The 600TSS never assigns permanent file space to a user unless specifically told to do so by that user. Permanent files are handled by the GECOS file system which is common to all programs operating under the GE-600 Comprehensive Operating Supervisor (GECOS).

Structure of the File System

The GECOS file system is described in detail in GECOS File-System reference manual, CPB-1513. The main points of interest to the 600TSS user are repeated here.

The GECOS file system is, in formal terms, a tree structure of indefinite length whose origin is the system master catalog. The primary nodes of the tree are user's master catalogs; the lower-level nodes are subcatalogs created by the user. The terminal points of the structure are the files themselves. A schematic representation of the file system's hierarchical structure is shown in Figure 1.



All user-ID's must be unique within the system; all subcatalog and file names are automatically qualified by the user's master catalog name and the names of any intermediate subcatalogs. The system master catalog cannot be accessed by the normal user.

*Identified by the user-ID.

Figure 1. Logical Structure of the GECOS File System.

Catalogs and Files

A catalog consists of a description containing catalog name, password, and permissions. A catalog cannot be read or written since it contains no user data.

To the GECOS file system, a file consists of a description containing file name, file size, password, permissions, and the specification of the physical file space. The file description is distinct from the physical file space which may contain user data and can be read or written.

Passwords

Passwords can be attached to any catalog or file. A password simply allows a user to traverse a catalog/file string. The user can get to a given catalog or file only if he can give the passwords for all higher-level catalogs in the string. And, the originator of a given string must also give the required passwords when traversing that string. However, when traversing a string, a password must not be given if none has been attached.

Permissions

Permissions, both general and specific for users, can be attached to any catalog or file. When permissions are attached at the catalog level, they apply to all subordinate catalogs and files. The originator of a catalog/file string has all permissions for that string, but must give all applicable passwords.

The allowable permissions are:

```
READ    -- allows a file to be read (referenced)
WRITE   -- allows a file to be written
APPEND  -- (presently treated as WRITE)
EXECUTE -- (presently treated as READ)
PURGE   -- allows catalogs and/or files to be
           purged (specific permission only)
MODIFY  -- allows catalog and/or file definitions to
           be changed (specific permission only)
```

Concurrent Use of a File

Multiple concurrent readers (or executors) of a file are allowed by the file system, but any other combination of access-modes are mutually exclusive. The following table shows how the file system accepts or rejects subsequent access-request for the same file, based on the permission(s) requested by the initial accessor of the file.

Initial Accessor	Subsequent Access Requests			
	READ	EXECUTE	WRITE	APPEND
READ	OK	OK	X	X
EXECUTE	OK	OK	X	X
WRITE	X	X	X	X
APPEND	X	X	X	X

User's Contact with the File System

The terminal user's contact with the GECOS File System is through the Old-New (OLDN) and Save-Purge (SAVE) subsystems, and possibly the ACCESS subsystem.

OLDN, when OLD is selected, writes the contents of the permanent (OLD) file onto the user's source file, *SRC. SAVE writes the contents of *SRC onto the named permanent file. (See the descriptions of OLDN and SAVE in Chapter 7.) In the case of either subsystem, to "access" a permanent file means to enter it into the user's available-file table (AFT) as explained in the following description of the AFT.

File Usage by Subsystem Programs

The 600TSS maintains a table of active files for each user. Before any I/O can be done on a file, an entry for that file must be placed in this table -- Available-File Table (AFT).

The AFT allows a complete file description to be kept in core, thus minimizing the access time for these files. The AFT also allows files to be identified by their file name alone; for permanent files the full file description may consist of many catalogs and passwords.

Getting File Entries Into and Out of the AFT

Since subsystem programs perform I/O and related functions via the file I/O derails, only the general use of the file I/O derails is described here.

Temporary Files

DRL DEFIL (define and access a temporary file) creates a temporary file and places the file entry in the AFT. All temporary files defined by subsystems should contain at least one special character (i.e., not alphabetic, numeric, period or dash) in the file name. Since special characters are not allowed in permanent file names defined from a terminal, this avoids any conflict.

DRL RETFIL (return a file) removes the file entry from the AFT and releases the file space. When a subsystem is finished with a file it should return it. All user's AFT files are released upon termination.

Permanent Files

DRL FILACT, function number 4 (access a permanent file) places the file entry in the AFT, and sets the file busy for the permission(s) requested.

NOTE: This function does not create a file. Before a permanent file can be accessed it must have been created by DRL FILACT, function number 3 (create a permanent file).

DRL RETFIL (return a file) removes the file entry from the AFT and sets it "unbusy" with respect to the current user.

Doing I/O on the File

After the file is placed in the AFT, the following can be executed:

DRL DIO (Do I/O on a file)

Reads or writes a file

DRL FILSP (Space a file)

Positions a file forward or backward

DRL REW (Rewind a file)

Positions a file to its beginning

DRL MORLNK (Add more links)

Increases the size of a temporary file

These are the only file I/O derails that affect or relate to the AFT and are the ones which will be most used by subsystem programs. The others, all of the FILACT functions except number 4, affect only the GECOS file system.

4. COMMAND LANGUAGE AND PRIMITIVES

In the design of many time-sharing programs, it is desirable that unique command language be recognized and that sequences of processes be initiated based on these commands. In 600TSS it is possible to define command language independently for each subsystem. Each command word has associated with it a list of primitives that are specified by the system designer. These primitives are interpreted by 600TSS to control the processes implied when the command is recognized. Command language is recognized by 600TSS only when the system is in the build-mode of keyboard input. The two keyboard input modes are described in "Keyboard Input Modes".

The command-language list and primitives are incorporated in the 600TSS Communication Region (block .TPRGD), along with the primary portion of the program descriptor (refer to Chapter 2).

The primary portion of the program descriptor is arranged with those for other subsystems in a contiguous block for rapid scanning. Table 2 shows the detailed format of an individual entry in the .TPRGD block for a single subsystem. Table 2 describes each entry or entry-type shown in Figure 2.

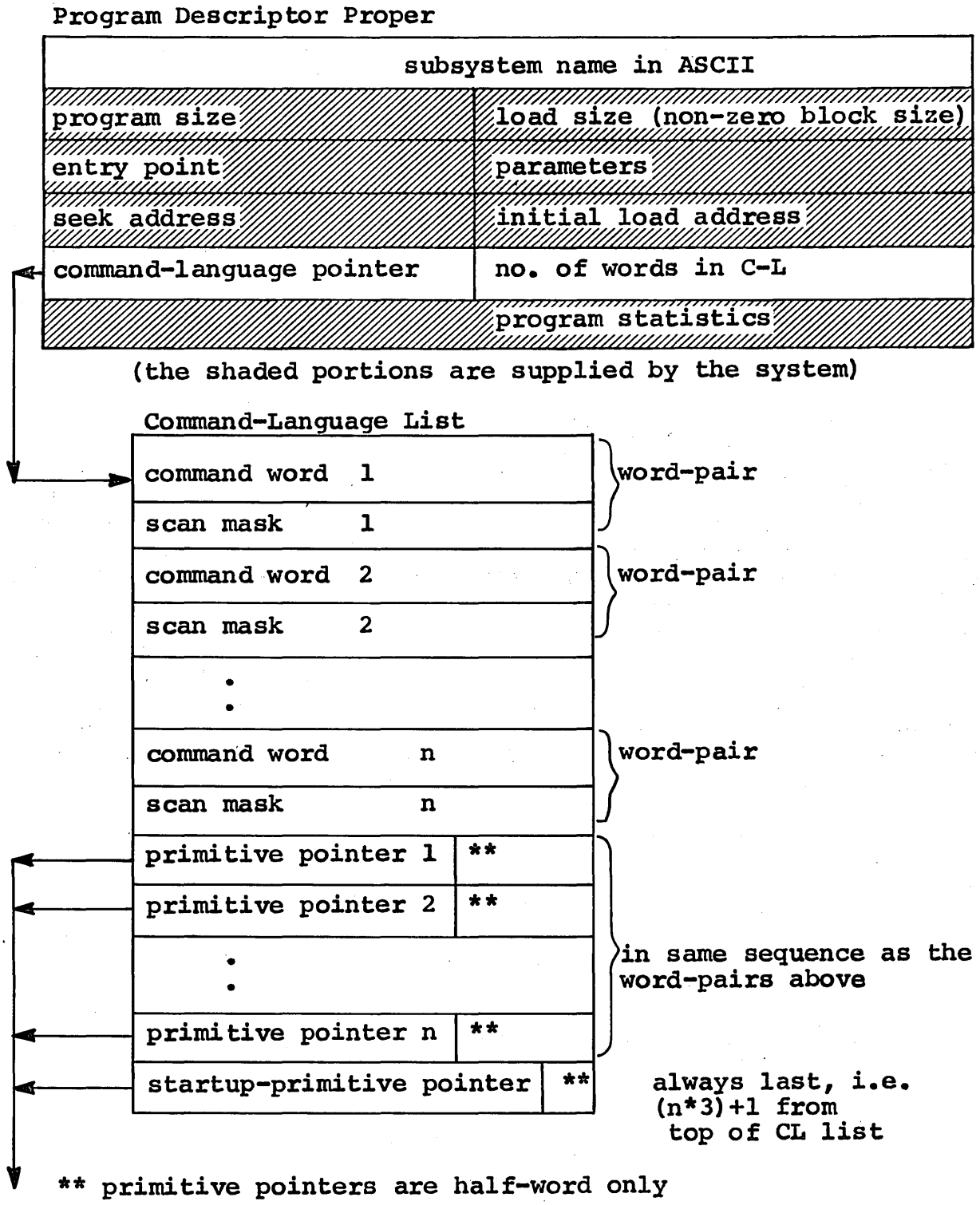


Figure 2. Program Descriptor Format

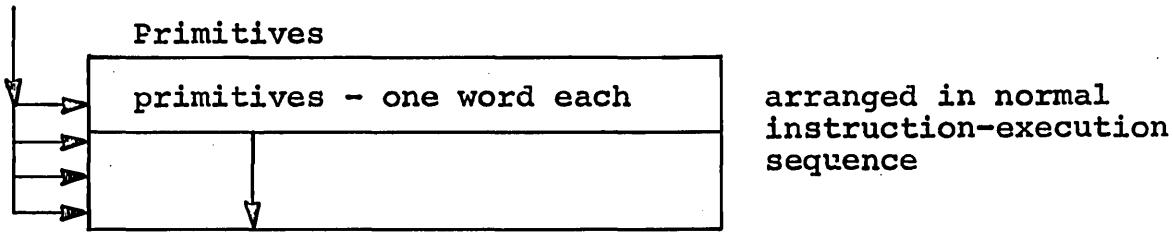


Figure 2. Program Descriptor Format

Entry	Description
Name in ASCII	Name of subsystem to be used to identify the program in response to the user's reply to SYSTEM?
Program size	Actual program size to be used in execution. The base register will be set to include this region. (Supplied by the system.)
Entry point	Address relative to zero that contains the first executable instruction. (Supplied by the system.)
Load Size	The size remaining when all leading and trailing zeros have been eliminated. This is used to reduce the size of the original copy of the program. (Supplied by the system.)
Initial Load Address	The address of the first non-zero word in the program. (Supplied by the system.)
Parameters	Flags defining the type of program: privileged, master subsystem, or normal subsystem. (See end of table.)
Seek Address	Location on mass storage where the original copy is stored. (Supplied by the system.)
Command Language Pointer	Address of the first word of the command-word/mask pairs.
Number of Words in Command Language	The number of command-word/mask pairs.
Program Statistics	Statistics kept by 600TSS Executive regarding usage of this program.

Command Word and Mask	Word pairs defining the command-language word and number of characters in the word: command word — mask XXXX — 0 XX — 777777 XXX — 777 The mask is used in the CMK instruction to remove irrelevant trailing characters from the comparison scan of input data. The command words, of up to four characters, must be in ASCII.
Primitive Pointers	Address of groups of primitives to be executed when a command language word is encountered. These pointers are in the same order as the command language words.
Primitives	See "Description of Primitives" for definition of the primitive format.
Startup Primitive Pointer	The last primitive pointer. This points to the block of primitives to be executed when the program is initiated from some source by a CALLP primitive.

PROGRAM-DESCRIPTOR PARAMETER DEFINITIONS

Bit	18-32	--	Not defined
Bit	33	--	Patches are in patch table
Bit	34	--	Do not set base register on dispatch
Bit	35	--	Permitted privileged derails

Table 2. Program-Descriptor Entries

KEYBOARD INPUT MODES

Two modes of keyboard input are available to a 600TSS subsystem. They are the direct-mode and build-mode.

In the direct-mode, the subsystem program in execution requests input. The requested input passes directly to the subsystem and no scan or interpretation is made by the 600TSS Executive. Thus, while in this mode there is no recognition of commands by 600TSS. The subsystem program can, of course, interpret the input, and take appropriate action.

In the build-mode no subsystem program is actually in execution and input is under control of 600TSS. All input is collected and written to an internal file (SY**) maintained for each user. Each line of input is scanned for command language while in this mode. When command language is found the execution of the associated primitives is initiated. The build-mode of input is initiated by the primitive Build Input.

DESCRIPTION OF PRIMITIVES

Format

Each primitive occupies one 36-bit word. The primitives, once initiated, are normally executed in sequence. Some primitives allow conditional transfer of control. These cause the execution to continue with another primitive at the specified location. The format of a single primitive is P, N, A, where

P is the primitive operation
N is an optional integer argument
A is an optional address

Macros exist for expansion of the primitives into the proper machine-word format.

Primitives

- CALLP a

  This primitive transfers control to the named subsystem program. The 600TSS finds the startup sequence in the program descriptor of the called program and takes its next primitive from this list. This CALLP primitive may occur in any list of primitives of a program descriptor and will interrupt the execution of primitives from the present list. However, it is normally necessary that control be returned to that level after the series of functions performed by the called program are completed. The location of the primitive when the CALLP was encountered is saved in a pushdown list in the user-status table. Thus it is possible to have several levels of calls and to be able to resume operation at a previous level. The primitive POPUP will resume operations at the previous level.

- EXEC

  This primitive initiates the loading and execution of the current subsystem program. This is accomplished by placing this job in the new interaction queue for the allocator. When the subsystem program has completed its functions, the subsystem returns control to the Executive via a DRL RETURN operation. This causes the next primitive in sequence to be executed.

- BIN

  This primitive initiates the building of input. While in this mode, 600TSS reads and accumulates data on a collection file (SY**) for a given user. While in this mode, each line of input is scanned for the command language associated with the subsystem. If no command language is found, 600TSS accumulates the input in a buffer and dumps it when required to the user's input collector file. If a command word is recognized, 600TSS Executive does the necessary housekeeping before the command is executed. Note that once the Build Input primitive is encountered, there is no "next" primitive implied. The next primitive will be defined when a command word is encountered.

- POPUP

  This primitive indicates that processing at this level is completed and that processing at the previous level is to be resumed. The 600TSS obtains the previous set of pointers from the user-status table, obtains the next primitive and continues the flow of control from that point. If the previous level does not exist, that is, this was the first level of control, POPUP calls the system routine which asks the user which subsystem he wishes to select next. All files defined during previous calls remain defined.

- IFALSE n,a

  This primitive provides for conditional execution of another block of primitives. The conditional test is based on the subsystem switch word (see Chapter 3). The interpretation is, "If bit n is false (off), transfer control to the block of primitives at location a". If the test is true (on), control passes to the next primitive in sequence. This function allows considerable interaction between the execution of subsystem programs and the interpretation of primitives. A subsystem can, via the appropriate derail, set or reset these switches. Bit positions are 0-35 counted from left to right.

- IFTRUE n,a

  The interpretation of this primitive is the same as IFALSE, except that transfer of control passes to a if the test is true (on).

- STFALS n,a

  This primitive provides the capability of setting the switches in the individual user-status-table. This allows considerable interaction with the subsystem programs since they can also test these switches. The subsystem program can then execute different blocks of code based on the setting of switches made by the primitives, which, of course, could be different for different sequences of primitives. The interpretation of the above primitive is, "set bit n false (off) in the present user-status-table and transfer control to the block of primitives starting at location a".

- STRUE n,a

  The interpretation of this primitive is the same as STFALS, except that the switch bit is set true (on).

STARTUP PROCEDURE

One set of primitives is always part of the program description for each subsystem program. This is the startup procedure that is used to initiate the process when a subsystem is selected. This provides potential flexibility in allowing initialization procedures before the subsystem program is executed.

EXAMPLES OF PROGRAM DESCRIPTORS

Two examples of subsystem program descriptors follow.

Example 1:

This subsystem, AB, has no command language and is to be executed directly when selected. When the subsystem completes its function control is to be returned to the previous level. The system designer must supply only the program name , a pointer to the null command-language list, a pointer to the startup procedure, a primitive to cause execution of the subsystem, and a second primitive to cause control to return to the previous level. This sequence is adequate for many subsystems placed into 600TSS. The program descriptor is shown in source-language form (utilizing the PRGDES macro):

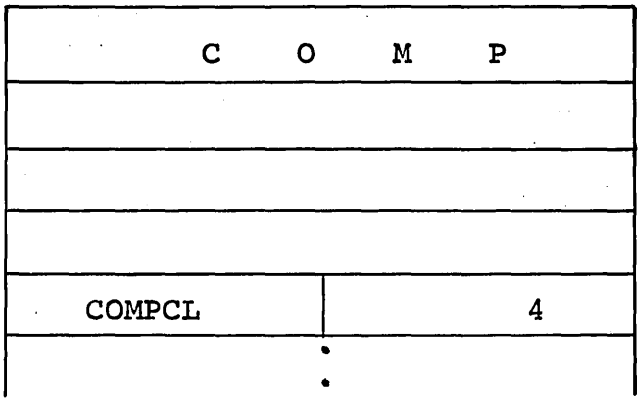
```
        .
    (program descriptors)
        .
        .
        .
      PRGDES  AB,ABCL,0        Program descriptor

        .                      (name, command-
        .                      language ptr., no.
        .                      of command-language
        .                      words)

    (end of program descriptors)
        .
   ABCL   ZERO  ABPRIM         Pointer to startup
        .                      primitive
        .
ABPRIM EXEC                    Load and execute
       POPUP                   Return to previous
                               level
```

Example 2:

The subsystem COMP has four command-language words: LIST, GO, MERGE, and BYE. It calls on a subsystem FILE in the startup procedure, and calls the subsystem's LIST and MGR during execution of command language. The COMP program is executed only upon receipt of the command GO. (During execution of COMP, it is, of course, possible for that program to recognize certain direct-mode input as its own "commands".)

Note that in primitives P2 and P3, bit 17 and bit 18 of the user's switch word are referred to, respectively. The significance of bit 17 is assigned by 600TSS: if on (true) valid data exists on SY**; if off (false) no valid data exists on SY**. This bit is set on or off automatically when the system is in build-mode. The significance of bit 18, however, is assigned by the subsystem in question, also bits 19 through 35.

PROGRAM DESCRIPTOR



COMMAND LANGUAGE

COMPCL

L	I	S	T
000	000	000	000
G	O		
000	000	777	777
M	E	R	G
000	000	000	000
B	Y	E	
000	000	000	777
P1	(list)		
P2	(go)		
P3	(merge)		
P4	(bye)		
P5	(startup)		

PRIMITIVES

P1	CALLP	LIST
	BIN	
P2	IFALSE	17,P2.1
	CALLP	MRG
P2.1	EXEC	
P4	POPUP	
P3	IFTRUE	18,P3.1
	CALLP	MRG
P3.1	BIN	
P5	CALLP	FILE
	BIN	

5. PROCEDURE FOR PLACING PROGRAMS IN THE SYSTEM

Subsystem programs may be placed in the Time-Sharing System either permanently or on a temporary basis. The temporary placement of a subsystem does not require an edit of 600TSS or construction of a program descriptor. This provides a more convenient means of loading and checking out a version of a subsystem still under development. Since some of the procedures required for permanent placement of a subsystem are also required for temporary placement, the former is described first.

PERMANENT PLACEMENT

There are four steps necessary to placing a subsystem permanently in the Time-Sharing System:

1. Write and assemble the program.
2. Edit into the GECOS III System.
3. Prepare and assemble the program descriptor and command-language/primitive list into the 600TSS Communication Region.
4. Modify and reassemble the TSTRT module of 600TSS to cause the subsystem to be included in 600TSS initialization.

Step (1) summary follows and steps (2) through (4) are discussed in detail.

Writing the Subsystem Program

The several restrictions and available facilities for writing 600TSS subsystem programs, discussed in previous sections, are summarized here:

- An unused data area of 100_{10} words must precede the subsystem coding, for use by 600TSS. Therefore, the first storage definition statement must be at least a BSS 36. The 64 words normally reserved by the loader can be used as part of the required area.

- If it is desired that the subsystem process any or all of the faults from which recovery is possible under 600TSS, coding to store the fault-vector transfers must be included in the subsystem.

  They cannot be loaded.

- No MME instructions are permitted. The analogous DRL functions defined by 600TSS must be used.

- For coding convenience, two macros are available for general use. These TSS macros are called by LODM .G3TSS. Then the macro call .SSDRL provides the derail address-value/mnemonic equivalences. The macro PRNTTY causes a message to be printed at the terminal; the format of this macro is:

  ```
  PRNTTY  n,(message),k
  ```

  where n is the number of characters in the message, and k (optional) is a pointer to a word containing control characters to be affixed to the end of the line (CR, LF, NULL, etc.). The length of the message is limited to the space available on the punch card between the n and k fields.

  The appendix "System Macros" describes these and other available 600TSS macros.

- All character input/output, file names, etc. must be in ASCII.

Editing Program to GECOS III

Editing to the GECOS III System is performed by the standard System Editor procedure. Briefly, the deck setup is as follows:

```
$SYSLD      CATALOG=.TSxxx
$LOWLOAD
$OPTION     NOSETU
$NOLIB
$OBJECT
   •
 program-binary deck(s)
   •
$DKEND
$ENTRY      ee...e
$EXECUTE
$ENDLD
```

The 600TSS catalog names are prefixed by .TS so the user need select for his subsystem a three-character identifier (xxx) that is unique only among the 600TSS subsystems. When placed with the rest of the subsystems, the deck(s) are edited into the GECOS III System.

Assembling the Program Descriptor

The program descriptor and command-language/primitive lists must be constructed next. Chapter 4 gives the format. These must be assembled into the 600TSS Communication Region deck, CD600T1.001, TSS-TSSA, and specifically within block .TPRGD. The definition of .LNPD, number of entries in the program descriptor list, may also need to be modified if the number of entries has been exhausted.

An inspection of the listing indicates the required position of the program-descriptor proper, and the conventional placement of the command-language and primitive lists. The program descriptor must be contiguous with the other program descriptors; the command-language/primitives may be placed anywhere following the last descriptor.

The PRGDES macro is provided for constructing the program descriptor:

```
PRGDES x, y, z, n
```

where

```
  x = subsystem name
  y = command-language pointer
  z = number of command-language words
n=0 - EXEC primitive exists in list
≠0 - no EXEC primitive exists in list
     (e.g., only CALLP's used)
```

Also, each of the primitives is generated by a macro called as follows:

CALLP	a	Call subsystem a
EXEC		Execute program
BIN		Build Input (go into build-mode)
POPUP		Return to previous control level
IFALSE	n, a	If bit n false, go to a
IFTRUE	n, a	If bit n true, go to a
STFALS	n, a	Set bit n false, go to a
STRUE	n, a	Set bit n true, go to a

Modifying the TSTRT Module

Lastly, the 600TSS deck CD600T1.015, TSS-TSTRT must be modified and reassembled so that the new subsystem is initialized and made known to the TSS Executive. A three-word entry to the table IN900 is made as follows:

word 1 -	ASCII	1, name	Subsystem name
2 -	BCI	1, .TSxxx	Catalog name
3 -	ZERO	parameters	Bit 35 = privileged program
			Bit 34 = master subsystem

If at any time it is not desired that a subsystem be initialized at load time, a zero patch at word 1 of the IN900 entry for that subsystem will suppress its inclusion in the TSS program file and initialization of its program descriptor.

Coordination of the Procedure

It is important that the steps for placing a program in the system are well coordinated. The subsystem may be edited to GECOS III (step 2) and the program descriptor assembled (step 3) at any time, but these must have been performed before TSS-TSTRT is modified and the system reloaded, or the 600TSS initialization will abort.

TEMPORARY PLACEMENT

The procedure for temporarily loading and checking out a developmental subsystem, without doing an edit of the system, utilizes the LODX subsystem. Octal patches may be made after loading and prior to execution.

In addition to its use in checkout, LODX might also be used to load programs not integrated into 600TSS, or to load little-used 600TSS subsystems.

Placement

The subsystem program portion is written and assembled as for permanent placement. (The program descriptor need not be assembled; it will not be referenced. Do not modify TSTRT.) The following steps are then performed:

- Create a random permanent file using ACCESS. The subsystem is stored and referenced from this file.
- Submit the program deck(s) as a GELOAD activity to GECOS III with the following deck set-up:

```
$OPTION SAVE/prog-name,NOGO
    o
      other control cards
    .
    .
      subsystem deck(s)
    .
$EXECUTE
$LIMITS
$PRMFL H*,,R,cat-name/filename
$ENDJOB
```

Loading the Subsystem

After submitting the decks as a GELOAD activity, the user may have the subsystem loaded and start check-out from a terminal.

In response to SYSTEM?, specify LODX. For FILENAME?, give the filename specified on the $PRMFL card. If patching is desired type an asterisk as the second field preceded by a comma.

Examples:

```
FILENAME?  A123
FILENAME?  A123,*
```

Octal Patching

If the asterisk was typed as part of the response to FILENAME?, the program is loaded and a carriage return, line feed, and question mark are given. The user then may type in patches in one of the following forms:

1. ?address patch
2. ?address patch1,patch2,...

In both forms, the address and patch must be separated by a single blank. All addresses and patches must be in octal, and the addresses are relative to the load map produced when the file was written. In the second type, sequential patches may be given beginning at the specified address. In this form, the patches are comma-separated and as many may be given as will fit on the line. Leading zeros need not be typed.

A type-in of DONE indicates that patching is completed:

```
? DONE
```

The subsystem program is then executed.

DEBUGGING FACILITY

A standard debugging subroutine is provided for inclusion in a subsystem during checkout. It may be used with a subsystem that is placed in the system either temporarily or permanently.

Title: Terminal Debug Subroutine (TDS)

Purpose:

The subroutine aids in checking out a 600TSS subsystem by allowing the user to gain control at selected locations within the subsystem. When TDS is in control at these locations, the user may display and/or patch selected areas of the subsystem and then either return to the subsystem normally, or to a specified location within the subsystem.

Usage During Subsystem Preparation:

The subsystem to be checked out must contain a SYMREF to TDS. At each location where control is to be gained by the user, the instruction:

```
XED  TDS
```

must be inserted and assembled with the subsystem. Then when the subsystem is either edited to GECOS-III or submitted as a GELOAD activity, a binary object deck of the TDS subroutine must be included.

Usage During Subsystem Checkout:

The user calls the subsystem to be checked out by the same procedure as would ordinarily be used. When any of the locations at which he indicated that he wished to gain control (XED TDS instruction) are executed, the following message will appear at the user's terminal:

```
xxxxxx FUNCTION?
```

where xxxxxx is the octal address, relative to subsystem zero, at which the XED instruction is located.

The possible responses to this message are listed below along with their respective results. To initiate transmission, all responses must be followed by a carriage return.

Response: S

This response indicates the user wishes a Snap, or display of certain memory locations. The TDS subroutine responds with a line feed and a question mark indicating it is ready to accept parameters. The parameters may be in any of the following forms:

aaa,nn — where aaa and nn are octal numerics implying that nn locations should be displayed beginning at location aaa

aaa-bbb — where aaa and bbb are octal numerics implying that all locations from aaa to bbb, inclusive, are to be displayed

aaa — where aaa is an octal location which is to be displayed

DONE — indicates the user is finished with the Snap function and wishes to select a new function

Only one set of parameters may be specified at a time, and when the request is satisfied the TDS subroutine responds with another question mark indicating that a new set of parameters may be typed.

Response: P

This response indicates the user wishes to patch, or replace the contents of, selected locations within the subsystem. The TDS subroutine responds with a line feed and a question mark indicating it is ready to accept parameters. The parameters may be in any of the following forms:

aaa/bbbb — where aaa is the octal location at which the octal patch bbbb is to be made. Fields aaa and bbbb must be separated by one blank. Where consecutive locations beginning at aaa are to be patched, successive patches may be given in the form of comma-separated fields. The patch fields (bbbb,cccc,...) may contain up to 12 octal characters which will be right justified and stored in the respective locations in memory.

DONE — indicates that the user is finished with the patch function and wishes to select a new function

Response: X

This response indicates that the user wishes to display the contents of all working registers. These include the AQ, the E, and all index registers.

The format of this printout is to be supplied.

Response: D

This response indicates that the user wishes to delete this particular call to TDS by storing a NOP instruction over the XED TDS instruction.

Response: R

This response causes control to return to the subsystem at the location following the XED TDS instruction.

Response: Rxxxx

This causes a special return to the subsystem at location xxxx, where xxxx is an octal address within the subsystem.

Error Indications and Messages :

1. When illegal input is typed in response to FUNCTION?, the same request reappears.

2. ILLEGAL INPUT - RETYPE - is typed in response to illegal input typed as parameters to the Snap and Patch functions.

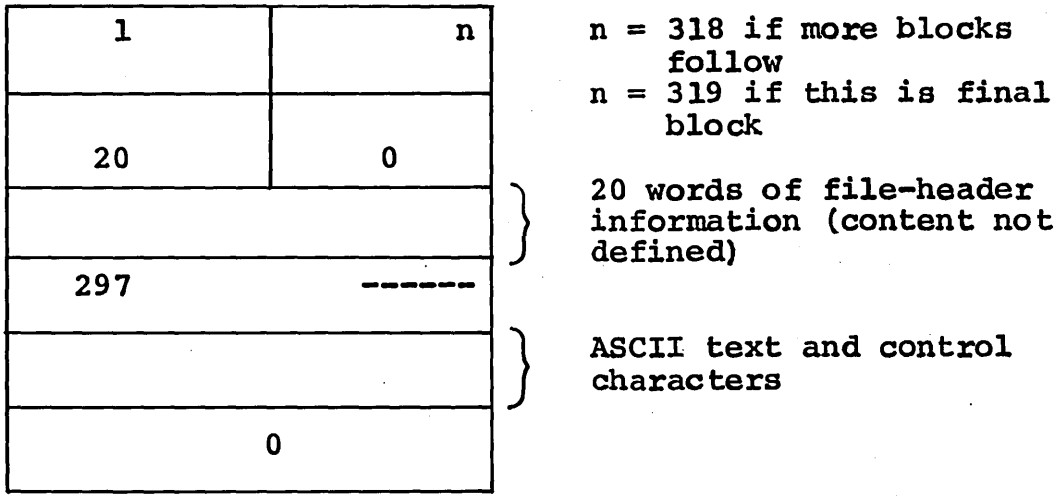
6. 600TSS FILE FORMATS

SOURCE FILE

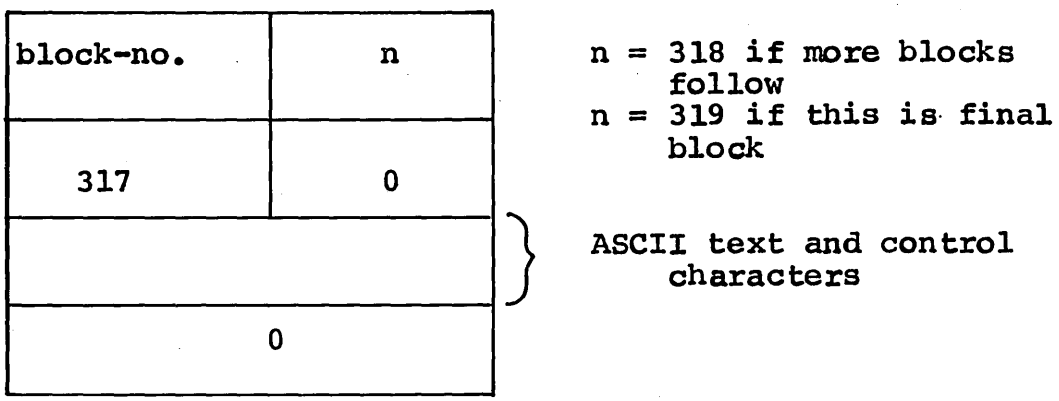
The standardization of source-text files allows more than one system to process these files. For example, using a standard file format allows EDITOR to operate on BASIC text. All text files are maintained as character strings in ASCII format. They are linked files that contain block and logical record control words that allow the files to be accessed by GEFRC. The standard source file used by GE-released subsystems is named *SRC.

Format:

Initial 320-word block



Second and succeeding 320-word blocks



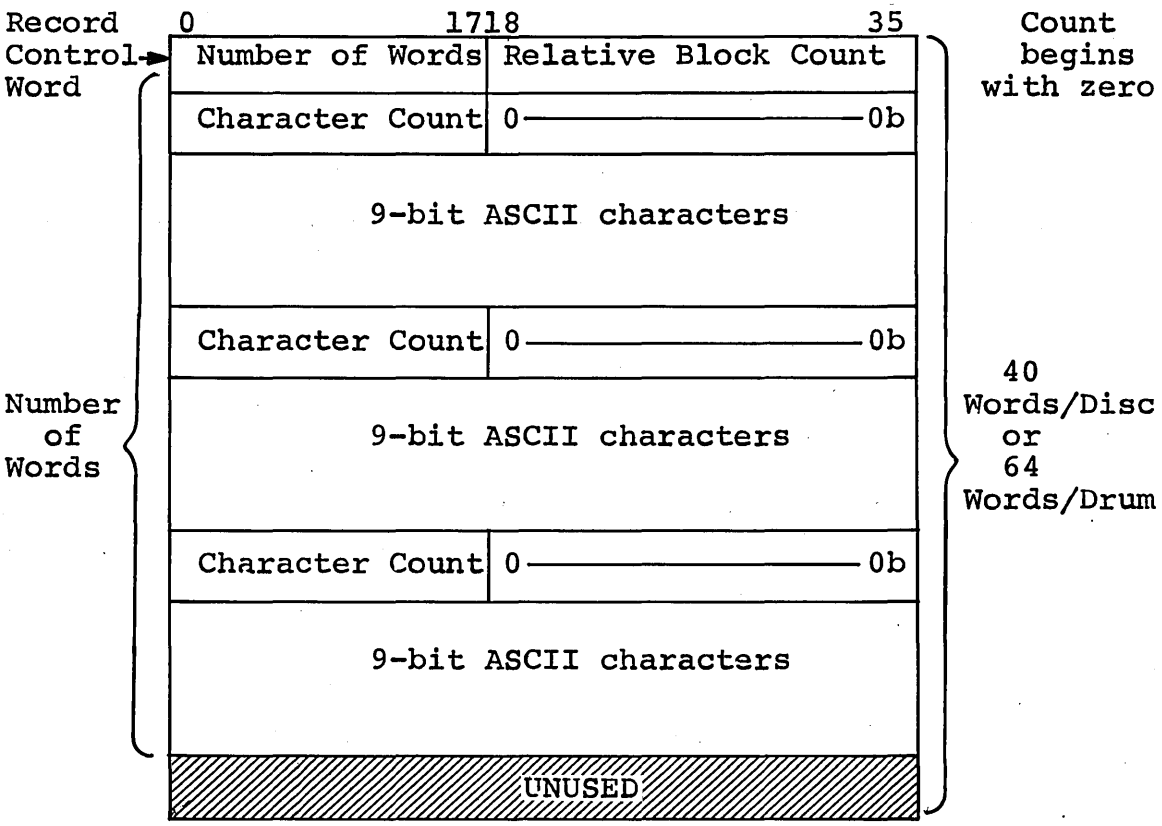
The ASCII consists of packed strings of 9-bit characters. The first character of each string is interpreted as a character count, in binary, of the number of characters following in the string. The next character following the string is another character count of the succeeding string. A character count of zero followed by a character of 036_8 indicates the end of data in a block. The value of n (318 or 319) in the block control word (word 1 of the block) indicates whether or not succeeding blocks follow. A character string does not extend from one block to another.

SY** FILE

SY** is a random file used by the system to accumulate input from a terminal while in the build mode (see Chapter 3). The file content is written by the Line Service module of 600TSS.

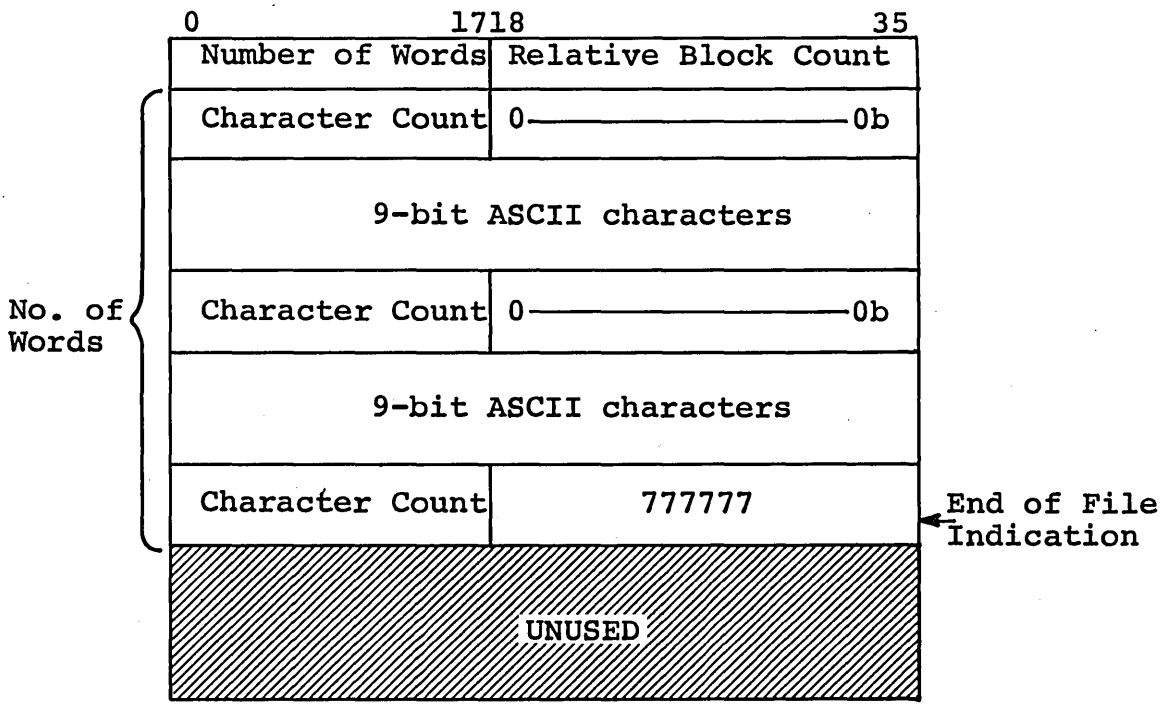
Format:

All non-empty records except the last



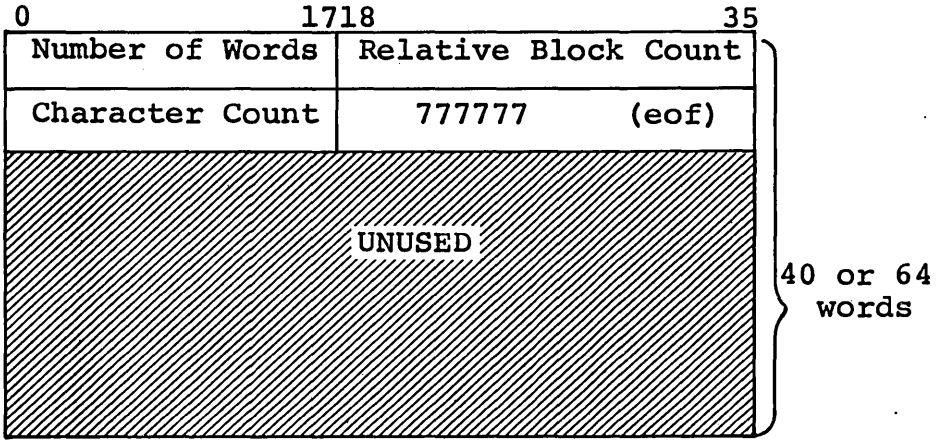
b (bit 35) = 1 if the string contains 80 characters with no carriage return; = 0 otherwise

Final (non-empty) record



b (bit 35) = 1 if the string contains 80 characters with no carriage return; = 0 otherwise

Empty record (a command word was the first line in input buffer)

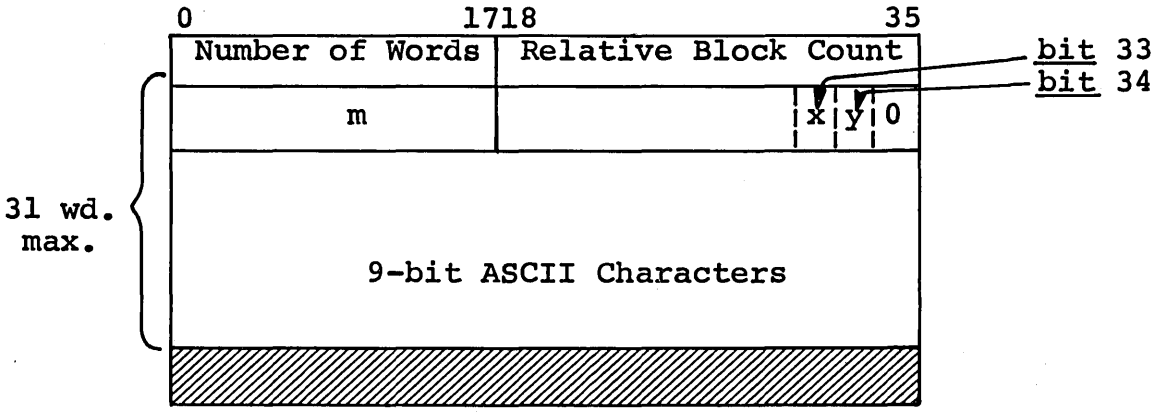


NOTE: An empty record may or may not be the first record in file.

TAP* FORMAT

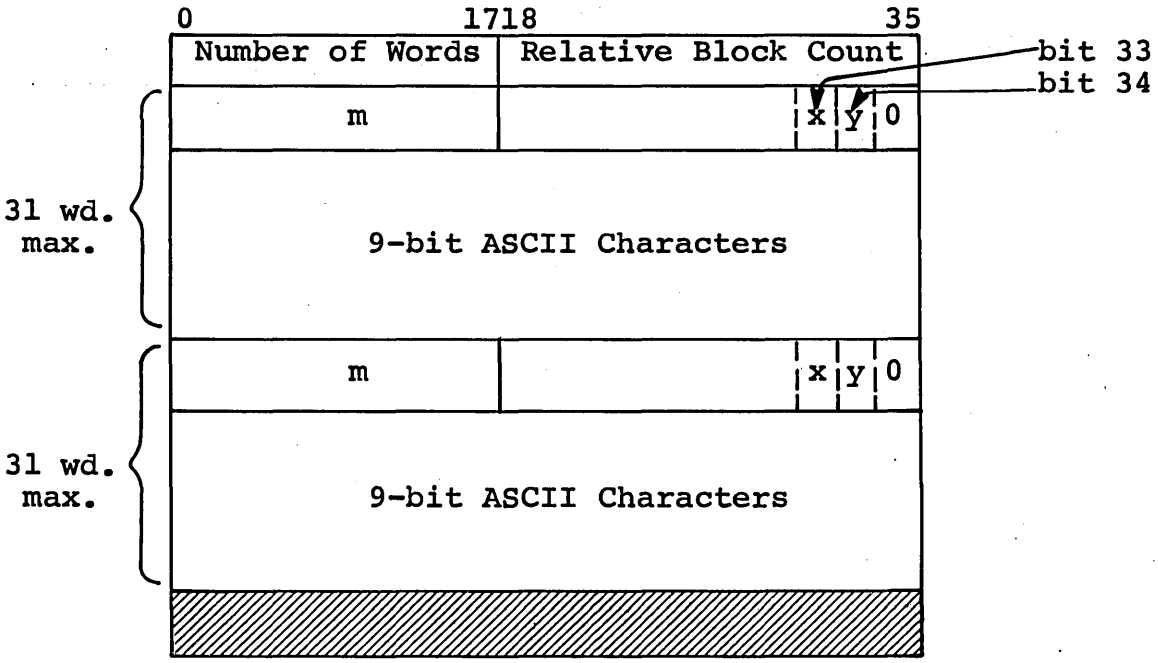
TAP* is the punched paper tape input-collector file which contains the unedited PPT input. It is a random file, with a maximum of 2 links.

1) Format from Disc - 40 words/block:



m = character count of input data block (≤120) - may be zero
x = 1 if timing error occurred
y = 1 if last block

2) Format from Drum - 64 words/block:



m = character count of input data block (≤120) - may be zero
x = 1 if timing error occurred
y = 1 if last block

7. GE-SUPPLIED SUBSYSTEMS

INTRODUCTION

A significant part of the GE-600 Time-Sharing System is implemented in subsystem form, rather than as part of the 600TSS Executive. These subsystems are, in general, available for use as part of a user-designed system within 600TSS. A brief functional description of each subsystem follows.

SUBSYSTEM DESCRIPTIONS

BSED

Title: Basic-Edit

Purpose:

Sort into ascending numerical sequence any lines (in BASIC-statement format) on input-collector (SY**) file and, if a source (*SRC) file exists, merges these new lines into that file without disturbing its numerical sequence. New line numbers which correspond to existing lines on the source file replace or delete the existing lines.

Usage

BSED is not selected directly by a user's response to the "SYSTEM?" request, but it is called as a result of commands issued while in the BASIC system. When one of the commands RUN, LIST, DONE, or SAVE is entered at a terminal after the user has entered new source language statements, the BSED subsystem is called. If no new source language lines have been entered since the last command was given to the system, BSED will not be called as the source file is assumed to be in proper order.

Error Comments:

IMPROPER FILE FORMAT - Source file is not in 600TSS ASCII format. The format of the file should be individual lines of source language appearing as an ASCII string with the character count of the string appearing in its first character position.

System Errors (Code numbers are used by the HELP subsystem.)

101 Read error on input-collector file. Last set of updates has been lost due to system problems and must be re-entered.

102 Read error on source file. Call in a new copy of the source file using the OLD command and retry.

103 Write error on updated source file. Call in a new copy of the source file using the OLD command and retry. All updates have been lost.

HELP

Title: HELP!

Purpose:

Give further explanations and user procedures to be followed in the event of error codes returned by other 600TSS subsystems. The error codes are the three-digit numbers that precede brief system-error messages.

Usage:

Selected by the terminal user in response to the question "SYSTEM?", when he types "HELP". The HELP subsystem's next request is "PLEASE ENTER ERROR CODE--". Here the user types the three-digit error code for which more information is being requested. An explanation of the error and possible recovery procedures are then typed at the terminal.

Error Comments:

ERROR CODE MUST BE NUMERIC - Self explanatory. User is requested to retype the code number.

ERROR CODE EXCEEDS 3 CHARACTERS - Self explanatory. User is requested to retype the code number.

ERROR CODE NOT ASSIGNED - Check code in message against that which was typed to HELP.

LIST

Title: List BASIC Source File

Purpose:

List at the terminal all or any part of a source file, by line number.

Usage:

Called by the BASIC subsystem when the user enters one of the forms of the LIST command while in build-mode.

Forms of the LIST command:

LIST	types entire file, line-by-line at the terminal
LIST n	where n is a line number, begins typing with line n and continues line-by-line to the end of file
LIST n,m	where n and m are line numbers, begins typing with line n and types all lines up to and including line m. When n is greater than m no lines are typed, and after typing "READY" the system is ready for a new command
LIST,m	where m is a line number, begins typing with the first line in the file and continues up to and including line m

Error Comments:

Except for read/write system errors which require calling up a new copy of the file, there are no error conditions.

LODX

Title: Load Subsystem-Program

Purpose:

Load a subsystem program from a system-loadable-format permanent file into the 600TSS system for checkout purposes. Also, when requested, apply octal patches to the subsystem after it is loaded into memory.

Usage:

After the user selects LODX in response to the "SYSTEM?" message, he is asked for "FILENAME?". Two comma-separated fields of information may be given:

Field 1 - only required field. The 1 to 8 character name of the file on which the program has been written.

Field 2 - optional patch indicator (*). Typing an asterisk after the comma separating the fields indicates to LODX that octal patches follow.

The catalog block(s) of the file is read to verify program name, if any, and to calculate the program size from the DCW(s). The required memory area is obtained by a DRL ADDMEM. The program is then loaded into this area, and the memory required by LODX is released when loading has completed.

Octal Corrections:

If the asterisk character was typed as part of the response to "FILENAME?", the program is loaded and then a carriage return, line feed, and question mark are typed at the terminal. The user then types the corrections in one of the following forms:

1. address correction
2. address correction 1, correction 2,....

All addresses and corrections must be octal and the addresses are relative to the load map produced when the file was written. The second form describes how sequential corrections may be typed beginning at the address indicated. In this case, the corrections are comma separated and as many may be typed as will fit on the line. Leading zero's need not be typed.

Diagnostic Messages of LODX System:

TYPE n FAULT AT xxxxxx	n is a type number and xxxxxx is the address where it occurred in relation to LODX BAR n=1 Illegal Op Code n=2 Memory Fault n=3 Fault Tag n=4 Divide Check
SYSTEM ERROR - RETYPE	system error occurred while reading Filename information; retype entire line and if same error recurs, the user is returned to system level
FILE UNAVAILABLE - CALL BACK LATER	specified file is currently busy and the user may try again later
RETYPE - NAME NOT KNOWN	file name does not appear in User Master Catalog or in Available File Table
ERROR IN ACCESSING FILE - STATUS x	x is the status code returned by the File System when accessing the requested file was attempted
CANNOT ACCESS FILE - STATUS UNKNOWN	illegal status returned by file system when file was accessed

PROGRAM NOT ABSOLUTE	program saved on file was not in correct format and relocation bits are not acceptable
EXCESSIVE ERRORS IN READING FILE	after four attempts are made to read a file where errors are encountered, suggest user rewrite the file
CATALOG NOT IN PROPER FORMAT	file must be random and in system loadable format
ENTRY LOC 100	program entry address must be greater than 100 to allow for TSS prefix
ILLEGAL FORMAT - RETYPE	occurs while user is typing patches; therefore, check description of patch format
ADDRESS OUT OF BOUNDS	patch location exceeds size of program loaded

LOGOFF

Title: Logoff

Purpose:

Terminate a user's session with TSS. The cost of the session is calculated and a message issued for the user's information concerning the cost of this session and his accumulated charges to date.

Usage:

Called either by the command BYE, if in the build-mode of the BASIC subsystem, or as a system selection at system level:

```
* BYE      (or)
SYSTEM?   LOGOFF  (or)
SYSTEM?   BYE
```

The charges for the user's session are calculated and the following message issued before the terminal is disconnected.

```
**RESOURCES USED $xx.xx, USED TO DATE $xx.xx=yy%
**TIME SHARING OFF AT zz.zzz ON mm/dd/yy.
```

NEW

Title: NEW

Purpose:

Determines if this is a legitimate user after requesting ID and PASSWORD, and, if so, enters him into the system. If not, a comment and a disconnect are issued.

Usage

Enabled automatically when the user dials into 600TSS.

NOTE: NEW cannot be requested as a system selection.

A message is issued identifying the system:

```
THIS IS THE GE-600 T/S SYSTEM ON date AT time CHANNEL xxxx
```

If an all-points message exists in the system, it appears next; it is information sent to each current user and each new user entering the system.

Next the request "USER ID -" is issued and NEW waits for the response. The user responds with his ID of no more than 12 characters by which he is known to the system. Then the request "PASSWORD--" and a mask on which to type the password (no more than 12 characters) appears. If NEW determines that the user is permitted in the 600TSS system, acknowledged by the given ID, password, and sufficient resources to pay for the session, the user is entered into the system and NEW has completed its function.

Error Comments:

The user is allowed two chances to type his ID and his PASSWORD correctly. After the first incorrect entry a message "ILLEGAL ID-RETYPE" or "ILLEGAL PASSWORD--RETYPE" is issued. If typed incorrectly a second time, a comment "ILLEGAL ID" or "ILLEGAL PASSWORD" is issued followed by a disconnect.

If the user's resources are overdrawn by more than ten percent, a message and a disconnect will be issued.

"RESOURCES EXHAUSTED. CANNOT ACCEPT YOU."

If the user's resources are overdrawn but by less than ten percent, a warning message is issued.

"RESOURCES OVERDRAWN n%"

NEWUSER

Title: New User

Purpose:

Allows a user to terminate a session with 600TSS and a new user to enter the system at the same terminal, without disconnection of that terminal. The advantage for the new user is that he is assured of a line into 600TSS.

Usage:

Called as a system selection:

SYSTEM? NEWUSER

The charges for the user's session are calculated and his accounting file updated. The following terminal message is issued for his information.

RESOURCES USED $xx.xx, USED TO DATE $xx.xx=yy%

The paper is spaced forward and the new user is now ready to be logged into the system. NEWUSER is terminated and the NEW subsystem is initiated.

OLDN

Title: Old-New Sequence

Purpose

Allow the user to specify what file (if any) he wants to work with as his source file.

Usage:

Called whenever a user selects a system that requires a source file, such as BASIC or EDITOR. The first response from 600TSS after calling one of these systems comes from the Old-New program:

OLD or NEW-

The legitimate responses to this question are:

NEW--the OLDN program will see if a source file has been defined (opened) and if it has, it will initialize it. If a source file has not yet been defined, it will be defined and initialized. The next response will be: READY FOR INPUT.

SAME--this response allows a user to keep his source file when changing from one system to another. The OLDN program will verify that the user already has a source file. The next response will be: READY FOR INPUT.

OLD XXXX (where XXXX is the file name)

If XXXX is not given the user will be asked "OLD NAME". The file name must consist of from one to eight alphabetics, numerics, dashes and/or periods. Blanks are edited out,i.e., FØIØLØE is equivalent to FILE. OLDN will see if the named file has been previously accessed (opened), and if not, it will attempt to access it. If this is not successful, an appropriate error message will be printed. After accessing the named file, OLDN will make sure that the user has a source file and will write the named OLD file to it. The next response from 600TSS will be: READY FOR INPUT.

LIB XXXX (where XXXX is the file name)

If XXXX is not given the user will be asked "OLD NAME". The OLDN routine will access the specified file from the Library catalog and write it to the user's source file. LIB is a special function to allow users to gain access to "read only" files which are maintained by the computer site. These would include information files, demonstration programs, etc. The file description is LIBRARY/file name. After this file is accessed the next response from TSS will be: READY FOR INPUT.

Note 1: In the BASIC system the OLDN subsystem will also be called when the user gives the command words OLD, LIB, or NEW. When "OLD XXXX" or "LIB XXXX" is given, the execution will be as described above. When "NEW" is given OLDN will initialize the source file and the next response will be: READY FOR INPUT.

Note 2: OLD files that are accessed through the OLDN subsystem alone (i.e., without previous use of the ACCESS subsystem) must be of the quick-access type. OLDN requests all permissions for the user on quick-access files.

Other responses from the Old-New program are:

PLEASE RESPOND WITH "OLD", "NEW", "SAME" or "LIB".

The user has responded to the OLD OR NEW question with something other than OLD, NEW, SAME, or LIB.

NO FILE NAME GIVEN

The user has not typed any non-blank characters in response to OLD NAME.

FILE NAME MUST BE 8 CHARACTERS OR LESS

A file name of greater than 8 non-blank characters was given in response to OLD NAME.

Error Comments:

130 - YOU DO NOT HAVE A "SAME" FILE.

At this session at the terminal, the user has not built a source file.

107 - UNABLE TO ACCESS SOURCE FILE - STATUS-YY

When a user first selects a system, a source file is defined and accessed (opened) for him. In this case, OLDN has been unable to define this file. The status will explain why:

yy=03 user's available-file-table is full.

yy=04,06 file space for the user's source file is not available at this time.

108 - UNABLE TO INITIALIZE SOURCE FILE

An I/O error has been encountered while attempting to initialize the user's source file.

114 - UNABLE TO READ "OLD" FILE

I/O errors have been encountered while attempting to read user's OLD file.

109 - NO DATA ON OLD FILE

OLDN has attempted to read an OLD file which contains no data. The file specified in response to OLD NAME must contain readable data.

110 - UNABLE TO WRITE SOURCE FILE

I/O errors have been encountered while attempting to write the user's OLD file to the user's source file.

131 - UNABLE TO ACCESS FILE - xxxx-STATUS-yy

OLDN has been unable to access the file xxxx. The status will explain why:

yy=02 non-recoverable I/O error has been encountered while trying to access file xxxx

yy=03 user does not possess the proper permissions on the file. Presumably, the file was opened previously through the ACCESS subsystem but READ permission was not requested.

yy=04 file xxxx is busy

yy=05 file xxxx does not exist in the user's master catalog

yy=36 user's available File Table is full

yy=40 system is temporarily loaded

SAVE

Title: Save or Purge Files

Purpose:

Allow the user to save data, programs, etc. on a permanent file for later retrieval, and to delete permanent files.

Usage:

Called whenever a user uses the SAVE or PURGE command word followed by a file name of from one to eight non-blank characters. The non-blank characters can consist of alphabetics, numerics, minus sign and period. All blanks are edited out, i.e., FØIØLØE is equivalent to FILE.

SAVE XXXX (where XXXX is a file name)

The SAVE program will first see if the named file has already been accessed. If it has, the user's source file is written to it. If it has not been previously accessed, SAVE attempts to access it. If successful, the source file is then written to it. If not successful because the named file does not exist, the SAVE routine has the named file created and accessed, and then writes the source file to it.

If the file has been successfully saved, the user will get the message:

DATA SAVED - xxxx (where xxxx is the file name)

If for some reason the SAVE cannot be accomplished, an appropriate error message is printed.

NOTE: Files accessed by the SAVE subsystem are accessed with all permissions requested for the user. Files created by the SAVE program are created with a general READ permission only. The created file emanates directly from the user's master catalog.

PURGE xxxx (where xxxx is a file name)

File xxxx will be deleted and the file space made available again to the user.

If the file was successfully purged, the user will get the message:

FILE PURGED - xxxx

If for some reason the purge could not be done, an appropriate error message will be printed.

Other responses from the SAVE program:

DATA NOT SAVED or NO PURGE DONE

Three successive procedural mistakes have been made by the user.

FILE NAME?

The user has specified an invalid file name. This message will be preceded by one of the following:

FILE NAME CONTAINS INVALID CHARACTER -x

or

FILE NAME MUST BE 8 CHARACTERS OR LESS

(or if blanks only were typed, just the File-Name message is given)

Error Comments:

131 - UNABLE TO ACCESS FILE - xxxx - STATUS-yy

The SAVE program has been unable to access (open) the file xxxx. The status will explain why:

yy=02 unrecoverable I/O errors were encountered

yy=03 user does not possess the proper permissions on file xxxx. Presumably the ACCESS subsystem was used to open the file, but WRITE permission was not requested.

yy=04 file xxxx is busy

yy=40 system is temporarily loaded

yy=36 user's available-file-table is full

128 - UNABLE TO CREATE FILE -- STATUS-yy

The file named in the SAVE command did not exist and the SAVE program was unable to create it. The status will explain why:

yy=02 unrecoverable I/O errors were encountered

yy=06,10 system is temporarily loaded

yy=13 user has no file space left

128 - UNABLE TO READ SOURCE FILE. CAN'T DO SAVE.

Unrecoverable I/O errors have been encountered attempting to read user's source file.

129 - UNABLE TO WRITE "SAVE" FILE.

Unrecoverable I/O errors have been encountered attempting to write the file named in the SAVE command.

140 - UNABLE TO PURGE FILE -- xxxx - STATUS yy

The SAVPRG program has been unable to purge file xxxx. The status will explain why:

yy=02 unrecoverable I/O errors were encountered

yy=03 user does not possess PURGE permission or file xxxx

yy=05 file xxxx does not exist in the user's master catalog

STATUS

Title: Status Report

Purpose:

Produce a report on the user's usage of 600TSS facilities during his current session at the terminal.

Usage:

Print the following report in response to SYSTEM ? type STATUS:

USER STATUS ON date AT time.

LOGON time, PROC USED x.xxx SEC., n DISC I/O, n CHAR KEY I/O

OPEN FILES: _____ _____ _____ _____

APPENDIX A

SYSTEM MACROS

The Time-Sharing System modules and subsystems require many definitions of the communication region, UST contents, DRL's, etc. These have been formulated into a set of macros. They are loaded by

LODM .G3TSn *

The macros and their use are as follows:

.TSCOM — This macro contains all the common communication definitions and the macro call .SSUST for the UST equivalences. It would not normally be used in a subsystem.

.SSUST — Subsystems would use this macro to provide the UST and DRL equivalences.

.SSDRL — Normal subsystem should use this macro to obtain the DRL definitions.

PRNTTY — n,(comment n chars. long),K.

This macro is used to print the error comment from a subsystem. K (optional) is a pointer to a word containing control characters to be affixed to the end of the line (CR,LF,NULL, etc.).

* The n depends upon the catalog name that is assigned in the latest System Development Letter.

APPENDIX B

OCTAL/ASCII CONVERSION EQUIVALENTS

OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.	OCTAL NUMB.	ASCII CHAR.
000	NULL	040	SP	100	@	140	GRA
001	SOH	041	EXP	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	ECT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	AXK	046	&	106	F	146	f
007	BELL	047	'	107	G	147	g
010	BSP	050	(	110	H	150	h
011	HT	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FFD	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	LBK	173	LBR
034	FS	074	LTN	134	RSL	174	VTL
035	GS	075	=	135	RBK	175	RBR
036	RS	076	GTN	136	CFX	176	NOT
037	US	077	?	137	_	177	DEL

APPENDIX C

COMMUNICATIONS CONTROL

ACK	Acknowledgement
CAN	Cancel
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
DLE	Data Link Escape
EM	End of Medium
ENQ	Enquiry
EOT	End of Transmission
ESC	Escape (Alternate Mode)
ETB	End of Transmission Bloc
ETX	End of Text
NAK	Negative Acknowledgement
SOH	Start of Heading
STX	Start of Text
SUB	Substitute Character
SYN	Synchronous Idle

FORM EFFECTORS

BSP	Backspace
CR	Carriage Return
FFD	Form Feed
HT	Horizontal Tabulation
LF	Line Feed
VT	Vertical Tabulation

ITEM SEPARATORS

FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator

TEXT MATERIAL

BELL	Bell, or other attention signal
CFX	Circumflex
DEL	Delete (Rubout)
EXP	Exclamation Point
GRA	Grave (reversed) Accent
GTN	Greater Than: Mathematical symbol
LBK	Left Bracket
LBR	Left Brace
LTN	Less Than: Mathematical symbol
NOT	Not: Mathematical symbol
NULL	Null
RBK	Right Bracket
RBR	Right Brace
RSL	Reverse Slash
SI	Shift In
SO	Shift Out
SP	Space
VTL	Vertical Line

APPENDIX D

GE-625/635 STANDARD CHARACTER SET

Standard Character Set	GE-Internal Machine Code	Octal Code	Hollerith Card Code	Standard Character Set	GE-Internal Machine Code	Octal Code	Hollerith Card Code
0	00 0000	00	0	↑	10 0000	40	11-0
1	00 0001	01	1	J	10 0001	41	11-1
2	00 0010	02	2	K	10 0010	42	11-2
3	00 0011	03	3	L	10 0011	43	11-3
4	00 0100	04	4	M	10 0100	44	11-4
5	00 0101	05	5	N	10 0101	45	11-5
6	00 0110	06	6	O	10 0110	46	11-6
7	00 0111	07	7	P	10 0111	47	11-7
8	00 1000	10	8	Q	10 1000	50	11-8
9	00 1001	11	9	R	10 1001	51	11-9
⌈	00 1010	12	2-8	-	10 1010	52	11
#	00 1011	13	3-8	$	10 1011	53	11-3-8
@	00 1100	14	4-8	*	10 1100	54	11-4-8
:	00 1101	15	5-8	)	10 1101	55	11-5-8
>	00 1110	16	6-8	;	10 1110	56	11-6-8
?	00 1111	17	7-8	'	10 1111	57	11-7-8
♭	01 0000	20	(blank)	+	11 0000	60	12-0
A	01 0001	21	12-1	/	11 0001	61	0-1
B	01 0010	22	12-2	S	11 0010	62	0-2
C	01 0011	23	12-3	T	11 0011	63	0-3
D	01 0100	24	12-4	U	11 0100	64	0-4
E	01 0101	25	12-5	V	11 0101	65	0-5
F	01 0110	26	12-6	W	11 0110	66	0-6
G	01 0111	27	12-7	X	11 0111	67	0-7
H	01 1000	30	12-8	Y	11 1000	70	0-8
I	01 1001	31	12-9	Z	11 1001	71	0-9
&	01 1010	32	12	←	11 1010	72	0-2-8
.	01 1011	33	12-3-8	,	11 1011	73	0-3-8
]	01 1100	34	12-4-8	%	11 1100	74	0-4-8
(	01 1101	35	12-5-8	=	11 1101	75	0-5-8
<	01 1110	36	12-6-8	"	11 1110	76	0-6-8
\	01 1111	37	12-7-8	!	11 1111	77	0-7-8

INDEX

Please cut along this line

DOCUMENT REVIEW SHEET

TITLE: GE-625/635 GECOS-III Time-Sharing System Prog Reference Manual

CPB #: 1514

From:

Name: _____

Position: _____

Address: _____

_____

Comments concerning this publication are solicited for use in improving future editions. Please provide any recommended additions, deletions, corrections, or other information you deem necessary for improving this manual. The following space is provided for your comments.

COMMENTS: _____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

Fold on two lines shown on reverse side, staple, and mail.

STAPLE

STAPLE

FOLD

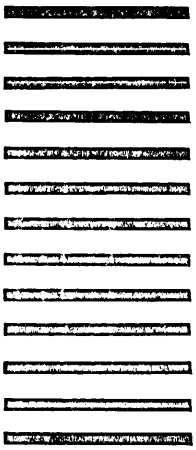
FIRST CLASS
PERMIT, No. 4332
PHOENIX, ARIZONA

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

GENERAL ELECTRIC COMPANY
PROCESSOR EQUIPMENT DEPARTMENT
13430 NORTH BLACK CANYON HIGHWAY
PHOENIX, ARIZONA 85029

ATTENTION: DOCUMENTATION B-90



FOLD

GENERAL ELECTRIC INFORMATION SYSTEMS DIVISION COMPUTER EQUIPMENT DEPARTMENT	GE-600 SERIES TECHNICAL INFORMATION BULLETIN	DATE July 1968 NO. 600-213
SUBJECT: Expanded Debugging Facility (Time-Sharing System)		REF. CPB-1514

This TIB describes the capabilities of the Debugging Facility for 600TSS subsystem development, as presently improved and expanded. Also included is a minor technical modification to the subsystem temporary-placement deck setup.

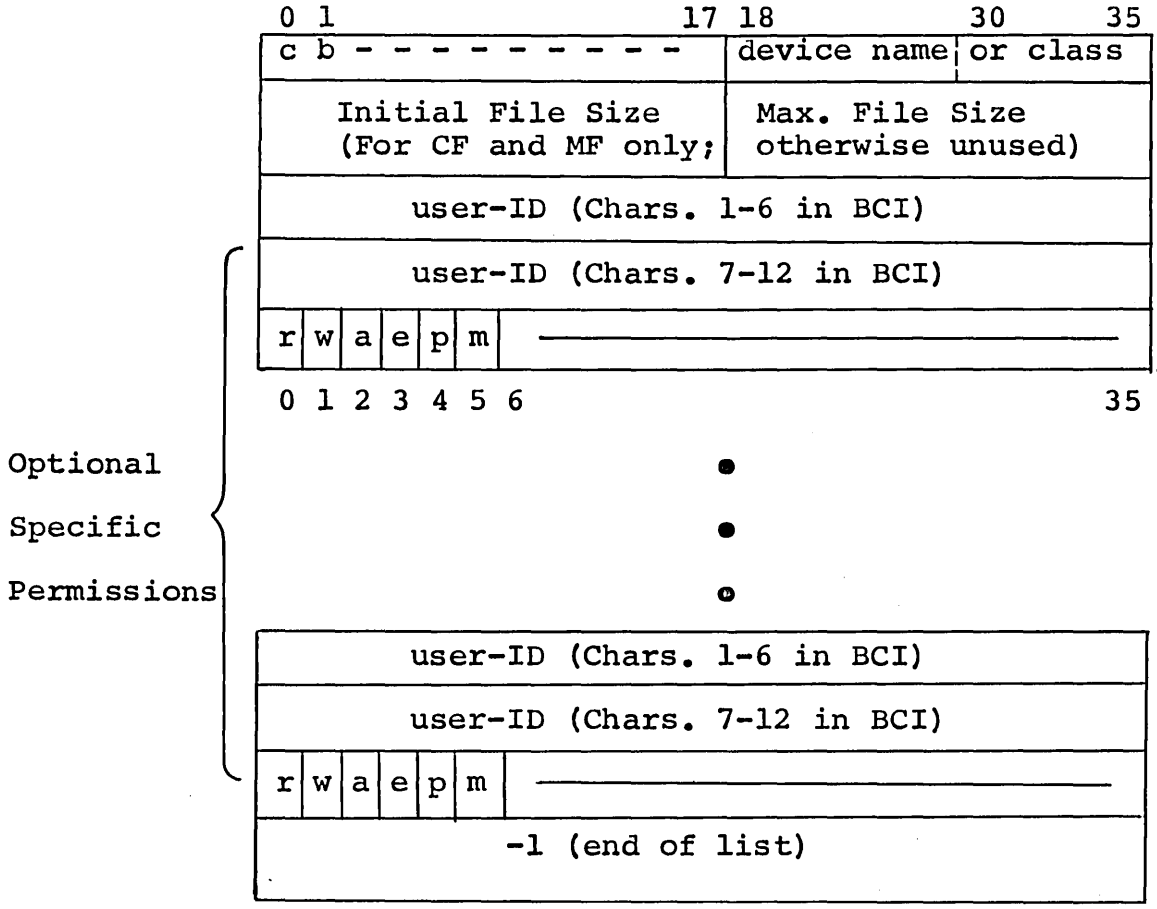
Please change the pages in your GE-625/635 GECOS III Time-Sharing System Programming Reference Manual, CPB-1514, as follows:

Remove	Replace	Insert
27-30	27-30	
59-64	59-64	
		64.1

This Technical Information is the first TIB to be issued for this edition of the manual.

It is suggested that you add this page to the front of your manual to show that this TIB has been entered.

(4) options



If corresponding bit is set:

c = Contiguous allocation desired
(CREATE FILE only)
r = Specific Read permission
w = Specific Write permission
a = Specific Append permission
e = Specific Execute permission
p = Specific Purge permission
m = Specific Modify permission
b = Random File type
(CREATE FILE only)

and device name or class is defined as:

Specific (named) device, in BCI -
e.g., DS1, where "DS1" would have been assigned as a specific device name in the installation's GECOS-startup deck.

Class of device (in bits 30-35) -

00 = DSU270 (large disc)
01 = DSU200 (standard disc)
03 = MDU200 (UNIVAC drum)
04 = MDU300 (Fairchild drum)

or where

-1 (bits 18-35) denotes the file with the most available space.

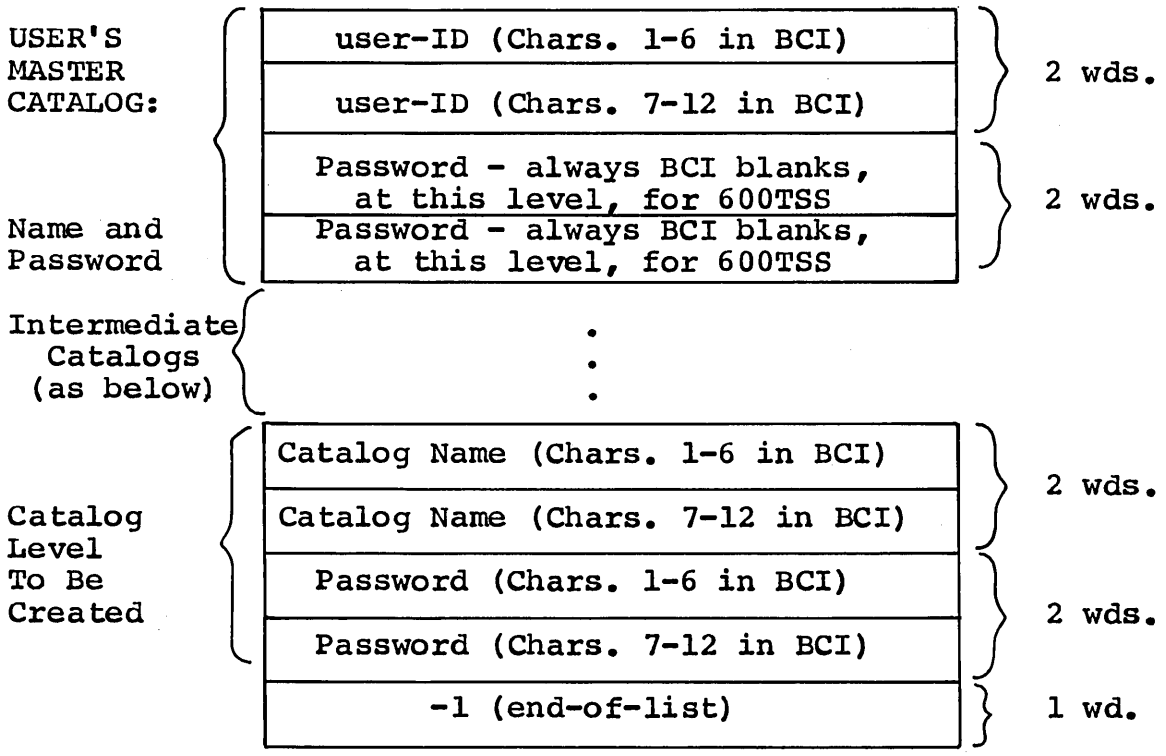
Create-Catalog Function

```
DRL     FILACT      [30]
ZERO    0,L(arglist)
ZERO    2,L(buffer)
```

where

```
arglist   ZERO   L(status-return),0
          ZERO   L(cat/file desc),L(permissions)

          ZERO   L(options)
```

cat/file desc



(1) All names and passwords are left-justified with trailing blanks.

(2) -1 in place of the user's-master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

This FILACT function, identified by the function-number "2", creates the specified new catalog at the level indicated. All existing intermediate catalogs must be specified in the cat/file desc table (i.e., the complete catalog string).

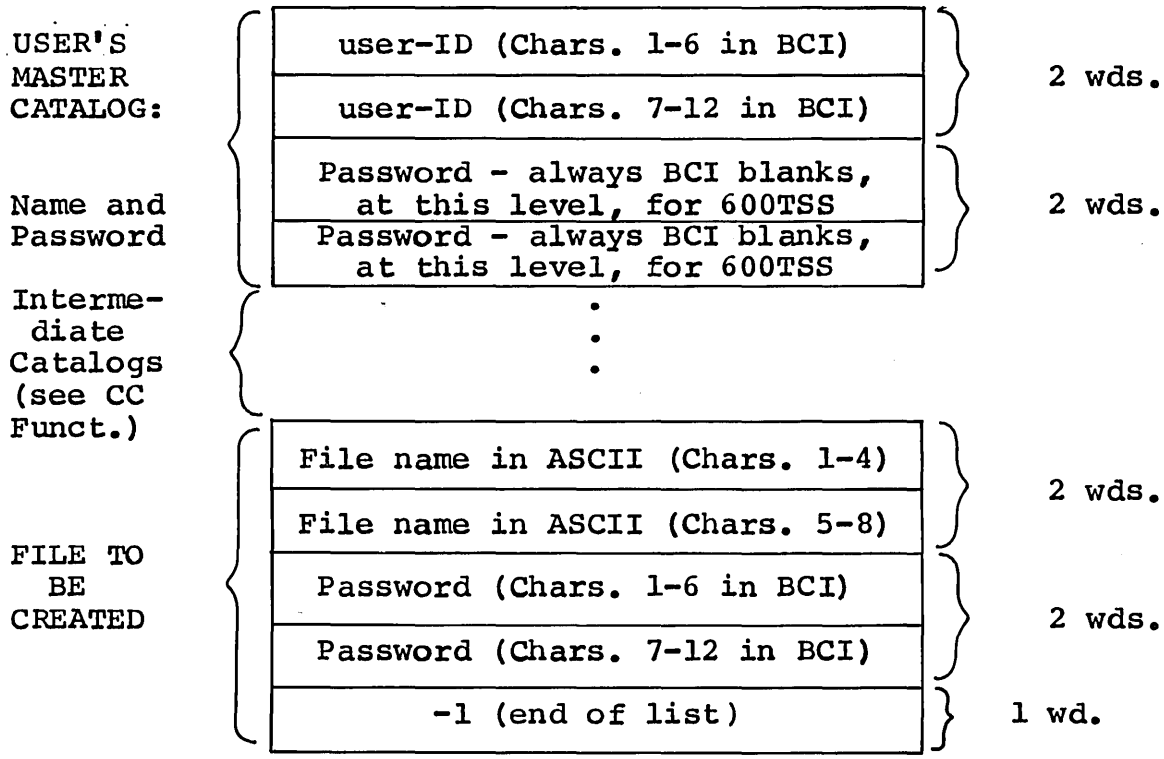
Create-File Function

```
DRL     FILACT
ZERO    0,L(arglist)
ZERO    3,L(buffer)
```

where

```
arglist  ZERO  L(status return),0
         ZERO  L(cat/file desc),L(permissions)
         ZERO  L(options)
```

cat/file desc



(1) All names and passwords are left-justified with trailing blanks.

(2) All entries are in BCI, except for the file name.

(3) -1 in place of the user's master-catalog name indicates that the user-ID of the current terminal user is to be filled in by the derail processor.

The Create-File function creates a permanent-file description from the information specified in both the cat/file desc and options parameters and will acquire the necessary file space. The file name is not entered in the user's AFT (see "Access-File Function").

Access-File Function

```
DRL     FILACT
ZERO    L(alternate name),L(arglist)
ZERO    4,L(buffer)
```

where

arglist ZERO L(status-return),1 for random/0 for linked file*

ZERO L(cat/file desc),L(permissions)

* note-if this field is nonzero the file will be accessed as a random file regardless of how it is defined. If the field is zero the file will be accessed according to how it is defined.

alternate name

Alternate name in ASCII, or all
Zeros if no alternate naming desired

} 2 wds.

This two-word entry is used when a file is to be accessed by a name other than that by which it was created. That is, a file created in the batch environment with a name of more than 8 characters, or a file whose name is the same as one already in the user's AFT.

NOTE: When an alternate name is used, the defined file name in the cat/file description must be in BCI and the alternate name in ASCII.

Placement

The subsystem program is written and assembled the same as for permanent placement. (The program descriptor need not be assembled; it will not be referenced. Do not modify TSTRT.) The following steps are then performed:

- Create a random permanent file using ACCESS. The subsystem is stored and referenced from this file.

- Submit the program deck(s) as a GELOAD activity to GECOS III with the following deck setup:

```
$OPTION SAVE/prog-name, NOGO
    •
      other control cards
    •
    •
      subsystem deck(s)
    •
$EXECUTE
$LIMITS
$PRMFL H*,WRITE,R, cat-name/filename
$ENDJOB
```

Loading the Subsystem

After submitting the decks as a GELOAD activity, the user may have the subsystem loaded and start check-out from a terminal.

In response to SYSTEM?, specify LODX. For FILENAME?, give the filename specified on the $PRMFL card. If patching is desired, type an asterisk as the second field preceded by a comma.

Examples:

```
FILENAME?  A123       To load and execute

FILENAME?  A123,*     To load, patch, and execute
```

Octal Patching

If the asterisk was typed as part of the response to FILENAME?, the program is loaded and a carriage return, line feed, and question mark are given. The user then may type patches in one of the following forms:

1. ?address patch
2. ?address patch1,patch2,...

In both forms, the address and patch must be separated by a single blank. All addresses and patches must be in octal, and the addresses are relative to the load map produced when the file was written. In the second type, sequential patches may be given beginning at the specified address. In this form, the patches are comma-separated and as many may be given as will fit on the line. Leading zeros need not be typed.

Typing DONE indicates that patching is completed:

```
? DONE
```

The subsystem program is then executed.

DEBUGGING FACILITY

During Checkout, a standard debugging subroutine is provided for inclusion in a subsystem that is placed in the system.

Title: Terminal Debug Subroutine

Symbolic Name: TDS

Purpose:

In checking out a 600TSS subsystem, TDS allows the user to gain control at selected locations within the subsystem. When TDS is in control, the user may display and/or patch selected areas of the subsystem, display and/or modify registers, and either return to the subsystem normally or to a specified location within the subsystem. The user may add or delete breakpoint locations during operation of the subsystem.

Usage During Subsystem Preparation:

TDS may be entered from the subsystem in either one of two ways:

1) At each location where the user is to gain control, the instruction:

```
XED  TDS
```

is inserted into and assembled with the subsystem.

2) Once the user has control he can add breakpoint locations at the terminal during the debugging process. (At least one breakpoint -- item 1 -- must have been provided.)

In either case, the subsystem to be checked out must contain a SYMREF to TDS. Then when the subsystem is submitted as a GELOAD activity (to be loaded later by LODX), a binary object deck of the TDS subroutine must be included. The installation may place this object deck on the Subroutine Library.

Usage During Subsystem Checkout:

The user calls the subsystem to be checked out by the LODX procedure. When any of the locations at which the user has placed a breakpoint (XED TDS instruction) are encountered, the following message will appear at the user's terminal:

```
xxxxxx FUNCTION?
```

where xxxxxx is the octal address of the breakpoint.

In the following messages, the requests and their respective results are listed. Note that "absolute" value refers to an address relative to subsystem zero; and, to initiate transmission, all requests must be followed by a carriage return.

Response: S parameters (Snap)
or
SA parameters (Snap Absolute)

This request indicates that the user wishes a snap or display of certain memory locations.

The S form specifies that an offset (or relocation) value is automatically added to the address parameter(s). Using the Offset function, the user will have set the offset value.

The SA form specifies an Absolute value for the following address parameters:

aaa,n displays (snaps) n locations starting at aaa

aaa-bbb displays (snaps) locations aaa through bbb

aaa displays (snaps) location aaa

The parameters follow the function identifier (S or SA) without intervening blanks.

When the Snap request is satisfied, the TDS subroutine responds with a question mark which indicates that another function, or a return to processing, may be requested.

Response: P parameter (Patch)
or
PA parameter (Patch Absolute)

This request indicates that the user wishes to patch or replace the contents of selected locations within the subsystem.

The P form specifies that an offset (or relocation) value is automatically added to the patch-location parameter. Using the Offset function, the user will have set the offset value.

The PA form specifies an absolute value for the following patch-location parameters:

aaa/bbb where aaa (1-6 octal digits) is the location at which the octal patch bbb is to be made. Fields aaa and bbb must be separated by one blank, and bbb can be any of the following:

```
 xxxxxxyyyyyy
Rxxxxxxyyyyyy
 xxxxxxyyyyyyR
Rxxxxxxyyyyyy R
```

where x is an octal digit of the upper-half word, y is an octal digit of the lower-half word, and R is a Relocation indicator specifying that the upper half, lower half, or both halves of the word are to be incremented by the offset value. Where consecutive patching begins at aaa, successive patches may be given in the form of comma-separated fields. The patch fields (bbb,ccc,....) may contain up to 12 octal characters which are right-justified and stored in the respective memory locations.

When this request is satisfied, the TDS subroutine responds with a question mark which indicates that another function, or a return to processing, may be requested.

Response: X (Display Registers)

This request indicates that the user wishes to display the contents of all working registers--A, Q, E, I, and all index registers.

Alternatively, this function allows selective designation of individual registers using the following forms of the X request:

form	meaning
Xn	Display the nth index register only
XA	Display the A-register only
XQ	Display the Q-register only
XE	Display the E-register only
XI	Display the indicator register only

When this request is satisfied, the TDS subroutine responds with a question mark which allows another request to be given.

Response: Mparameters (Modify Registers)

This request indicates that the user wishes to modify the contents of a working register.

The permissible forms of this request are:

form	meaning
MXnØxxxxxx	Modify nth index register
MAØxxx...xx	Modify A-register
MQØxxx...xx	Modify Q-register
MEØxxx	Modify E-register
MIØxxxxxx	Modify indicator register

where x is an octal numeric, and the right-hand, blank-separated field is always the modification data.

After this request is satisfied, the TDS subroutine responds with a question mark which allows another request to be given.

Response: B parameters (Breakpoint)
or
BA parameters (Breakpoint Absolute)

This request indicates that the user wishes to establish a new breakpoint, or debugging location, within the system. Establishing a breakpoint is analogous to assembling an XED TDS instruction into the subsystem at a location logically preceding the instruction residing at the address specified in the breakpoint request. The instruction that has been replaced at the specified address is executed following the requested breakpoint function. Each time the specified address is encountered in the execution of the subsystem, TDS will print BREAKPOINT aaa, stop execution, and ask for a function request (FUNCTION?). Location aaa will be relative to the offset value, if any.

The permissible forms of the Breakpoint request are:

form	meaning
B	Break at effective address offset + 0
BA	Break at location 0, Absolute
Baaa	Break at effective address offset + aaa
BAaaa	Break at loation aaa, Absolute

After the request is satisfied, the TDS subroutine responds with a question mark which allows another request to be given.

Response: Oxxxxxx (Offset)

This request indicates that the user wishes to set (or reset) the offset value to xxxxxx, where x is an octal numeric. The offset value is initially set to zero by TDS. When this request is satisfied, the TDS subroutine responds with a question mark which allows another response to be given.

Response: D (Delete breakpoint)

This request indicates that the user wishes to delete the current breakpoint by replacing it with either a NOP instruction or the original instruction if the breakpoint was placed there by the TDS Break option.

When this request has been satisfied, the TDS subroutine responds with a question mark which allows another request to be given.

Response: R (Return)

This request causes control to return to the subsystem at the location following the XED TDS or breakpoint XED instruction.

Response: Rxxxxxx (Return to location)
or
RAxxxxxx (Return to Absolute location)

This request causes a special return to the subsystem at location xxxxxx, where xxxxxx is an octal address. In the RA form of this request, the offset is added.

Error Indications and Messages:

1. ILLEGAL INPUT - RETYPE -- typed message when illegal input is typed in response to -FUNCTION?" or "?".

2. ILLEGAL COMMAND, MUST PRECEDE DATA WITH S,P,R,D,X,B,O, or M -- typed message in response to parameters not preceded by a function-type indicator.

3. ROOM FOR BREAKPOINT ENTRIES EXHAUSTED -- typed message when no more breakpoints can be accepted.

INFORMATION SYSTEMS

GENERAL ELECTRIC

LITHO U.S.A.