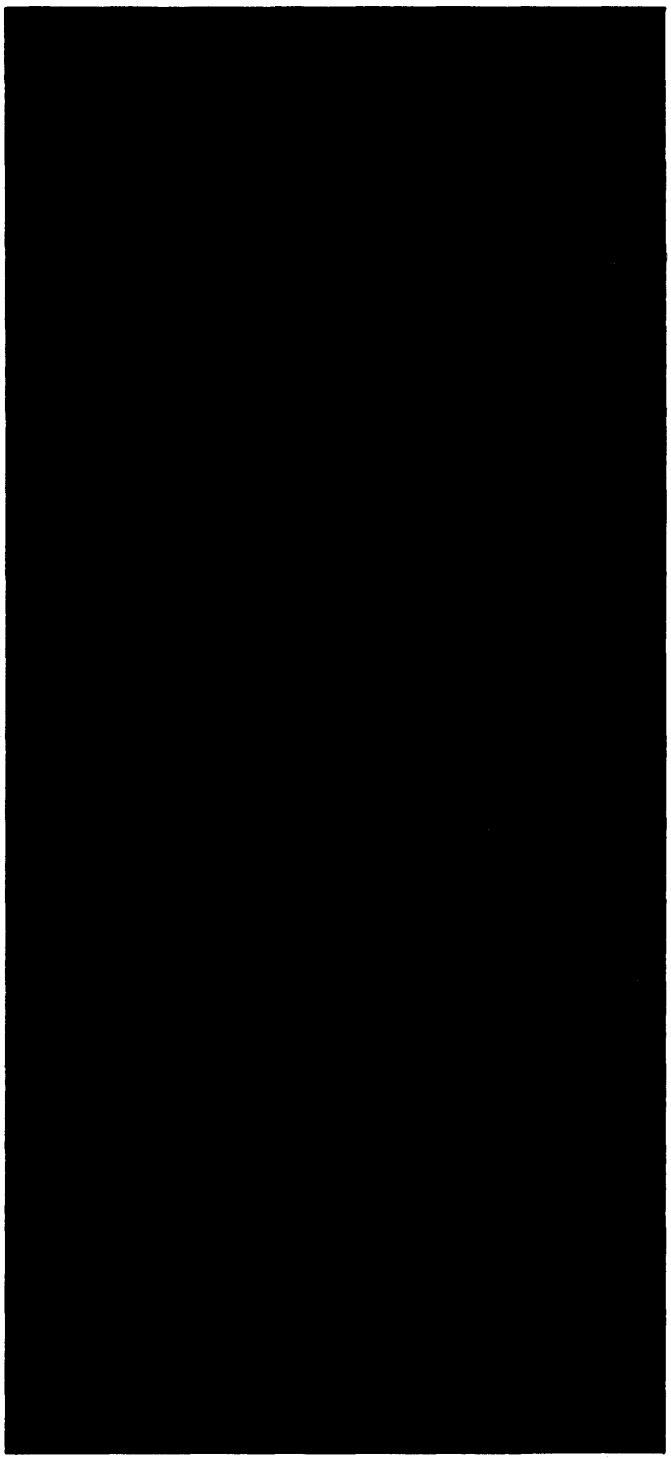
Honeywell

TIME-SHARING SYSTEM PROGRAMMERS' REFERENCE MANUAL

SERIES 600/6000

GCOS

SOFTWARE



Honeywell

TIME-SHARING SYSTEM PROGRAMMERS' REFERENCE MANUAL

SERIES 600/6000

GCOS

SUBJECT:

Subsystem Organization, Programming Methods for Extending System Capabilities, Instructions for Placing a Program in the System, Command Language and Primitives, File Formats, and Honeywell-Supplied Systems.

SPECIAL INSTRUCTIONS:

This manual, Order Number BR39, Rev. 1, supersedes CPB-1514C, dated September 1970, Addendum No. 1, dated February 1971 and BR39, Rev. 0, dated September 1970. The new order number is assigned to be consistent with the overall Honeywell publications numbering system. Series 600 Software Release 5.0 and Series 6000 Software Release C (SR was formerly System Development Letter-SDL) information has been added to this edition of the manual. Technical additions and changes from the previous edition are indicated by change bars in the margins; deletions are indicated by asterisks.

DATE:

November 1971

ORDER NUMBER:

BR39, Rev. 1 (Supersedes CPB-1514C)

FILE NUMBER:

1613, 1713

PREFACE

This manual provides methods for the experienced programmer to develop and extend the capabilities of the time-sharing system via new subsystems.

The text material includes an explanation of subsystem organization, information required to program for the Series 6000[1] GCOS Time-Sharing System, and instructions for placing a program in the system. Other sections are devoted to command language and primitives, file formats, and Honeywell-supplied subsystems.

GCOS is a coded system designed to extend the power of Series 600/6000 in the areas of program preparation and maintenance, data control, operations control, and utility functions. It is supported by comprehensive documentation and training; periodic program maintenance and, where feasible, improvements are furnished for the current version of the system, provided it is not modified by the user.

[1]All references in this manual to Honeywell Series 6000 systems are equally applicable to Honeywell Series 600 systems, except those references explicitly specifying Series 600 only.

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 600 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 600:		
Hardware reference:			
Series 600	System Manual	371	BM78
DATANET[1] 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards	1688	BS19
System initialization:			
GCOS Startup	System Operating Techniques	DA10	DA10
Communications System	GRTS/355 Startup Procedures	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File system	GCOS File System	1513	BR38
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processing	DA37	DA37
Program maintenance:			
Object Program	Source Object Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
Peripheral on-line testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Language processors:			
Macro Assembly Language	Programming Reference Manual	1004	BN86
COBOL Language	COBOL Reference Manual	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
FORTRAN IV Language	FORTRAN IV	1006	BN88
DATANET 355	DATANET 355 Macro-Assembly Program	1660	BB98
Generators:			
Sorting	Sort/Merge	1005	BN87
Merging	Sort/Merge	1005	BN87
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator	1663	BW23
Remote terminal system:			
DATANET 355	GRTS/355 Programming Reference	1664	BJ66
DATANET 30	GRTS/30 Programming Reference	1558	BR68

[1]Trademark.

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 600:		
Service and utility routines:			
File I/O	File and Record Control	1003	BN85
Loader	General Loader	1008	BN90
Utility programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Program	1651	BS07
FORTRAN	FORTRAN IV Subroutine Libraries	1620	BR95
Controller loader	Relocatable Loader	DA12	DA12
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch	1642	BR99
System Programming	GCOS Time-Sharing System - System Programmer's Reference	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	Time-Sharing FORTRAN	1566	BR70
Text Editing	Time-Sharing Text Editor	1515	BR40
Handbooks:			
Console Messages	GCOS Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GCOS GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards and Abort Codes	1691	BJ69
Software maintenance (SMD):			
Table definitions	Introduction and System Tables	1488	BR17
Startup program	Startup (INIT)	1489	BR18
Input system	Input System	1490	BR19
Peripheral allocation	Dispatcher/Peripheral Allocation	1491	BR20
Core allocation/rollcall	Rollcall, Core Allocation and Operator Interface	1492	BR21
Fault processing	Fault Processing	1493	BR22
Channel modules	I/O Supervision (IOS)	1494	BR23
Error processing	Exception Processing	1495	BR24
Output system	Termination and SYSOUT	1496	BR25
File system modules	File System	1497	BR26
Utility programs	GCOS Utility Routines	1498	BR27
Time-sharing system	Time-Sharing Executive	1501	BR29

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 6000 SYSTEM

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 6000:		
Hardware reference:			
Series 6000	Summary Description	DA48	DA48
DATANET 355	DATANET 355 Systems Manual	1645	BS03
Operating system:			
Basic Operating System	Comprehensive Operating Supervisor (GCOS)	1518	BR43
Control Card Formats	Control Cards	1688	BS19
System initialization:			
GCOS Startup	System Startup and Operation	DA06	DA06
Communications System	GRTS/355 Startup Procedures	1715	BJ70
Storage Subsystem Startup	DSS180 Disk Storage Subsystem Startup Procedures	DA11	DA11
Data management:			
File System	GCOS File System	1513	BR38
Integrated Data Store (I-D-S)	Integrated Data Store	1565	BR69
File Processing	Indexed Sequential Processor	DA37	DA37
Program maintenance:			
Object Program	Source Object Editor	1723	BJ71
System Editing	System Library Editor	1687	BS18
Test system:			
On-Line Peripheral Testing	GCOS On-Line Peripheral Test System (OPTS-600)	1573	BR76
Language processors:			
Macro Assembly Language	Programming Reference Manual	1004	BN86
COBOL Language	COBOL Reference Manual	1652	BS08
COBOL Usage	COBOL User's Guide	1653	BS09
ALGOL Language	ALGOL	1657	BS11
JOVIAL Language	JOVIAL	1650	BS06
FORTRAN Language	FORTRAN	1686	BJ67
DATANET 355	DATANET 355 Macro-Assembly Program	1660	BB98
Generators:			
Sorting	Sort/Merge	1005	BN87
Merging	Sort/Merge	1005	BN87
Simulators:			
DATANET 355 Simulation	DATANET 355 Simulator	1663	BW23

FUNCTION	APPLICABLE REFERENCE MANUAL TITLE	FORMER PUB. NO.	ORDER NO.
	Series 6000:		
Service and utility routines:			
File I/O	File and Record Control	1003	BN85
Loader	General Loader	1008	BN90
Utility Programs	Utility	1422	BQ66
Conversion	Bulk Media Conversion	1096	BP30
System Accounting	GCOS Accounting Summary Edit Program	1651	BS07
FORTRAN	FORTRAN IV Subroutine Libraries	1620	BR95
Controller Loader	Relocatable Loader	DA12	DA12
Time-sharing systems:			
Operating System	GCOS Time-Sharing System General Information	1643	BS01
System Programming	GCOS Time-Sharing Terminal/Batch	1642	BR99
System Programming	GCOS Time-Sharing System - System Programmer's Reference	1514	BR39
BASIC Language	Time-Sharing BASIC	1510	BR36
FORTRAN Language	FORTRAN	1686	BJ67
Text Editing	Time-Sharing Text Editor	1515	BR40
Remote terminal system:			
DATANET 355	GRTS/355 Programming Reference	1664	BJ66
DATANET 30	GRTS/30 Programming Reference	1558	BR68
Handbooks:			
Console Messages	GCOS Typewriter Messages	1477	BR09
Index	Comprehensive Index	1499	BR28
Pocket guides:			
Time-Sharing Programming	GCOS Time-Sharing System	1661	BS12
Macro Assembly Language	GCOS GMAP	1673	BS16
COBOL Language	COBOL	1689	BJ68
Control Card Formats	GCOS Control Cards and Abort Codes	1691	BJ69

Rev. 7112

CONTENTS

CONTENTS (cont.)

SECTION I

INTRODUCTION

The Series 600/6000 GCOS (formerly GECOS) Time-Sharing System (TSS) is a feature of an integrated batch/remote time-sharing system. The time-sharing portion of GCOS 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 TSS does not occupy core storage if it is not in use; it does take a variable amount of core if the system is being used. Primary features of the system are discussed in the following paragraphs.

SIMULTANEOUS BATCH AND TIME-SHARING

The simultaneous batch and time-sharing feature allows the user to develop time-sharing applications without dedicating a complete computer system 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 SYSTEM EXTENSION BY USER

As in the batch system, it is necessary that the user be allowed to write programs for his unique applications. The time-sharing system is designed as an executive, or monitor, servicing generic subsystems. The subsystems are analogous to slave programs in the batch environment. Any number of actual subsystems can be combined into one logical subsystem for ease of use by the ultimate user.

It is expected that users will add capabilities via new subsystems to suit their local installation requirements. The time-sharing system has been designed to minimize the efforts required to generate and install these subsystems. The debugging facilities permit checkout of a new subsystem while normal time-sharing operation continues.

SECTION II

SUBSYSTEM ORGANIZATION

A subsystem consists of two logically and physically separate parts: a program and a program descriptor. The organization of these parts is the subject of this section.

SUBSYSTEM PROGRAM ORGANIZATION

A subsystem program (that is, the block of code to be executed) is organized like a batch-environment slave program. It can be written in any language whose object code is loadable by the General Loader at system edit time. As many other subsystems as desired may be included, as well as SYMREF, SYMDEF, and BLOCK pseudo-ops and library subroutines. The program executes within the TSS core storage area; the base register is set around the code as loaded, so that the subsystem is always unaware of its relative position in core.

The following restrictions apply to subsystem programming:

- A subsystem data area of 100 (decimal) words must precede the program coding. TSS uses this space for such things as bookkeeping during program swap and register storage. (The user must provide 36 of these words by using BSS 36 as the first code generated in his subsystem.)
- Intentional faults called derails (DRLs) -- instead of MME functions, as in a batch processing environment -- must be used to request service functions. These functions are provided by the TSS Executive.

SUBSYSTEM PROGRAM DESCRIPTOR ORGANIZATION

The Communication Region module, TSSA, consists of three block common areas: .TSCOM, the communication region proper, .TPRGD, the program descriptor block, and .TPCOM, the command language and primitive lists block.

The GMAP-generated program descriptor is stored in the .TPRGD block in the TSS Communications Region. The descriptor is separated into a primary portion and a command-language/primitive list. (See Section IV for details.)

Primary Portion of Program Descriptor

The primary portion of the descriptor is fixed in length and is placed in sequence with the other subsystem descriptors at the beginning of the program descriptor block. It includes the following designer-supplied information:

- Subsystem name (in ASCII lower-case alphanumeric)
- Pointer to subsystem command-language/primitive list
- Number of subsystem build mode command-language words, if any

Program Descriptor Command-Language/Primitive List

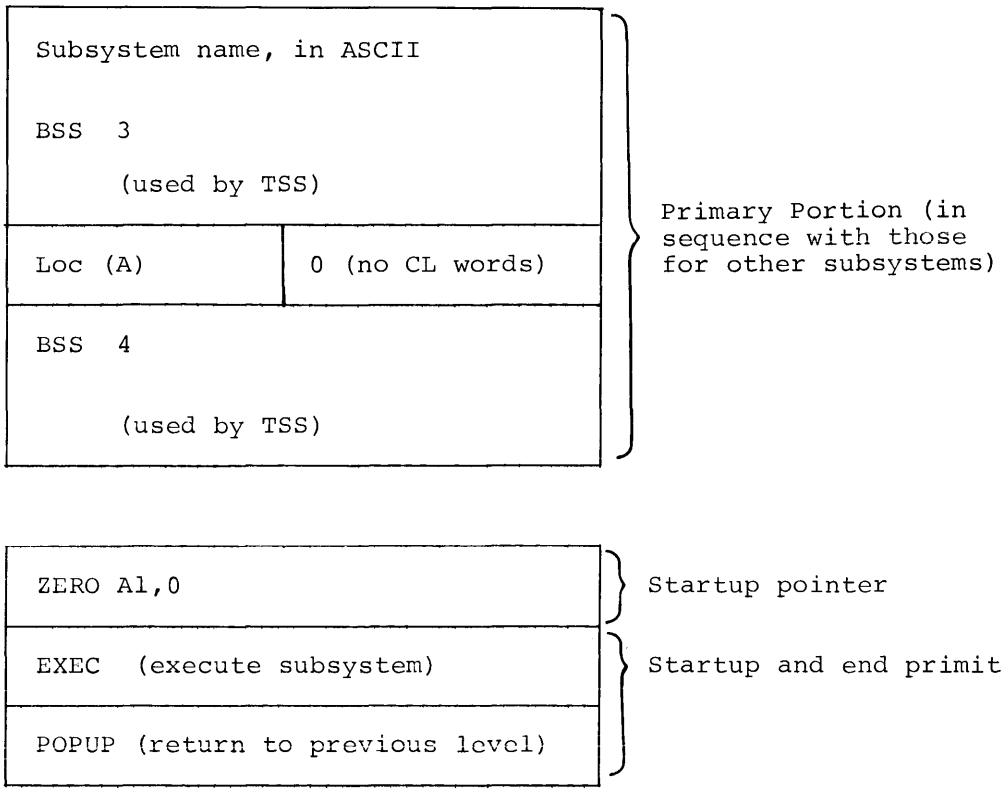
The command language list, pointed to by the primary portion, comprises for each command (1) the command itself, (2) an associated mask word, and (3) a pointer to the primitives for the command. The list ends with a pointer to the primitives for the subsystem startup procedure.

A primitive is a high-level instruction sequence that directs subsystem execution when the subsystem is initially encountered (startup primitive) or when build-mode command language is encountered (command primitive). The primary function of primitives is to combine several related subsystems logically into a larger subsystem, so that the user at the terminal needs to deal only with the larger subsystem. Within the CARDIN subsystem, for example, one can call up the SCAN subsystem in build mode.

The subsystem startup procedure allows the designer to specify either (1) a direct execution of the subsystem after selection by the user or (2) calling of other subsystems or performance of other preliminary operations following user selection and before subsystem loading and execution. The BASIC subsystem, for example, calls the OLDN subsystem (via a primitive) to assign a current file and provide the OLD/NEW file request. A second primitive then places the subsystem in build mode to accumulate terminal input until a command-language word is encountered. (See Section IV for details.) The LIST subsystem, on the other hand, begins execution immediately upon selection, provided that no data exists in the input collector file (SY**).

Program Descriptor Example

For the simplest case, a subsystem with no command language (CL) and a direct execute at startup, the program descriptor layout would be as follows:



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

SECTION III

PROGRAMMING FOR THE TIME-SHARING SYSTEM

Writing a subsystem program for the time-sharing system (TSS) is not significantly different from programming in slave mode for the batch environment. The primary difference concerns service functions. All MME functions provided by GCOS are eliminated, and similar functions are provided for time-sharing by the TSS Executive via derail (DRL) instructions.

BASE REGISTER PROTECTION

While the subsystem program is in execution, the base register is set to prevent the subsystem from referencing 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 (decimal) words in each subsystem for bookkeeping. This area must be at the beginning of the program. The loader normally reserves 64 words at the beginning of a program, and this area is usable for the bookkeeping as well. Thus, a subsystem program would normally start with a BSS 36 to reserve the additional space required to make a total of 100 words.

Words 0-17 (octal) of this data area represent the subsystem fault vector and are defined as for a GCOS slave. These words are used in pairs, one pair for each type of fault which can be returned to the subsystem program, as shown on the next page.

Words

0,1	Illegal op-code fault Zero op-code fault Command fault Connect fault Lockup fault
2,3	Memory fault
4,5	Fault tag
6,7	Divide check
10,11	Overflow fault
12	Not used
13	.LBRT - location of abort and reason code (not yet implemented)
14,15	.LBRK - break status fault
16,17	.LSZTM- time and size limit 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, TSS checks the second word of the corresponding fault-vector pair. If the second word is not zero, and the address specified is within subsystem bounds, TSS stores the IC and indicators in the first word of the pair. If the second word is zero, TSS aborts the subsystem program, giving a message to the terminal. For example, if a subsystem program wishes to recover from divide check, overflow, and Break status faults, the vector would be set up as follows:

0 1		No recovery wanted
2 3		No recovery wanted
4 5		No recovery wanted
6 7	TRA DVCK	Transfer to divide check recovery routine
10 11	TRA OVFLO	Transfer to overflow recovery routine
12	Not used	
13	Not implemented	
14 15	TRA BRKFLT	Transfer to Break status wrapup routine
16 17	TRA SZTM	Transfer to size/time-limit-exceeded wrapup vector

The transfers, if any, must be stored in the fault vector by coding within the subsystem. Transfer vectors must be reinitialized after each recovery fault. The rest of the 100 (decimal) words in the subsystem data area are reserved exclusively for TSS usage.

SUBSYSTEM SWITCH WORD

The TSS provides a subsystem switch word (.LSWTH) for each user. This word is used to maintain status or to pass information between subsystems during execution. Subsystem programs can modify or test the bit settings (1) with the SETSWH and RSTSWH derails or (2) with the primitives defining control flow in the user-selected subsystem. Thus, a programmer can make execution of primitives dependent upon conditions set up by his subsystem program.

Bits 0-17 are reserved for use by subsystems issued with the time-sharing system. Bits 18-35 are for user definition. Current TSS usage is as follows.

Bits 0 - ALGOL RUN (RUNY Subsystem) or Call TAPE subsystem after execution (EDTX Subsystem).

1 - JOVIAL RUN (RUNY Subsystem) or Call RECO Subsystem after execution (EDTX Subsystem).

2 - BREAK received while in EDBN (EDITOR build mode).

3 - Unused

4 - Unused

5 - Transparent Paper Tape Mode (LUCID)

6 - Object code execution limits in effect

7 - Break status requested

8 - OLDP#/NEWP# command in effect

9 - CHAIN overlay (BASIC) in use

10 - Automatic dump (SY**) engaged

11 - HOLD command in effect

12 - Paper tape source file (*TAP) exists

13 - FORTRAN source file exists

14 - Last RESEQUENCE command was in a non-BASIC subsystem

15 - OLD/NEW command received from user

16 - NOPARITY command in effect

17 - Valid data on SY** file

As already noted, users may not define usage of bits 0-17 but can manipulate these bits by using derails or primitives. The bits of special interest to a user designing his own subsystem are described in the following paragraphs.

LUCID Command Bit (5)

Normally, paper tape output from a time-sharing terminal must be in ASCII 7-bit, even-parity code. However, typing the LUCID (or #LUC in the EDITOR subsystem) command instead of the TAPE (or #TAP) command when using a manufacturer-supplied subsystem permits input of non-ASCII tape (for numerical control, for example). Any character code from 000-377 (octal) is accepted and is stored without editing or parity modification on the time-sharing file.

Typing LUCID causes the Executive to set bit 5 of the subsystem switch word. If bit 5 is set, the paper tape subsystem (TPTA) requests a larger- than-normal TAP* file for accumulation of the input. When input terminates, TAP* is rewound by TPTA and is left in the AFT as is. TPTA then resets bit 5 and returns to the SYSTEM? level.

The user can implement the LUCID command in his subsystem, or he can design the subsystem to handle non-ASCII tape input automatically. For example, running under LODX, the TAP* file size would first be defined as required. A DRL SETSWH would set bit 5 of .LSWTH and execute a DRL TAPEIN to start input. Operation would then be as already described until termination of tape input. Control would then return to the subsystem, which would reset bit 5.

Input is terminated whenever a pause greater than 1 second is encountered during transmission. The X-OFF normally required for termination of paper tape input via a DATANET[1] 355 is not necessary.

NOTE: This feature cannot be used with a DATANET 355 unless communication is via a high-speed line adapter (HSLA).

---

[1] Trademark

Break Status Bit (7)

Using the Break status bit makes it possible to pass this status to a subsystem for unique action on its part rather than terminating the subsystem and returning to the previous level, as will otherwise automatically occur.

To use this feature, the programmer must set up subsystem location 15 as a transfer vector. (Locations are in octal, as are all others following.) Also, he must set bit 7 of the subsystem switch word with a DRL SETSWH.

When a Break status occurs, it is detected by the line service module, TSSJ. The module then interrogates bit 7. If it is off, the break is processed normally. If the bit is on (as described in the preceding paragraph), the status is transferred to .LTSSV; and bit 35 of .LFLAG is set. (Both are UST locations.)

Executive Module, TSSM, in returning to subsystem execution via RETSBS, examines bit 35 of .LFLAG before dispatching. If bit 35 is off, return to subsystem execution is normal. If bit 35 is on (as described in the preceding paragraph), TSSM turns it off and examines location 15 of the subsystem for a transfer vector. The transfer will be made to the user's coding for action following a Break status for the subsystem. (Or, if location 15 contains a DRL RETURN, the derail is executed; and the subsystem terminates normally.)

When a valid transfer vector is present, TSSM withdraws the current IC and I from subsystem location 22 and places it in location 14. It then places the contents of location 15 (the transfer vector) in location 22 and clears location 15 to prevent blind looping. (Note: therefore, the subsystem must restore the contents of location 15 after each break if continued unique processing of breaks is desired.) TSSM then continues with normal dispatch to the subsystem.

If no transfer vector or an out-of-range transfer is present in location 15 (and bit 7 of .LSWTH is set), TSSM terminates the subsystem, sends either an ILLEGAL BREAK VECTOR or an OUT OF RANGE message to the terminal, and returns to the subsystem-request level, just as TSSJ does in normal procedure.

HOLD/SEND Bit (11)

During printer or paper tape output from his subsystem program, the user can prevent interruptions at the terminal from console-issued or master-user-issued messages by setting bit 11 of the subsystem switch word. This can be done with the primitive STRUE or the derail SETSWH.

When the bit is reset with the primitive STFALS or the derail RSTSWH, the last message withheld is typed at the terminal. The user assumes responsibility for any warnings he may have missed during the interval the bit was on.

PARITY/NOPARITY Bit (16)

Upon initial connection for direct access in the TSS, bit 16 of the subsystem switch word is off; and the terminal receives ASCII 7-bit, even-parity code. At the subsystem-selection level, the NOPARITY command turns bit 16 on; and the terminal receives 8-bit, parity-independent code. The PARITY command is used to turn the bit off again. (These commands are also available within the BASIC, FORTRAN, and CARDIN subsystems.)

Thus, by controlling bit 16 with the SETSWH and RSTSWH derails, a user programming his own subsystem can cause an automatic change in the code sent to a terminal. Note, however, that this option currently functions only with type 4 terminals (a hardware limitation). See DRL TERMTP for identification of terminal types.

SYSTEM MACROS

A set of TSS macros is available to facilitate the writing of TSS modules and subsystems. Two of these useful to the designer of a normal subsystem are .SSDRL and PRNTTY. the .SSDRL macro provides the symbolic address-value equivalences for the service function DRLs (see the section on "Service-Function Derails"); PRNTTY is convenient for issuing messages to the terminal.

The TSS macro library is loaded at assembly time by the call LODM.

Appendix A describes the function of LODM and gives the definition of each macro.

DERAILS

Since a subsystem program is prevented from referencing any memory area outside that allocated and protected by the base register, a service function request must be made by an intentional fault--a derail (DRL). DRL functions and their associated addresses are shown in the table on the following page.

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 calling sequence.

The two following subsections contain descriptions of the derails available to the designer of a normal subsystem. (DRLs that are privileged or otherwise restricted are marked with a superscript in the table on the following page and are not described in this manual.) The first section, "General Service Function Derails," comprises those not concerned with mass storage file activities. The second section, "Mass Storage File Activity Derails," comprises those used by TSS for performing service functions for files on magnetic disc or drum peripheral devices.

Within each subsection the descriptions appear in alphabetical order by derail name. To locate a description by page number, see the index under the heading "DRL" or under the derail's name or the name of its function. (These names appear in the table on the following page.)

General Service Function Derails

DRL ABORT, ABORT (octal 7)

8	16
DRL	ABORT

This derail indicates to the 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 subsystem.

DRL Name	DRL Function Name	DRL Address octal	DRL 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 a 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 and Line Number	23	19
PDIO[1]	Do I/O on a System File	24	20
RESTOR	Overlay-Load a Subsystem	25	21
SPAWN	Pass File to Batch Processor	26	22
TAPEIN	Start Paper Tape Input	27	23
CALLSS	Internal Call to Another Subsystem	30	24
USERID[1]	Pass USERID and Priority to Executive	31	25
TERM[1]	Clean Up UST After User Termination	32	26
PASUST	Pass UST to Subsystem	33	27
MORLNK	Add Links to Temporary File	34	28
NEWUSR[1]	Log-on New User Without Disconnect	35	29
FILACT	Permanent File Activities	36	30
SETLNO	Set Line Number/Increment in UST	37	31
SYSRET	Return to System	40	32
STPSYS[1]	Stop Execution of Master Subsystem	41	33
STATUS	Check I/O Status (not implemented)[2]	42	34
DRLDSC	Disconnect Terminal	43	35
PASDES	Pass AFT File Names And Descriptions	44	36
JSTS	Obtain Job Status	45	37
CGROUT	Process Line Switch	46	38
PART	Partial Release of Temporary File	47	39
GROW	Grow a Permanent File	50	40
ABTJOB	Abort Batch Job	51	41
CONSOL[1]	Talk to System Console	52	42
SWITCH	Switch Temporary File Names	53	43
DRLIMT	Store Processor Time Limit	54	44
JOUT[1]	Batch Output Request	55	45

---

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

[2] If executed, this derail is effectively a TRA*+2 instruction.

DRL Name	DRL Function Name	DRL Address octal	DRL Address decimal
KOTNOW	Keyboard Output from Unfilled Buffer	56	46
OBJTIM	Processor Time and Core Size Limit	57	47
PASFLR	Pass File to Remote Batch Processor	60	48
STOPPT	Stop Paper Tape Input	61	49
DRLSAV	Save Program on Permanent File	62	50
TASK	Spawn a Special Batch Activity	63	51
PSEUDO	Simulated Keyboard Input	64	52
PRGDES	Pass Program Descriptors	65	53
GWAKE	Go to Sleep, Wake Me in N-Sec	66	54
IDS	To Make an Entry into .MIDSC	67	55
ATTRI	Pick up Users Attributes (Type 7 Block)	70	56
T.STAT[1]	Write Statistical Collection File	71	57

DRL ABTJOB, ABORT BATCH JOB (octal 51)

```
          8         16
          DRL       ABTJOB
loc       BCI       1,snumb
```

This derail is used to abort the batch job submitted from the requesting terminal via TSS, identified by snumb. On return, the lower-half of the derail-argument word (loc)will contain the following result-code:

0 = job not in system
1 = job not initiated from requesting terminal (not your job)
2 = abort initiated

DRL ADDMEM, ADD MEMORY (octal 16)

```
8         16
DRL       ADDMEM
```

C(A)	Return location	0
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 lower boundary. The number of words specified must be modulo 1024, or the number will be rounded.

DRL ATTRI, PICK UP USERS ATTRIBUTES (octal 70)

IDS subsystems that access permanent IDS files need access to the file attribute information. This information is maintained by the file system in the type 7 block. This derail will allow a subsystem to retrieve the attribute information.

Calling format:

```
8         16
DRL       ATTRI
ZERO      L(destination), L(fileid)
ZERO      L(buffer), L(status)
RETURN
```

where:

1. Destination - a 57 word buffer in the users' subsystem to store the attribute information.
2. Fileid - 2 word filename in BCD.
3. Buffer - 158 word working buffer used by file system.
4. Status - 1 word status return location.

DRL CALLSS, INTERNAL CALL TO ANOTHER SUBSYSTEM (octal 30)

```
8         16
DRL       CALLSS
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; that is, nesting to more than two levels is not allowed.

The called subsystem may be any subsystem known to TSS (one with a program descriptor in the TSS communication region). Name is the subsystem name as recorded in the descriptor.

DRL CGROUT, PROCESS LINE SWITCH (octal 46)

```
8         16
DRL       CGROUT
VFD       18/0,6/op,H12/line-id
ZERO      L(snumb),0
```

where:

op = 25 for line-switch from TSS to DAC

line-id = line-identifier (station code) of the line to be switched

or

op = 26 for line-switch from DAC to TSS

line-id = 0 if all lines connected to snumb have been reswitched, or

= line-id of a line still connected (receiving output).

This derail is used for two purposes:

(1) To switch a remote-terminal line, identified by line-id and connected to TSS, to direct-access connection with the slave program identified by snumb. (The line-id can either be considered as a two-character code interpreted in BCI--that is, the GERTS "station code" -- or as a four-digit octal number, which corresponds to the "channel" number printed by TSS.) The specified batch slave program must have been submitted through TSS.

(2) The complementary use of this derail is to switch any lines connected to the slave program identified by snumb back to TSS, provided such lines were originally connected to TSS. This usage of the derail (op = 26) may need to be repeated until line-id (bits 24-35 of the argument) indicates that all lines have been switched. Note that line-id needs to be rezeroed before each repeat call.

DRL CORFIL, DATA FROM/TO CORE FILE (octal 17)

```
8         16
DRL       CORFIL
```

C(A)	Data location	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 equal to or less than 10. The left-half of Q contains the number of the core-file cell (i) at which transmission is to begin. The core-file cells are numbered 1 through 10. The right-half of Q (k) indicates the type of operation desired:

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

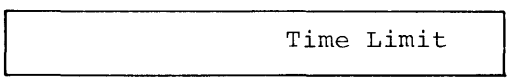
DRL DRLDSC, DISCONNECT TERMINAL (octal 43)

8	16
DRL	DRLDSC

This derail gives the subsystem the ability to disconnect terminals that do not disconnect upon reception of the EOT character (ASCII 004). Any input that has been placed into the line-buffer area for the terminal is written out; upon completion of the MME GEROUT the disconnect is issued. The subsystem will not be reentered after execution of the DRLDSC. This derail is normally used to ensure the disconnection of a terminal of any type.

DRL DRLIMT, STORE PROCESSOR TIME LIMIT (octal 54)

8	16
DRL	DRLIMT



This derail stores the processor time limit for the subsystem in .LIMIT of the UST. The time limit is placed in the A-register, right-justified, expressed in seconds. The derail processor converts the value to clock pulses and stores it in the UST.

DRLIMT functions in conjunction with DRL OBJTIM. If an installation time limit has been enabled by execution of OBJTIM and a time limit has also been stored in .LIMIT by DRLIMT, then the smaller of the two is stored in .LIMTR and used by the time limit check in TSSM (see OBJTIM description).

DRLIMT is chiefly for use with the FORTRAN and BASIC subsystems, where it is executed by the RUN-nn option and allows the user to specify a time limit shorter than that set by the installation.

DRL GWAKE, WAKE ME LATER (octal 66)

```
8          18
LDQ        L(time)
DEL        GWAKE
```

where:

time - contains the number of seconds, right ustified the user wishes to sleep.

This derail causes the calling program to be set inactive and eligible for swap of the number of seconds specified in the Q-register. There is an inherent delay of 1 to 2 seconds in the derail. Breaks will cause the user to be awakened before the sleep time has elapsed.

DRL IDS, MAKE AN ENTRY INTO .MIDSC (octal 67)

DRL IDS will allow a time-sharing subsystem to access an I-D-S perm file under control of the I-D-S concurrent feature. I-D-S concurrent access assures that all time-sharing subsystems having access to the same I-D-S file do not interfere with each other or with any batch job also accessing the same file.

I-D-S bit in UST, bit 28 of .LFLG2

Calling format to DRL IDS:

```
DRL      IDS
ZERO     0, function number
ZERO     1L (arg list pointer),0
RETURN   abnormal
RETURN   normal
RETURN   variable
```

Function numbers

```
 1 - open
 7 - request file
 8 - release file
 9 - remove entry from PIC
10 - enter entry from PIC
12 - validate data
14 - remove TSS subsystem
```

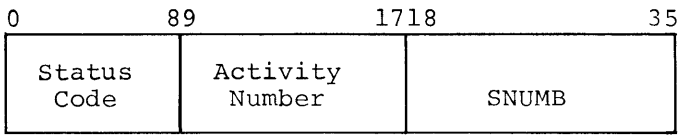
---

[1] for function 14 -- 0, 0

DRL JSTS, OBTAIN JOB STATUS (octal 45)

```
           8         16
           DRL       JSTS
loc        BCI       1,snumb
```

This derail is used to obtain the processing status of the batch job identified by snumb. On return the contents of the derail-argument word (loc) are as follows:



Status codes:

```
 0 - Output ready
 1 - Reading - card reader
 2 - Reading - magnetic tape
 3 - Reading - reading remote
 4 - Waiting - allocation
 5 - Waiting - peripherals
 6 - Waiting - core
 7 - In hold
 8 - In limbo
 9 - Executing
10 - Swapped out
11 - Waiting - tape
12 - Too big
13 - Overdue
14 - Output waiting
15 - Output complete
16 - Not in system
17 - Output complete
18 - Not accessible
19 - Not your job
20 - Aborted
```

The activity number is in binary.

DRL KIN, KEYBOARD INPUT LAST LINE (octal 4)

```
    8         16
    DRL       KIN
    ZERO      L(dat),L(count)
    ZERO      L(status)
```

This derail retrieves the last line of input. Normally this sequence would follow immediately the KOUTN sequence; however, this is not necessary. 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.

DRL KOTNOW, KEYBOARD OUTPUT FROM UNFILLED BUFFER (octal 56)

```
8          16
DRL        KOTNOW
ZERO       L(tally),L(char)
```

The call for KOTNOW is the same as the call for KOUT. The action is also the same, except that KOTNOW forces keyboard output from a partially filled buffer rather than waiting until the buffer has filled. This feature allows users to substitute KOTNOW for KOUT in subsystems with low output, where several messages may stack up in the buffer before its content is sent to the remote terminal by KOUT.

KOTNOW is a separate entry point to the KOUT coding, primarily for setting a flag and providing the test and decision logic to retain control within the derail until the buffer has been emptied.

The KOTNOW flag is tested immediately after the KOUTN test. If the flag has not been set, normal processing continues with a return to the subsystem. If the flag has been set by an entry at KOTNOW, the buffer is emptied and the allocator notified that I/O is in progress. An exit is then made to LINSRV.

DRL KOUT, KEYBOARD OUTPUT (octal 2)

```
8          16
DRL        KOUT
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 terminal. 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. Therefore, they must be refreshed prior to each execution.

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 is initiated. At this point, execution of the subsystem is 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      LINE2,19
       .
       .
       .
LINE1   ASCII 4, THIS IS LINE 1
LINE2   ASCII 5, THIS IS SECOND LINE
       .
       .
ENDC    OCT  015012177000
```

This sequence prints two lines with three characters -- carriage return, line feed, delete (rubout) -- appended to each line.

NOTE: Because of timing considerations and character set differences between terminal types it may be necessary to follow the carriage return, or line feed characters, with a number of delete (rubout) characters.

DRL KOUTN, KEYBOARD OUTPUT THEN INPUT (octal 3)

```
8          16
_____________________

DRL        KOUTN
ZERO       L(tally),L(char)
```

This derail 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 that for KOUT in that the line is left open for a response. The response can be retrieved by means of DRL KIN.

DRL OBJTIM, PROCESSOR TIME AND CORE SIZE LIMIT (octal 57)

```
8          16
_____________

DRL        OBJTIM
```

This derail causes the object program elapsed processor time and program core size to be checked against installation- or user-specified limits. If the installation has not assembled or patched the time and size limits into TSSA or a derail DRLMT has not been done, this DRL will have no effect.

In addition, this derail exists primarily for installation use in setting size and time limits for FORTRAN and BASIC programs. Even if these limits are set, they may not be satisfactory values for the subsystem being designed. If the user desires to set a separate processor object time limit for the subsystem or allow the subsystem user to set one he should use DRL DRLIMT.

OBJTIM sets bit 6 of .LSWTH (subsystem switch word), which makes the limits stored in TSSA effective for the current interaction. (The core size limit is stored in .TASSZ in number of words; the time limit is stored in .TASTM in number of clock pulses--seconds*64000.)

If bit 6 of .LSWTH is set, TSSM places the contents of .TASTM in .LIMTR of the UST. When OBJTIM and DRLIMT are concurrently in effect the smaller of the two time limits (installation or user) specified is placed in .LIMTR. Each time through TSSM, the contents of .LIMTR is decremented until it runs out or the interaction ends.

Core size is checked in TSSL at the time of the derail OBJTIM and at the time of any later ADDMEM request.

When either limit is exceeded, word 15 (octal) of the subsystem data area is checked for a fault vector. If a valid vector is present, TSS action is as described in Section III under "Subsystem Data Area and Fault Vector" (except that only a certain small time limit is given for wrapup). Bit 34 in word 14 is set when the size limit is exceeded; bit 35 is set when the time limit is exceeded. If no valid vector is present, TSS types one of the following messages and returns to the subsystem-selection level:

64 - EXECUTE TIME LIMIT EXCEEDED

65 - OBJECT PROGRAM SIZE LIMIT EXCEEDED

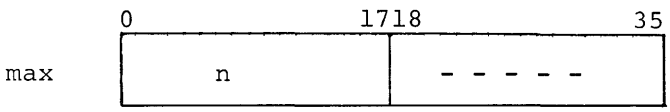
The time limit in .LIMTR (and the limit in .LIMIT, if present), word 17 of the subsystem data area, and bit 6 of .LSWTH are reset at the end of the interaction.

DRL PASAFT, PASS LIST OF FILES TO SUBSYSTEM (octal 22)

```
8          16
_____________________

DRL        PASAFT
ZERO       L(table),L(max) or 0
```

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 names or all user's file names (n = 0) in the area specified. The format of the table passed back at location table is:

Word 1	No. of active files[1] (bits 18-35)
2	File 1, characters 1-4
3	File 1, characters 5-8
.	.
.	.
N*2	File n, characters 1-4
N*2+1	File n, characters 5-8

[1]Maximum of 20

DRL PASDES, PASS AFT FILE NAMES AND DESCRIPTIONS (octal 44)

```
8         16
DRL       PASDES
ZERO      L(buff),0
```

where:

buff = BSS 3*(number of files in AFT)+1

This derail will place in buff the names and descriptions of the files in the user's available file table (AFT). The format of buff on return is:

Word 1	No. of active files
2	Filename, characters 1-4
3	Filename, characters 5-8
4	File description (see below)
5 6 7	. . .

File description format:

Bits	0-5	Device type
	6-17	Number of words per physical block
	if bit 18 = 0	bits 24-35 = number of links in the file
	if bit 18 = 1	bits 24-35 = number of 320-word blocks in the file
	if bit 19 = 0	Linked file
	if bit 19 = 1	Random file
	if bit 20 = 0	Temporary file
	if bit 20 = 1	Permanent file
	21-23	Unused

DRL PASFLR, PASS FILE TO REMOTE BATCH PROCESSOR (octal 60)

This derail is like derail SPAWN in all repects, except that the job is passed to the remote batch processor.

DRL PASUST, PASS UST TO SUBSYSTEM (octal 33)

```
8       16
DRL     PASUST
ZERO    L(buffer),N
```

where: N = No. of words to pass back.

This derail copies the user 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 TSS. This DRL should be used carefully, since this UST definition is subject to change.

DRL PRGDES, PASS PROGRAM DESCRIPTOR TO SUBSYSTEM (octal 65)

```
8       16
DRL     PRGDES
ASCII   1,SSNAME
ZERO    L(BUF),0
```

where:

SSNAME - is the name of the subsystem; the descriptor of the subsystem will be passed to the calling program.

BUF - is a nine-word buffer in the calling program in which the program descriptor will be placed.

This derail copies the program descriptor of the requested subsystem to the nine-word buffer specified by the call. If the specified subsystem does not exist, the nine-word buffer will be set to zeroes. Return is always to the word following the calling sequence.

The program descriptor is described in Section IV of this manual.

DRL PSEUDO, SIMULATED KEYBOARD INPUT (octal 64)

```
8          16
-------------------------------
DRL        PSEUDO
ZERO       L (tally), L (status)
return
```

where:

tally is a pointer to a preinitialized TALLYB, which in turn references the contigious block of data to be moved to the UST.

status is a pointer to a 1-word status return location (not currently implemented but must be specified). Return is to the second location past the DRL in the calling subsystem.

This derail allows a subsystem to place data in the UST I/O buffer for recovery by another subsystem using DRL KIN. For example, subsystem X could use this DRL to pass data to subsystem Y, which has been activated by a DRL CALLSS. When Y's activities end, control is returned to X. In this manner, X passes data to Y and other subsystems for action. Subsystem X is, in effect, a surrogate for a user subsystem, availing itself of the capabilities of a "library" of existing subsystems to perform its functions.

For each execution (1) the data to be transferred must be in a contigious block of not more than 62 words (244 nine-bit characters, with a carriage return as the last character in the string) and (2) the tally must be reinitialized. The data may be coded in any form, so long as it is placed in 9-bit characters. To the DRL KIN it appears just like keyboard input.

Error checks are made by the derail processor upon receipt of the derail. A detected error causes output of one of the following messages:

xxxxxx ADDRESS OUT OF RANGE

Tally address or status word address xxxxxx is outside subsystem limits.

DRL PSEUDO TALLY INCORRECT

Tally is set for 6-bit characters or attempts to pulse out more than 248 characters.

NOTE: This derail is not available on DATANET 760 terminals.

DRL RELMEM, RELEASE MEMORY (octal 15)

```
8       16
DRL     RELMEM
```

C(A)	Return location	0
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 (the right-half must be zero). This address will be taken relative to the new base. When low memory is released, it is the responsibility of the user to reestablish the fault vector or clear it to zero.

DRL RESTOR, OVERLAY-LOAD A SUBSYSTEM (octal 25)

```
8       16
DRL     RESTOR
ASCII   1, name
ZERO    loc, 0 or non-0
ZERO    tra, 0
```

where:

name is the ASCII name by which the subsystem is identified in its program descriptor.

loc is the address at which to load the subsystem. If the lower half of this parameter is 0, the subsystem location 0 is loaded at loc. If the lower half of this parameter is not 0, the subsystem is loaded such that the initial load address is at loc.

tra is the location of the next instruction to be executed.

To overlay-load a permanent file or a program from a multiprogram permanent file, the following coding is required:

```
8       16
DRL     RESTOR
ZERO    nameptr,0 or 1
ZERO    loc, 0 or non-0
ZERO    tra, bufloc
```

where

nameptr points to 3 words containing:

for nameptr,1 (used in referring to a multiprogram file) - an ASCII file name (1-8 characters) in words 1 and 2 and a BCD program name (1-6 characters) in word 3.

for nameptr,0 (used in referring to a single-program file) - the filename, as above, in words 1 and 2. Word 3 is ignored.

loc is the location at which to start loading the file. If the lower half of the parameter is 0, location 0 of the program on a permanent file will be loaded at loc. If both loc and the lower half are 0, the subsystem is loaded according to the program control block initial load address. If the parameter is non-0, the initial load address of the file will be loaded at loc.

tra is the location of the next instruction to be executed. If tra is 0, the entry address is taken from the program control block.

bufloc is (1) the location of a 40-word work buffer, when a single-program file is to be restored (nameptr,0) or (2) the location of a 40-word or 64-word buffer (its size depending on block size for the device where the file resides) when a program from a multiprogram file is to be restored (nameptr,1).

This function will load a subsystem or the contents of a permanent file as an extension of, or an overlay of, the calling subsystem. (See also DRLSAV.) When the lower portion of the parameter containing loc is nonzero, the user is indicating that the data area affixed by the loader is to be relinquished. This allows such things as floatable libraries to be loaded without wasted space. Upon return, the upper half of the A-register contains the number of words loaded.

A user subsystem is aborted, with one of the following messages to the originating terminal, if an error occurs during a DRL RESTOR:

000ss BAD STATUS-DRL SAVE/RESTOR FILE

where ss is erroneous status

SAVE/RESTOR FILE NAME NOT IN AFT

DRL RETURN, RETURN TO PRIMITIVE LIST (octal 5)

```
8         16
DRL       RETURN
```

This derail indicates to the Executive that this subsystem process has reached a normal termination. The TSSH 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 Section IV.)

DRL RSTSWH, RESET SWITCH WORD (octal 11)

```
8         16
DRL       RSTSWH
```

This derail is similar to 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.

DRL DRLSAV, SAVE PROGRAM ON PERMANENT FILE (octal 62)

```
8         16
DRL       DRLSAV
ZERO      nameptr, 0 or 1
ZERO      loc 1, loc 2
ZERO      entry addr, load org
ZERO      tra, bufloc
```

where:

nameptr points to 3 words containing:

for nameptr,1 - the name of an existing permanent file (1-8 ASCII characters) in words 1 and 2 and a BCD program name (1-6 characters) in word 3.

for nameptr,0 - word contents as above, but used for first program saved on a file.

loc 1 and loc 2 are the initial and final addresses of the program.

entry addr and load org are the entry address and load origin to be placed in the control block for the file. (They may be 0,0.)

tra is the location of the next instruction to be executed.

bufloc is the location of a 40-word (DSS204) or 64-word (other mass storage devices) buffer, its size depending on the block size of the device where the file resides.

This DRL writes an element (program) on a permanent file in standard program format. The element can then be loaded with a DRL RESTOR. If the saved program is the first on the file (nameptr,0), a catalog block and an available space block are initialized. If the program is saved on an existing file (nameptr,1), the catalog and available space blocks are updated. Up to nine programs can be written on a device having 40-word blocks; as many as 14 can be written on a device having 64-word blocks. The file name referred to must be in the AFT upon entry to the derail.

The entry address and load origin are not actually used by DRLSAV but may be required for a later RESTOR and are therefore included.

Upon return to the transfer address, the A-register contains 0 if the DRLSAV was successful. It contains a 1-bit in position 35 if the file is not large enough to accommodate the requested DRLSAV. The user may grow the file and attempt the derail again.

A user subsystem is aborted, with one of the following messages to the originating terminal, if an error occurs during a DRLSAV:

000ss BAD STATUS-DRL SAVE/RESTOR FILE

where ss is erroneous status

SAVE/RESTOR FILE NAME NOT IN AFT

REACHED CATALOG LIMIT DRL SAVE

Since catalogs are presently limited to one hardware block, number of programs which can be saved on a file is (block size/4) -1.

drl loc BAD DRL SAVE DATA LOC

where drl loc is the derail location

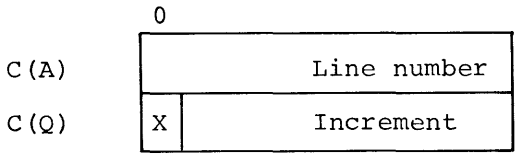
DRL SAVE-DO FIRST PROG SAVE BEFORE ADDED PROG

Nameptr,1 was attempted on new file.

DRL SAVE ON RANDOM FILE ONLY

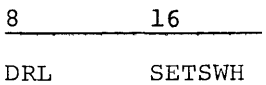
DRL SETLNO, SET LINE NUMBER/INCREMENT IN UST (octal 37)

8	16
DRL	SETLNO



This derail initiates automatic line-numbering mode. The specified line number and increment value are stored in the user status table for use by Line Service. An indicator for automatic line-numbering mode is set in .TFLG2 (in UST). A blank/no-blank indicator is set in .TFLG2 also, as specified by bit 0 of the Q-register. If the bit value (X) is 1, a blank is not supplied following the line number.

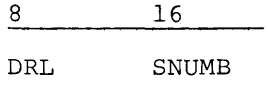
DRL SETSWH, SET SWITCH WORD (octal 10)



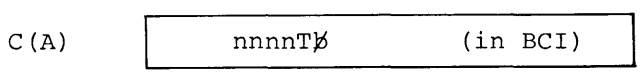
A 36-bit switch word (.LSWTH) is provided for each user. This derail provides a method of setting individual bits of this word. The value of the Q-register is ORed into the switch word. Thus, any bit that is on or true is set true in the switch word. Other bits are not disturbed. The value returned in 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.

Bits 0-17 are reserved for use of subsystems issued with the time-sharing system. Bits 18-35 are kept free for user-generated subsystems. (See Subsystem Switch Word paragraph.)

DRL SNUMB, OBTAIN SNUMB (octal 20)



If a subsystem wishes to spawn a batch job, a unique SNUMB is required. The TSS Executive obtains a SNUMB using a MME GESNUM routine. It is returned in the following format:



The trailing character T in the BCI form serves to distinguish this job and its output as a TSS-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 -- that is, DRL SPAWN (possibly notifying the user of the assigned SNUMB).

DRL SPAWN, PASS FILE TO BATCH PROCESSOR (octal 26)

```
8         16
DRL       SPAWN
ZERO      L(snumb),L(buffer)
```

C(A)	0	0 or 1

0 = immediate return
1 = return after batch job completes

C(Q)	File-name ptr.	0

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, numbered "1") 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.

L(snumb) specifies the location of the SNUMB obtained via DRL SNUMB. L(buffer) 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 - no PAT's for PASFIL
4 - SNUMB not given
5 - no program number available,
    try again

DRL STOPPT, STOP PAPER TAPE INPUT (octal 61)

```
8         16
DRL       STOPPT
```

This DRL causes a halt to paper tape input via a DATANET 355 when it is necessary to send an error message to the terminal. Because of interface and queueing system requirements unique to the DATANET 355, it is vital that these messages not be delayed until termination of tape input, as with the DATANET 30.

DRL SYSRET, RETURN TO SYSTEM (octal 40)

```
8         16
DRL       SYSRET
```

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

NOTE: This is not the normal return from a subsystem. The derail RETURN is the normal end-of-process return.

DRL TAPEIN, START PAPER TAPE INPUT (octal 27)

```
8         16
DRL       TAPEIN
ZERO      L(tally),L(char)
```

where L(tally) and L(char) are as in DRL KOUT.

*

This derail gives a subsystem the ability to print a message and start the input of a paper tape from the keyboard terminal. The TSS Executive will build the terminal input on the tape collector file (TAP*) in the format described in Section VI. When the paper tape input is complete, the subsystem receives control. The derail does not issue an X-ON character to start the tape reader (this is a user action). Note that some output is required, as in any terminal operation.

NOTE: Paper tape input must be interrupted for output of error messages by execution of DRL STOPPT, if time-sharing is via a DATANET 355.

DRL TASK, SPAWN A SPECIAL BATCH ACTIVITY (octal 63)

8 16

```
LDA       0,DU
DRL       TASK
ZERO      L(SSA buffer), L(file list)
```

where

SSA buffer is a 645-word buffer in the slave service area that provides 640 words for writing to the *J file.

File list is a list of files for which PATs must be copied to the SSA buffer. All files listed must be in rewound condition and exist in the user's AFT.

The format of the BUFFER PASSED TO DRL TASK is:

USER	TASK		
	*	0	*J PAT BODY
		.	
		.	
*		15	URGENCY
		.	
		.	
	*	50	TIME OF DAY
	*	55	STATION ID, SOURCE TYPE
*		56	SNUMB
	*	57	.SACT
*		58	ACTIVITY NAME .SACTY
	*	59	.STATI (0)
	*	60	.SNPAT
	*	61	.SPTBE
	*	62	.SNIO
		.	
		.	
*		305	I/O TIME LIMIT
	*	306	*J SCT, LINK
		308	LOCATION OF PAT BODIES
		.	
		.	
	*	570	LOCATION PAT PTRS
*		573	JOB TIME LIMIT
*		575	JOB I/O TIME LIMIT
	*	597	SCT POINTER (PUSH DOWN FILE)
	*	598	IOC TIME FOR THIS FILE (PUSH DOWN FILE)
	*	599	RANDOM (005200)
	*	600	NUMBER OF LINKS FOR PUSH DOWN FILE
	*	601	DESCRIPTORS FOR PUSH DOWN FILE
	*	602	
	*	608	.STQTM
*		609	.SWIT
	*	610	ACTIVITY TIME LIMIT
	*	611	GELOOP TIME LIMIT
	*	612	.SUID (USERID)
	*	614	MME GECALL
	*	615	ACTIVITY NAME
	*	618	ACTIVITY*, SNUMB (PREFIX WORD 30)
*		619	GELOAD LIMITS
*		620-629	IDENT CARD
*		630-639	COMMENT CARD
		640-644	USED BY DRL TASK (*J FILE DCW's)

TASK checks values of user supplied items and erases all other unused cells.

The format of the file list is as follows:

L	Number of files listed
+1	ASCII file name 1 (char. 1-4)
+2	char. 5-8
+3	File code 1 (BCI)
+4	ASCII file name 2 (char. 1-4)
+5	char. 5-8
+6	File code 2 (BCI)

.
.
.

Upon return a code appears in the A-register, as follows:

0 No error on DRL execution. (If this code appears, the Q-register contains the current status of the batch activity.)

1 Undefined file

2 No SNUMB

3 Duplicate SNUMB

4 No program number available

5 Activity name undefined

6 Illegal user limit (time,size, etc.)

7 Bad status on *J read or write

This derail was created to spawn a job to the Series 6000 FORTRAN compiler but may be used for other similar applications. In execution, DRL TASK first copies the PATs for the files in the file list to the designated SSA buffer. The buffer contents are then written on the first two blocks of the user's *J file. GEPOP then assumes control and enters the batch activity at core allocation level. Upon completion of the spawned activity, the DRL is again entered. The first two blocks of *J are read back to the user's SSA buffer. The derail then uses the current PATs now in the buffer to make any necessary updates in the PATs for the files in the file list.

If the user enters a BREAK or attempts a disconnect while the spawned activity is executing, it will be ignored until TASK has completed the final phase of its execution.

The following abort messages are possible with TASK:

ADDRESS OUT OF RANGE

The SSA buffer or file list address is outside subsystem limits.

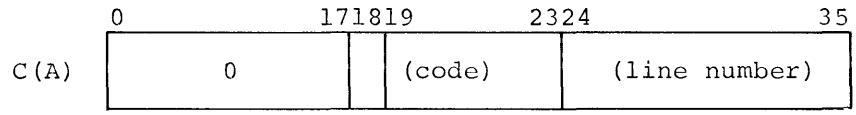
ILLEGAL ENTRY DRL TASK

See specific error code returned in A-register.

DRL TERMTP, TERMINAL TYPE AND LINE NUMBER (octal 23)

8	16
DRL	TERMTP

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, in BCI, in the rightmost 12 bits of the A-register and the terminal type code in bit positions 19-23. The codes for terminal type will be the same as defined by the GRTS programming system. Refer to Series 6000 GRTS Programming Reference Manual, for further details.



DATANET-30

Code	Terminal Type	
1 -	Reserved for system	
2 -	115 voice grade	
3 -	115 Telpak	
4 -	TTY	
5 -	DATANET 760 4 lines/screen	46 characters/line
6 -	DATANET 760 8 lines/screen	
7 -	DATANET 760 16 lines/screen	
10 -	DATANET 760 26 lines/screen	
11 -	VIP775 22 lines/screen	
12 - 17 -	Reserved for system	
20 -	2741	
21 - 60 -	Reserved for system	
61 - 77 -	Reserved for customer use	

DATANET-355

Code	Terminal Type	
1 -	Reserved for system	
2 -	Reserved for system	
3 -	Remote computer	
4 -	TTY	
5 -	DATANET 760 4 lines/screen	46 characters/line
6 -	DATANET 760 8 lines/screen	
7 -	DATANET 760 16 lines/screen	
10 -	DATANET 760 26 lines/screen	
11 -	VIP765/775 22 Lines/screen	
12 - 13 - 17 -	Reserved for system	
20 -	2741	
21 - 24 -	Reserved for system	
25 -	Mass store link	
26 - 27 -	Reserved for system	
30 -	MRS200 (document handler)	
31 -	DRD200 (document handler)	
32 -	DRD236 (document handler)	
33 - 60 -	Reserved for system	
61 - 77 -	Reserved for customer use	

DRL TIME, OBTAIN PROCESSOR TIME AND TIME OF DAY (octal 21)

```
8          16
DRL        TIME
ZERO       L(date)
```

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 entered, in ASCII code, with slashes inserted between the values in the following form:

DATE	M	M	/	D
+1	D	/	Y	Y

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.

Mass Storage File Activity Derails

TSS file usage is discussed in the paragraphs beginning on page 3-54. This discussion will be helpful to the subsystem designer who wishes to adhere to standard practice in using the file activity derails.

DRL DEFIL, DEFINE AND ACCESS A TEMPORARY FILE (octal 6)

```
8          16
DRL        DEFIL
ZERO       L(arg),L(stat)
```

arg	Filename, (characters 1-4)	in ASCII
arg+1	Filename, (characters 5-8)	

	0 5	6 16	17	18	19 35
arg+2	a		b	c	d

where:

a = device type as follows --

00 - DSS 270 (disk)
01 - DSS 200 (disk)
02 - DSS 167 (disk)
03 - MDS 200 (drum)
04 - MSS 800 (disk)
05 - DSS 170 (disk)
06 - DSS 180 (disk)
07 - DSS 181 (disk)

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 first word:

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

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	0-5	Device type
	6-17	Number of words per physical block
if bit 18 = 0		bits 24-35 = number of links in the file
if bit 18 = 1		bits 24-35 = number of blocks in the file
if bit 19 = 0		Linked file
if bit 19 = 1		Random file
if bit 20 = 0		Temporary file
if bit 20 = 1		Permanent file
	21-23	Unused

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

DRL DIO, DO I/O ON USER'S FILE (octal 1)

```
8          16
_____________________

DRL        DIO
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 file name 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 (Refer to Series 6000 Comprehensive Operating Supervisor, for the status codes.)

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. 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.

Standard statuses are returned, except that Device Busy has special meaning. It is used to notify the user that his file name is not yet defined. (True Device Busy status is never returned by the GCOS file system.) Logical end-of-file is returned as major status 17.

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.

The requesting subsystem is also aborted, with an error message sent to the terminal, if the terminal user does not have the necessary permissions. In particular, if a read is requested and the file does not have READ permission, a check is made to see whether EXECUTE permission was granted on the file when opened. If it was and if the subsystem is BASIC, FORTRAN, or CARDIN, the file read is executed. Otherwise, the read request is denied.

DRL FILACT, PERMANENT FILE ACTIVITIES (octal 36)

Grouped under DRL FILACT are the following permanent file functions:

Create Catalog (CC)
Creat File (CF)
Access File (AF)
Purge Catalog (PC)
Purge File (PF)
Release File (RF)
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. These are handled by DRL RETFIL.

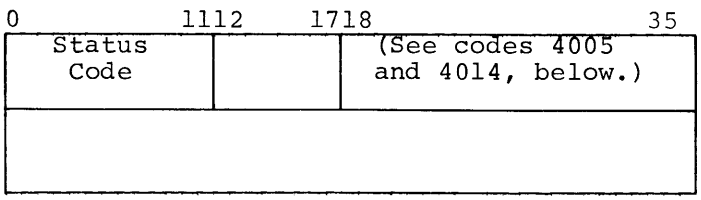
The following parameter descriptions are common to most of the DRL FILACT calling sequences. Following a calling sequence in which one or more of these parameters differ from the common layout, the description of such parameters is given.

(1) buffer:

```
buffer    BSS    380
```

This buffer is required in all cases as File System working storage.

(2) status return:



NOTE: The status return words are automatically zeroed by the File System when initialization takes place for the selected permanent file function.

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

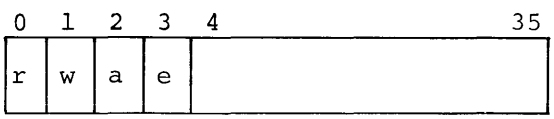
Status codes:

```
4000   NO ERRORS

4001   NAME NOT IN MASTER CATALOG
4002   I/O ERROR ON DEVICE XXX SA = NNN......NNN
4003   PERMISSIONS DENIED
4004   FILE BUSY: TRY LATER
4005   INCORRECT CAT/FILE DESCRIPTION AT AAA......AAA
4006   LLINK SPACE EXHAUSTED, DEVICE XXX
4007   UNDEFINED DEVICE YYY ZZZZZZ
4010   LINK SPACE EXHAUSTED, DEVICE XXX
4011   NON-UNIQUE NAME
4012   SIZE REQUESTED LS THAN ALLOCATED
4013   SPACE REQUEST GR THAN ALLOWED
4014   PASSWORD REQUIRED AT AAA......AAA
       PASSWORD AAA......AAA AT AAA......AAA INCORRECT
4015   I-D-S FILE IN ABORT STATUS
4016   I-D-S FILE IN RECOVERY STATUS
4017   SEEK ERROR ON DEVICE XXX SA = NNN......NNN
4020   FAILURE IN NAME SCAN (IMP.)
4021   UNDEFINED DEVICE (IMP.)
4022   DEVICE LINK TABLE CHKSUM ERROR
4023   INCONSISTENT FSW BLOCK COUNT
4024   INTERNAL LINK TABLE CKSM ERROR
4025   REQUESTED ENTRY NOT ON-LINE
4026   NON-STRUCTURED FILE ENTRY
4027   FILE IN EFFECTIVE STATUS
4030   ILLEGAL PACK TYPE
4031   ACCESS GRANTED TO I-D-S FILE
4042   INVALID FILE CODE OR PAT PTR
4043   INVALID CATALOG BLOCK ADDRESS
4044   PERMISSION DENIED - SHARED FILE
4045   INVALID SPACE IDENTIFIER
4051   CHECKSUM ERROR - DEVICE XXX SA = NNN......NNN
4052   DEVICE XXX RELEASED

WHERE: XXX           =DEVICE NAME (ST1,DS1,...)
       NNN......NNN =OCTAL REPRESENTATION OF THE SEEK ADDRESS
       AAA......AAA =12 BCD CHARACTERS OF THE CATALOG ELEMENT
                     IN ERROR
       YYYY          =TYPE /OR/ NAME
       ZZZZZZ        =DEVICE NAME OR CLASS OF DEVICE
```

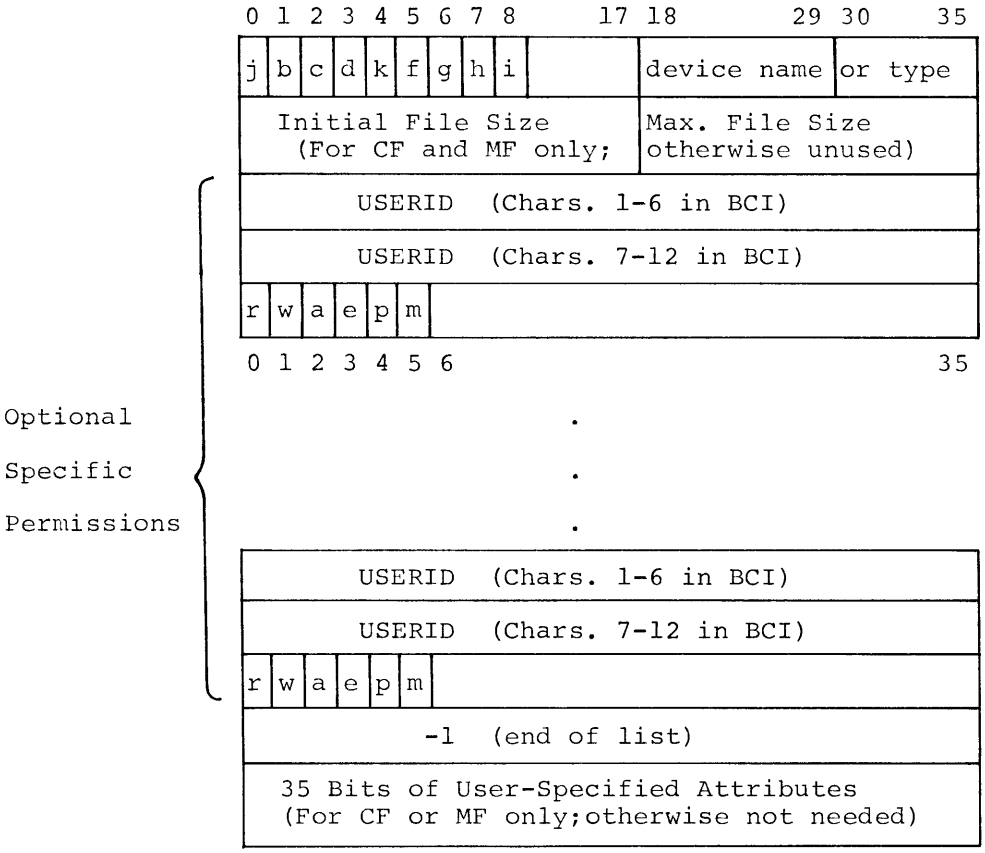
(3) permissions (assigned or requested)



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:

j = contiguous allocation desired
(Create File only)
b = random file type
(Create File only)
c = USASCII file (Create File only)
d = I-D-S file
k = allocation in 320-word blocks
(Create File only)
f = tape file
g = DSU 167
h = I-D-S attributes file
i = user-specified attributes
(Create File or Modify 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

Device name or type is defined as:

Specific (named) device, in BCI (for example, DS1, where "DS1" would have been assigned as a specific device name in the installation's GCOS Startup deck)

Type of device (in bits 30-35) -

```
00 = DSS 270 (disk)
01 = DSS 200 (disk)
02 = DSS 167 (disk)
03 = MDS 200 (drum)
04 = MDS 300 (drum)
05 = DSS 170 (disk)
06 = DSS 180 (disk)
07 = DSS 181 (disk)
```

or where:

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

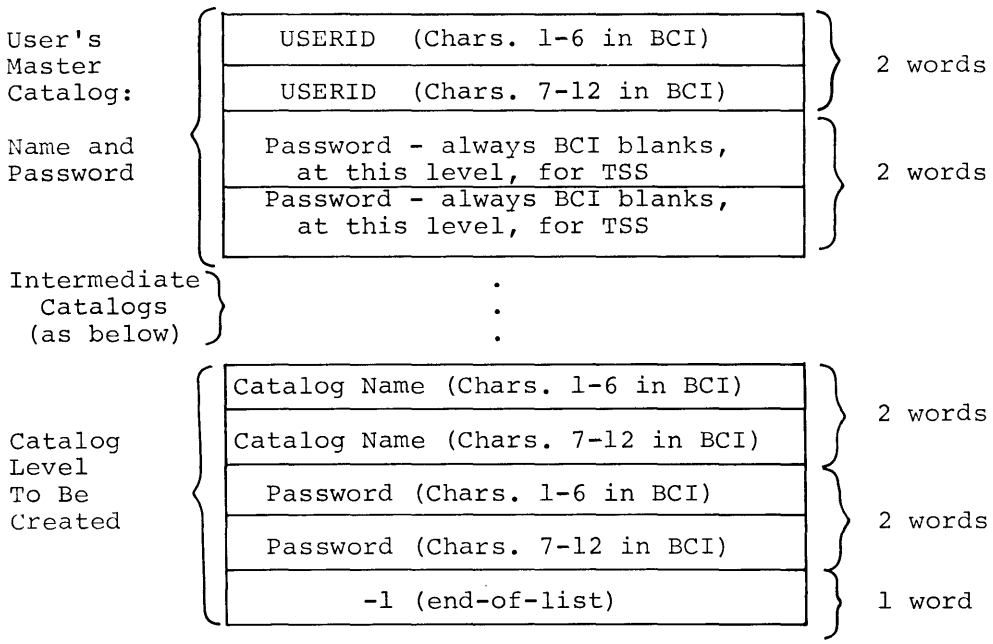
FILACT, CREATE CATALOG FUNCTION

```
8       16
DRL     FILACT
ZERO    0,L(arglist)
ZERO    2,L(buffer)
```

where:

```
arglist  ZERO  L(status-return),0
         ZERO  L(cat/filedescr),L(permission)
         ZERO  L(options)
```

cat/filedescr:



(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/filedescr table (that is, the complete catalog string).

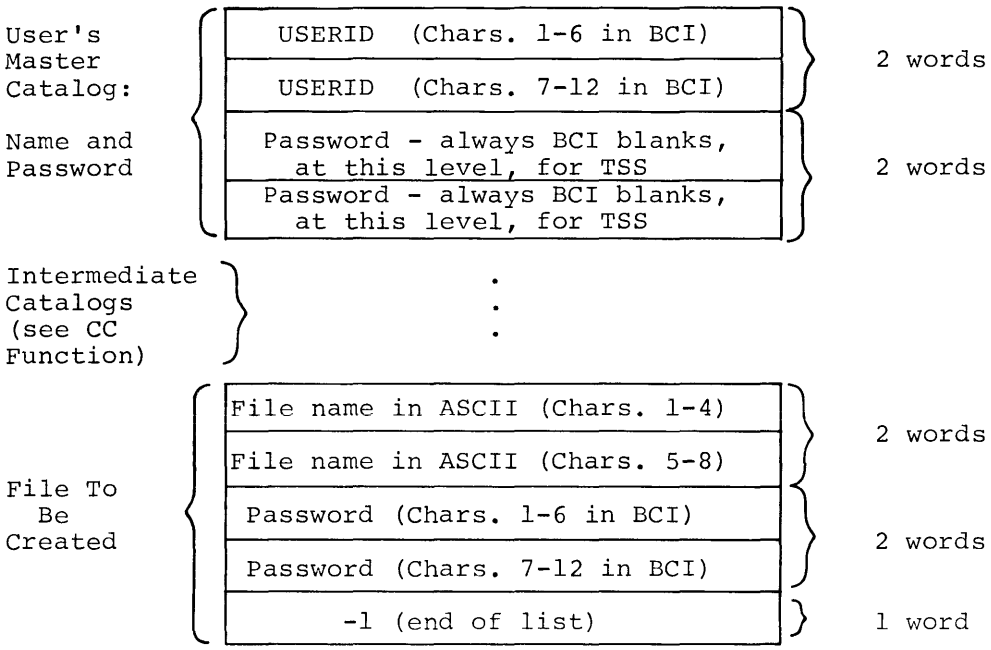
FILACT, CREATE FILE FUNCTION

```
8       16
DRL     FILACT
ZERO    0,L(arglist)
ZERO    3,L(buffer)
```

where:

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

cat/filedescr:



(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 USERID of the current terminal user is to be filled in by the derail processor.

options:

Includes the appended word of user-specified attributes, following the -1 word, if the i bit of options + 0 is on.

NOTE: If this bit is on, bits 1-35 of the attributes word are placed in the file descriptor and are returned on a subsequent file access.

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

FILACT, ACCESS FILE FUNCTION

```
8          16
DRL        FILACT
ZERO       L(altname),L(arglist)
ZERO       4,L(buffer)
```

where:

arglist ZERO L(status return),1 for random/0 for linked file[1]
ZERO L(cat/filedescr),L(permissions)

altname

altname in ASCII, or all
zeros if no alternate naming desired

2 words

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.

status return:

0 1	11	12	18	35
Status			pd	
n	User-specified attributes			

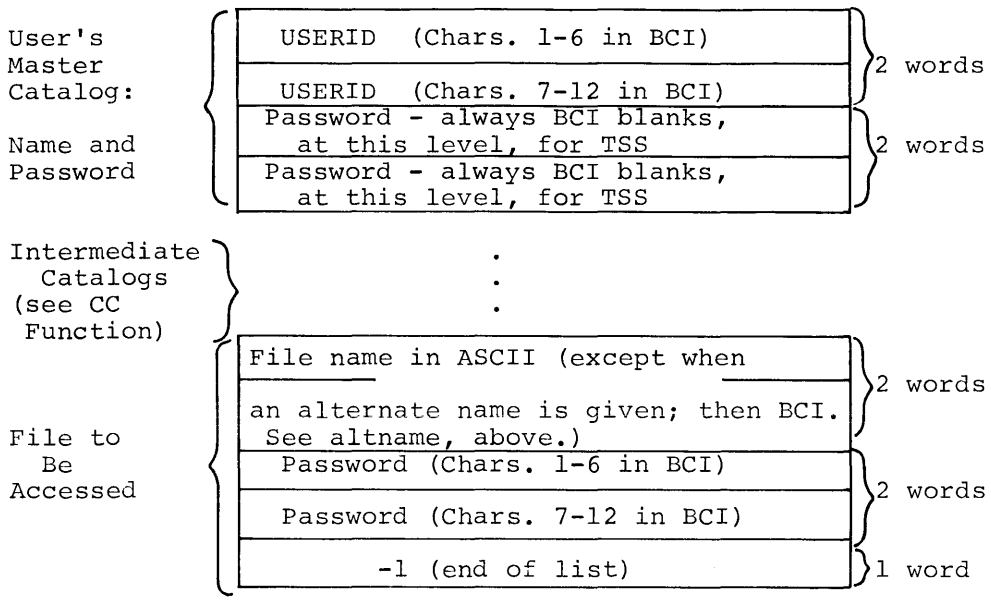
2 words

where-:

pd = 400300 (Permissions Denied status) when the file is already open and the permissions are not equal to or a subset of those current (See following discussion.)
n = 0 for a null file
n ≠ 0 for a non-null file

[1]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.

cat/filedescr:



(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 USERID 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 permissions requested. If the file is already open and the new permissions requested in arglist are equal to, or a subset of, those already granted, a status code of 4037 (Duplicate Name in AFT) is returned. If the file is already open and the requested permissions are not equal to, or a subset of, those current, a status code of 4003 (Permissions Denied) is returned left-justified in bits 18-35, with the 4037 status code in bits 0-11.

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	0-5	Device type
	6-17	Number of words per physical block
	if bit 18 = 0	bits 24-35 = number of links in the file
	if bit 18 = 1	bits 24-35 number of 320-word blocks in the file
	if bit 19 = 0	Linked file
	if bit 19 = 1	Random file
	if bit 20 = 0	Temporary file
	if bit 20 = 1	Permanent file
	21-23	Unused

The status return word pair indicates, in addition to the status code in the first word, whether or not the accessed file is null (empty) -- bit 0 of the second word. The 35 bits of the user-specified attributes are also returned in the second word.

FILACT, PURGE/RELEASE CATALOG/FILE FUNCTION

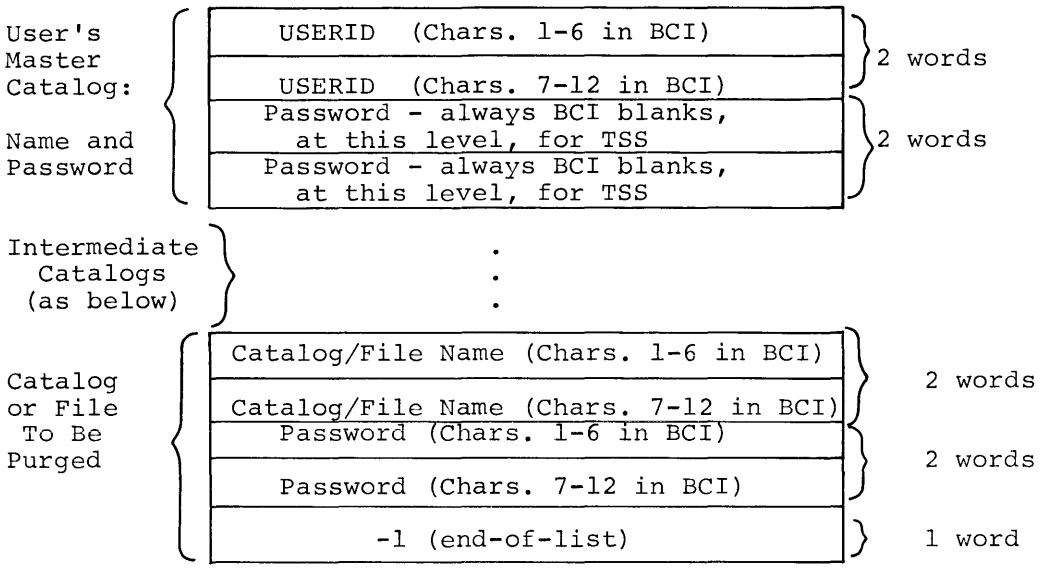
```
8         16
-------------------------
DRL       FILACT
ZERO      0,L(arglist)
ZERO      n,L(buffer)
```

where:

```
arglist   ZERO  L(status return),0
          ZERO  L(cat/filedescr),0

n (function number)
          n =  8 - Purge Catalog
          n =  9 - Purge File
          n = 22 - Release File
```

cat/filedescr:



(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 USERID of the current terminal user is to be filled in by the derail processor.

At the file level, this function deletes the file descriptor from the file system and either releases the corresponding file space (n=22) or zeros and releases the space (n=9). At the catalog level, this function deletes the named catalog and all catalogs and files subordinate to it.

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

FILACT, MODIFY CATALOG/FILE FUNCTION

```
8         16
DRL       FILACT
ZERO      0,L(arglist)
ZERO      n,L(buffer)
```

where:

```
arglist  ZERO  L(status return),0
         ZERO  L(cat/filedescr),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, 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 (that is, 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, not just additions.

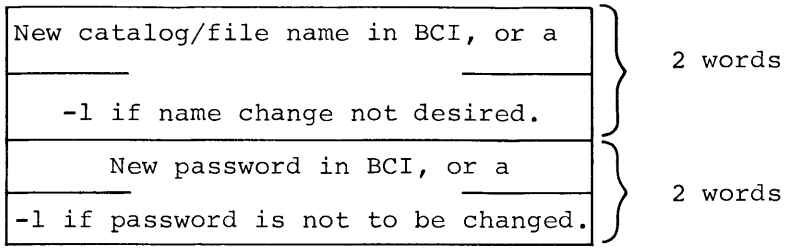
(2) delete all specific permission (for one user) - all permission bits must be zero.

(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.

(5) replace the user-specified attributes bits in the file descriptor with those (1-35) of the appended attributes word, if bit i of options is set.

new name/password:



This function will modify a catalog or file descriptor, depending upon the function number specified. Unlike alternate naming in Access File, the changes made by this function are permanent. However, for a file that has already been opened, subsequent reference to the file must still be by its original name. The reason is that, even though the modification has taken place, the AFT entry has not been altered.

DRL FILSP, SPACE A LINKED FILE (octal 13)

```
8         16
DRL       FILSP
ZERO      L(fileid),L(n)
ZERO      L(stat),0
```

where:

fileid (2 words) contains the file name in ASCII

n contains the number of 320-word blocks, n, 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 and error message is sent to the terminal.

DRL GROW, GROW A PERMANENT OR TEMPORARY FILE (octal 50)

```
8         16
-------------------------
DRL       GROW
ZERO      L(n), L(fileid)
ZERO      L(buff),L(stat)
```

where:

n(0-17) contains the number specifying the unit of growth. If bit 0 is On (bit 0 = 1) the unit is in links; if bit 0 is Off (bit 0 = 0) the unit is in 320-word blocks. If n is zero the system determines the growth rate.

file name (2-words) is the name of the file in ASCII.

buffer is a 158-word work area required for permanent files only.

status is a two-word status return area.

This derail is used to grow an already accessed (opened) permanent or temporary file up to its maximum size limit.

The number n in the location pointed to by the first argument specifies the growth rate or, if it is 0, specifies that the system is to set the growth rate. (For a temporary file, the system-controlled rate is 1 link.)

The contents of bit position 0 in the location pointed to by the first argument are used to specify the unit of growth:

0 - The unit of growth is n 320-word blocks for a permanent file, or the number of blocks is rounded up to make full links for a temporary file. For example, if n=15, a permanent file is grown by 15-block units; a temporary file is grown by 2-link units (1 link=12 blocks).

1 - The unit of growth is n links for either type of file (1 link=12 blocks).

On return:

```
stat (0-11) = 4000 - no errors
            = 4002 - I/O error - cannot proceed
            = 4010 - Link space exhausted
            = 4020 - Failure in name scan
            = 4024 - Internal link table checksum error
            = 4040 - No PAT space available
```

DRL MORLNK, ADD LINKS TO TEMPORARY FILE (octal 34)

```
8         16
-----------------------------
DRL       MORLNK
ZERO      L(links),L(fileid)
return
```

where:

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

fileid (2 words) contains the file name in ASCII

On return, links contains the following:

bits 0-5 - error indication, as follows:

Bit (set)		
	0	PAT full
	1	Link space exhausted
	2	File is permanent file
	3	File name not in AFT
	4	No links requested
	5	(Not currently used)

bits 18-35 - number of links obtained; will be number requested or 0. If 0, see error indication in bits 0-5.

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

NOTE: The programmer may use DRL GROW in place of DRL MORLNK and the system will determine if the file is permanent or temporary.

DRL PART, PARTIAL RELEASE OF TEMPORARY FILE (octal 47)

```
8         16
DRL       PART
ZERO      L(fileid),n
```

where

fileid (2 words) contains the file name in ASCII

n is the number of links to be released

The derail is used to release a portion of a temporary file. If the specified number of links, n, is greater than or equal to the number of links in the file, the file is reduced in size to one link.

On return, the result is indicated in the upper half of the A-register:

C(AU) = 0 if request is satisfied

C(AU) = 1 if file is nonexistent or not a temporary file

If the request is satisfied, the file is in rewound position.

DRL RETFIL, RETURN A FILE (octal 14)

```
8         16
DRL       RETFIL
ZERO      L(fileid),L(buff)
```

where:

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

buff (BSS 380) is a work area used by the system. This parameter is required only for permanent files.

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 is deleted from the AFT.

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

DRL REW, REWIND A LINKED FILE (octal 12)

```
8          16
-------------------------
DRL        REW
ZERO       L(fileid),L(stat)
```

where:

fileid (2 words) contains the file name 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.

DRL SWITCH, SWITCH TEMPORARY FILE NAMES (octal 53)

```
8          16
-------------------------
DRL        SWITCH
ZERO       L(fileid1),L(fileid2)
Error return
Successful return
```

where:

each fileid (2 words) contains a file name in ASCII

This derail is used to switch the names assigned to two temporary files specified by fileid1 and fileid2. Note that this effectively accomplishes an exchange of the contents of the two files.

On an error return:

C(A) = 0 if either file is nonexistent

C(A) = nonzero if either file is not a temporary file

TSS FILE USAGE

Temporary User Files Assigned by TSS

The usage of standard temporary user files is described here on the basis of what is done by the Honeywell-supplied TSS subsystems, 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 TSS. All standard temporary files should have at least one asterisk (ASCII 052) in their names to differentiate them from user-created files.

There are two standard temporary files for each terminal user: the collector file, SY**, and the current file, *SRC.

COLLECTOR FILE (SY**)

The SY** 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 Section IV). This is the raw data 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 TSS, provided that the selected subsystem used build mode for the collection of new or additional input destined for a source file. Examples of SY** input are the numbered language statements in BASIC and the text entries in EDITOR. SY** file format is described in Section VI.

CURRENT FILE (*SRC)

An *SRC file is assigned to a user by the OLDN (Old-New) subsystem. In the BASIC subsystem, for example, the OLDN subsystem is called by the first primitive of the startup procedure. OLDN produces the old-new file request sequence -- OLD OR NEW- (and, conditionally, OLD NAME-).

The current 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 collector (SY**) file contains all the raw input, including any mistakes and corrections, other than keying errors corrected by @ or CRTL/X.

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

For 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 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 feature leaves the OLD (permanent) file as backup copy (except when using OLDP/NEWP commands).

The format of the *SRC file is described in Section VI.

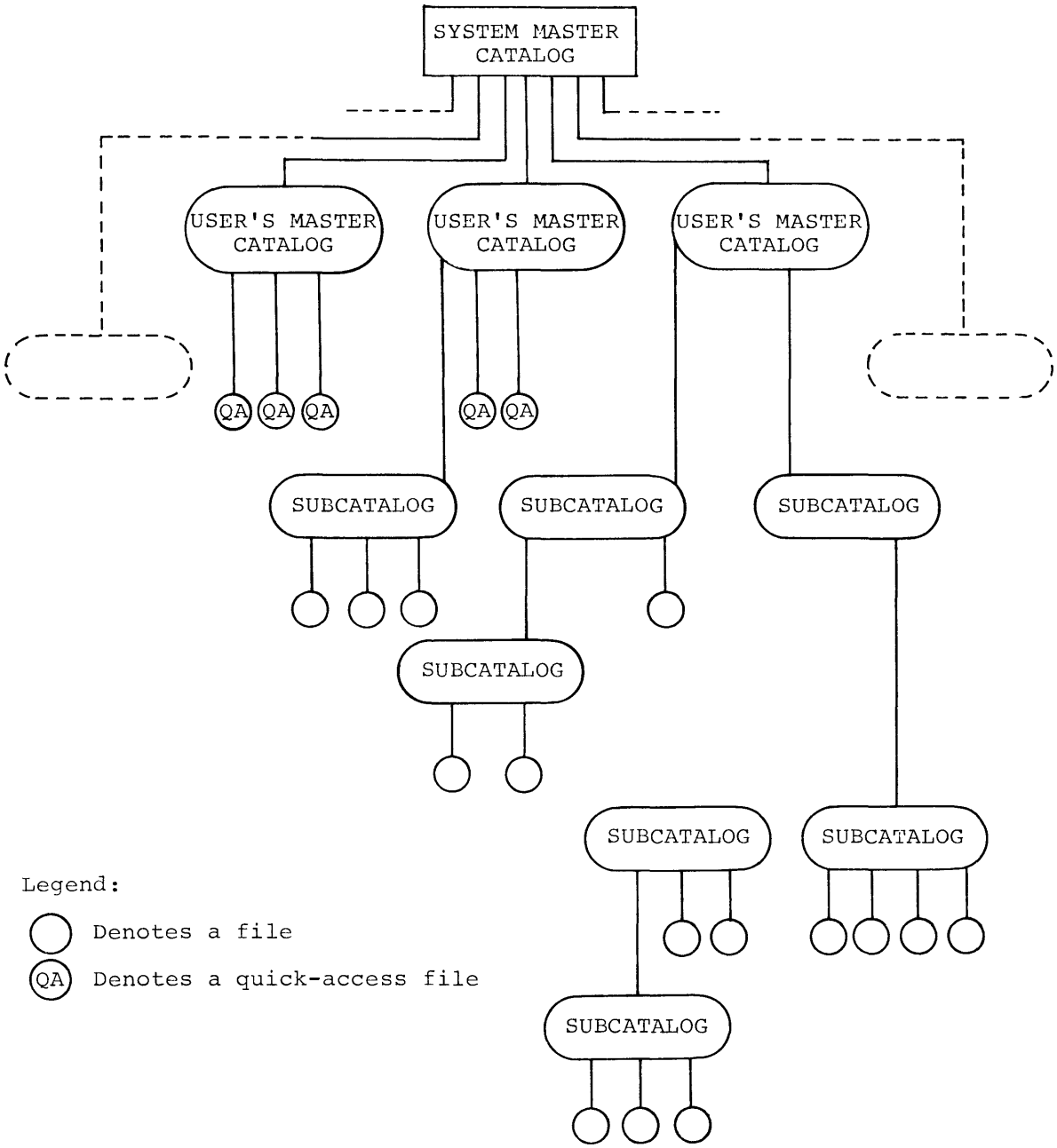
Permanent Files Assigned by User

TSS never assigns permanent file space to a user unless specifically told to do so by that user. Permanent files are handled by the File System, which is common to all programs operating under GCOS. Permanent time-sharing files are ordinarily created by using SAVE or PERM commands; otherwise, they are created via the ACCESS subsystem or a batch FILSYS activity.(See also NEWP/OLDP commands.)

STRUCTURE OF THE FILE SYSTEM

The GCOS File System is described in the Series 6000 GCOS File System. The main points of interest to the TSS user are repeated here.

The GCOS 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. (See the diagram on the next page.)



The master catalogs for each user are identified by USERID. A USERID must be unique within the system. All subcatalog and files 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.

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.

In the GCOS 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. 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, 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 the passwords.

The allowable permissions are:

Read	-- allows the user to execute RUN and also LIST, FDUMP, etc., on the file.
Write	-- allows a file to be written.
Append	-- (presently treated as Write)
Execute	-- allows the user to execute RUN on the file. He may not execute LIST or any other operation that allows him to see the file's contents. If the file is a source file, he is not allowed to see the resultant object file. In FORTRAN or BASIC, if the file is a source file and the only permission is Execute, a RUN command cannot be used to build an object file.
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)
Exclusive	-- the current user has exclusive use of this file, blocking out all other users.

Multiple concurrent reading or executing of a file is allowed by the File System, but multiple writing or appending is not.

USER'S CONTACT WITH THE FILE SYSTEM

The terminal user's contact with the GCOS File System is mainly through the Old-New (OLDN) and Save/Resave-Purge (SAVE) subsystems.

OLDN, when OLD is selected, writes the contents of the permanent (OLD) file onto the user's current file, *SRC. SAVE or RESAVE writes the contents of *SRC onto the named permanent file. (See the description of OLDN in Section VII.) In 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.

Available File Table (AFT) Usage

TSS maintains an available file table (AFT) for each user. Before any I/O can be done on a file, an entry for that file must be placed in the AFT.

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

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 (that is, other than alphabetic, numeric, period, or hyphen) in the file name. The asterisk is used by Honeywell-supplied subsystems. Since special characters are not allowed in permanent file names defined from a terminal, any conflict is avoided.

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 the file. All user's files in the AFT are released upon termination.

PERMANENT FILES

DRL FILACT function number 4 (Access File) places the file entry in the AFT and sets the file "busy" for the permissions 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 File).

DRL RETFIL (return a file) removes the file entry from the AFT and sets it "not busy" with respect to the current user.

FILE I/O

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

DRL DIO -- Reads or writes a file

DRL FILSP -- Positions a file forward or backward

DRL REW -- Positions a file to its beginning

DRL MORLNK -- Increases the size of a temporary file

DRL GROW -- Adds space to a permanent file, up to its maximum size

DRL PART -- Releases a portion of a temporary file

DRL SWITCH -- Switches two temporary file names

These are the only file I/O derails that affect or relate to the AFT and are most often used by subsystem programs. The others (all of the FILACT functions except Access File) affect only the GCOS File System.

SECTION IV

COMMAND LANGUAGE AND PRIMITIVES

In the design of many time-sharing programs, it is desirable to recognize unique command language and initiate process sequences based on these commands. The user can define command language independently for his subsystem. Each command word has associated with it a list of primitives that are specified by the system designer. These primitives are interpreted by TSS to control the processes implied when the command is recognized. Command language is recognized by TSS only when the subsystem is in the build mode of keyboard input. (See "Keyboard Input Modes" later in this section.)

The command-language/primitive lists are incorporated in the TSS communication region (block .TPCOM), along with the primary portion of the program descriptor. (Refer to Section II.)

The primary portion of the program descriptor is arranged with those for other subsystems in a contiguous block for rapid scanning. Program descriptor format is shown on the next page. The process of assembling the program descriptor portions into the communication region is described in Section V.

Program Descriptor Proper: (block .TPRGD)

subsystem name in ASCII	
program size[1]	load size (nonzero block size)[1]
entry point[1]	parameters[1]
seek address[1]	initial load address[1]
command-language pointer	no. of words in command language
program statistics[1]	

Command-Language List: (block .TPCOM)

command word 1	word-pair
scan mask 1	
command word 2	word-pair
scan mask 2	
.	
.	
command word n	word-pair
scan mask n	
primitive pointer 1	in same sequence as the word-pairs above
primitive pointer 2	
.	
.	
primitive pointer n	
startup-primitive pointer	always last (that is, (n*3)+1 from top of command language list)

Primitives

primitives - one word each	arranged in normal instruction-execution sequence
↓	

[1] Supplied by the system

Program Descriptor 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 nonzero word in the program. (Supplied by the system.)
Parameters	Flags defining the type of program: privileged, master, 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 TSS Executive regarding usage of this program.
Command Words and Masks	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	Addresses 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 in this section 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

Bits (set)	18-31	-- Not defined
	32	-- Program stored on secondary program file #Q
	33	-- Patches are in patch table
	34	-- Do not set base register on dispatch
	35	-- Permitted privileged derails

KEYBOARD INPUT MODES

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

In the direct mode, the subsystem program in execution requests input via the keyboard I/O derails. The requested input passes directly to the subsystem, and no scan or interpretation is made by the TSS Executive. Thus, while in this mode, there is no recognition of commands by TSS. 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 TSS. All input is collected and written to a system-assigned, temporary 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 (BIN).

PRIMITIVES

Format of Primitives

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.

Note that a in CALLP is the symbolic address of the program descriptor for the desired subsystem. It may or may not correspond to the ASCII subsystem name (determinable by inspection of the .TPRGD listing).

Primitive Descriptions

- CALLP a (where a is a program-descriptor location)

  This primitive transfers control to the subsystem program descriptor located at a. TSS finds the startup sequence in the program descriptor of the called subsystem and takes its next primitive from this list. CALLP may occur in any list of primitives of a program descriptor and will interrupt the execution of primitives from the current list. However, it is normally necessary that control be returned to the previous level after the series of functions performed by the called program is 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, TSS reads and accumulates data on a collection file (SY**) for a given user. Each line of input is scanned for the command language associated with the subsystem. If no command language is found, TSS accumulates the input in a buffer and dumps it when required to the user's input collector file. If a command word is recognized, TSS Executive does the necessary housekeeping before the command is executed. Not 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. TSS 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, if 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 Section III). 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 subsystem switch word. 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. The settings, 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 subsystem switch word 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 word bit is set true (on).

STARTUP PROCEDURE

One set of primitives is always part of the descriptor 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.

USE OF EXISTING SUBSYSTEMS

When a new subsystem is to be added to TSS, the implementor may wish to incorporate a command language within the subsystem. Instead of processing commands within the new subsystem (that is, using direct mode) or writing additional new subsystems, existing subsystems which perform the desired functions may be utilized.

The command word selected to activate an existing subsystem may be the same as that used by another calling subsystem (for example, LIST in BASIC and FORTRAN) or it may differ. The command-language list of the selected subsystem determines the command-language words, whereas the corresponding primitive pointers and the primitive list determine which additional subsystems are called. For example, the standard LIST subsystem might be activated as a result of the command word DISPLAY. It is recommended, however, that for overall system consistency new command language conform to standard usage wherever possible.

Although the command-language/primitive-pointer list must be unique for a subsystem, its primitive sequences need not be. That is, appropriate portions of other subsystems' primitive lists may be utilized. When preparing the program descriptor and lists, refer to the communication region module (TSSA) listing for existing primitive sequences that may be used.

In a subsystem which uses the BIN (Build Input) build mode primitive in the usual fashion, any primitive sequence that executes a CALLP must first test for data on SY** (switch-word bit 17 on) and call an SY** editor (see BSED in Section VII, for example). Otherwise, any new data on SY** prior to the command is lost.

An inspection of Example 1 under Examples of Program Descriptors in this section and the command lists of the major Honeywell-supplied subsystems will suggest usable working and command-processor subsystems.

Just as in the major system-selectable-only subsystems such as BASIC or FORTRAN, user-designed subsystems may have a null program portion, as opposed to the null command-list type of subsystem illustrated in Example 2 following. While the latter's primitive list contains only an EXEC and corresponding POPUP primitive, the former type of subsystem contains no EXEC primitive but only CALLPs. Thus, no corresponding program portion exists. (There is no logical necessity for this type of subsystem; it is rather a matter of programming convenience, with a view to greater generality.)

PROGRAM DESCRIPTOR EXAMPLES

Two examples of subsystem program descriptors follow.

Example 1

In this example, the new subsystem being integrated into TSS is a compiler/loader called MYLAnguage with a line-number-dependent source language similar to BASIC. Associated with the subsystem are two command words which cause execution of the subsystem program -- COMP (for compile-only) and GO (for compiler and/or go). (Either or both of these commands could have variable-field options for direct interpretation by the MYLAng subsystem.) However, the implementor wishes his users also to be able to call a permanent file from the File System, to start a new file, to list a file, to save or purge a file, and to escape to the subsystem-selection level (SYSTEM?).

Since this is a line-number-dependent subsystem and the implementor chooses to utilize the standard TSS temporary files SY** and *SRC (refer to Sections III and VI), Honeywell-produced standard subsystems exist to perform these auxiliary functions: The LIST subsystem, the OLDN subsystem (for OLD/NEW), and the SAVE subsystem (for SAVE/PURGE). The standard line-number editor, BSED (Basic Edit), also is available for editing keyboard input on SY** and merging the input onto *SRC (refer to Section VII). The implementor reasonably chooses to retain the standard TSS command words -- LIST, OLD, NEW, SAVE, and PURGE -- to activate the command-processor subsystems. He also uses the standard escape command, DONE, which simply causes a POPUP to the calling level (the TSS Executive).

The program descriptor and command-language/primitive list are shown on the following page. The startup procedure calls the OLDN subsystem (program-descriptor location symbol OLDNEW) with bit 15 of the subsystem switch word off, indicating that the OLD OR NEW request message is to be issued. The OLD and NEW commands also call OLDN, but with bit 15 on to bypass the issuance of this message.

Before any command processor is called that destroys the current file (all but OLD and NEW), bit 17 of the subsystem switch word is tested to see if SY** has received any build input preceding the last-received command. If build input has been received, BSED is called to write it to *SRC before the appropriate subsystem is called. MYLA utilizes the user's bit 18 of the switch word to differentiate, for the subsystem program, between an execution resulting from COM (on) or GO (off). (The subsystem program itself could make this differentiation by direct inspection of the command -- via DRL KIN -- but the method employed would be more economical, unless the DRL KIN is also required otherwise for inspection of the variable field.)

If the subsystem were to provide for the unlikely case of build-input immediately preceding a DONE command, a test of SY** (bit 17) and a conditional call of BSED would be indicated just prior to the POPUP primitive (P6). In a nonhypothetical situation, the automatic dumping of SY** on overflow (dummy command word of OCT 004004004001) would have to be provided for as well.

Program Descriptor

M Y	L A
MYLACL	8

.
.
.

Command Language

MYLACL

C	O	M	P
000	000	000	000
G	O		
000	000	777	777
L	I	S	T
000	000	000	000
O	L	D	
000	000	000	777
N	E	W	
000	000	000	777
S	A	V	E
000	000	000	000
P	U	R	G
000	000	000	000
D	O	N	E
000	000	000	000
	P1 (comp)		
	P2 (go)		
	P3 (list)		
	P4 (old)		
	P4 (new)		
	P5 (save)		
	P5 (purge)		
	P6 (done)		
	P7 (startup)		

Primitive List

P1	STRUE	18,P1.1
P1.1	IFALSE	17,P1.2
	CALLP	BSED
P1.2	EXEC	
	BIN	
P2	STFALS	18,P1.1
P3	IFALSE	17,P3.1
	CALLP	BSED
P3.1	CALLP	LIST
	BIN	
P4	STRUE	15,P7.1
P5	IFALSE	17,P5.1
	CALLP	BSED
P5.1	CALLP	SAVE
	BIN	
P6	POPUP	
P7	STFALS	15,P7.1
P7.1	CALLP	OLDNEW
	BIN	

Example 2:

The 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, and a primitive to cause control to return to the previous level. This sequence is adequate for many subsystems placed into TSS. 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
```

SECTION V

PLACING SUBSYSTEM PROGRAMS IN THE SYSTEM

Subsystem programs may be placed in the time-sharing system either permanently or temporarily. The temporary placement of a subsystem does not require an edit of TSS 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 SUBSYSTEM 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 GCOS.
3. Prepare and assemble the program descriptor and command-language/primitive list into the TSS communication region.
4. Modify and reassemble the TSTART module of TSS to cause the subsystem to be included in TSS initialization.

A summary of step 1 follows; steps 2 through 4 are discussed in detail.

Writing the Subsystem Program

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

- An unused subsystem data area of 100 (decimal) words must precede the executable subsystem coding, for use by TSS.

  The 64 words normally reserved by the loader can be used as part of the required area. Therefore, the first storage definition statement must be at least a BSS 36.

- If it is desired that the subsystem process any or all of the faults from which recovery is possible under TSS, coding to store the fault-vector transfers must be included in the subsystem. They cannot be loaded and must be reinitialized after each recovery attempt.

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

- For coding convenience, two macros are available for general use. These TSS macros are called by LODM .G3TSn (see Appendix A). Then the macro call .SSDRL provides 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.

  Appendix A, System Macros, describes these and other available TSS macros.

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

Editing Subsystem Program to GCOS

Editing to GCOS 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 decks
     .
     .
$  DKEND
$  ENTRY      entry point   (optional)
$  EXECUTE
$  ENDLD
```

The TSS catalog names are prefixed by .TS so the user must select for his subsystem only a three-character identifier (xxx) that is unique among the TSS subsystems. When placed with the rest of the subsystems the decks are edited into GCOS.

Assembling the Program Descriptor

The program descriptor and command-language/primitive lists must be constructed next. Section IV gives the format. These lists must be assembled into the TSS communication region deck (CD600T1.001), TSS-TSSA, and specifically within blocks .TPRGD and .TPCOM. The size of the BSS area, following the program descriptor list, should be decremented one space each time a descriptor is added. This should be done to keep the overall size of TSSA constant. However, if the required number of entries exceeds the currently assigned value of .LNPD, the definition of .LNPD (maximum number of entries in the program descriptor list) must be modified. In this case, all TSS Executive modules referencing .LNPD must be reassembled.

An inspection of the listing indicates the required position of the program descriptor proper and the conventional placement of the command language/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  (in ASCII)
  y = command-language pointer
  z = number of command-language words
n=0 - EXEC primitive exists in list
≠0 - no EXEC primitive exists in list
     (for example, only CALLPs used)
```

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
```

Primitives are described in detail in Section IV.

Modifying the TSTART Module

The TSS deck CD600T1.015, TSS-TSSO, 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 Bit 32 = place on secondary program file #Q

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.

TEMPORARY SUBSYSTEM 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. Permission from the master user must be on record in TSS before a particular user can employ LODX.

LODX may also be used to load little-used subsystems not integrated into TSS.

The subsystem program is written and assembled for permanent placement. (The program descriptor need not be assembled; as it will not be referenced. Do not modify TSSO.) 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 decks as a GELOAD activity to GCOS with the following deck setup:

```
$  IDENT .....
$  USERID ID$PASSWORD
$  LOWLOAD
$  OPTION SAVE/prog-name,NOGO,NOSETU
     .
     .          other control cards
     .

     .
     .          subsystem decks
     .
$  EXECUTE
$  LIMITS .....
$  PRMFL  H*,WRITE,R,cat-name/filename
$  ENDJOB
```

Loading the Temporary Subsystem

After submitting the decks as a GELOAD activity, the user may have the subsystem loaded and checkout started from a terminal.

In response to SYSTEM?, specify LODX filedescr. For filedescr, give the file description specified on the $ PRMFL card.

LODX will then load the file and request a function response:

```
FUNCTION (RUN,PATCH,FILE,SAVE)
?
```

Responses:

RUN

Gives control to the program just loaded.

PATCH

Issues question marks for patch insertion, going into execution upon receipt of * or D. Upon null response (carriage return only) return is to the FUNCTION...? level.

FILE filedescr

This response accepts a file description, or will ask FILENAME? The specified file will be used as a patch source. The format of the file is exactly the same as a series of patches entered from the keyboard. If CARDIN or BASIC is used to create the file, the patch addresses will appear as pseudo line numbers.

A patch file created via the Text Editor may also contain D or * to indicate RUN. If an end of file is reached, LODX will return to the FUNCTION...? level to permit additional patches from the keyboard.

SAVE

This response saves the current file as it exists in core onto the permanent file whose file descriptor was used to load the file. In this way the user may alter his program by patches and save the altered program for later use. Upon completion of SAVE, LODX will return to the FUNCTION...? level.

Octal Patching the Temporary Subsystem

If PATCH is typed as the response to FUNCTION?, 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:

```
?address patch
?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 * or D after the question mark indicates that patching is completed.

The subsystem program is then executed.

SUBSYSTEM DEBUGGING FACILITY

During checkout, the Terminal Debug Subroutine (TDS) is provided for inclusion in a subsystem that is placed in the system. TDS is included on the System Subroutine Library. Its primary SYMDEF is TDS.

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, 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.

TDS 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 must be provided with an XED TDS, as in item 1.)

The subsystem to be checked out must contain a SYMREF to TDS. Then when the subsystem is submitted as a General Loader activity (to be loaded later by LODX), a binary object deck of the TDS subroutine must be included. *

The following program shows how to construct an H* file for use under LODX. It includes the use of TDS.

```
          1         8         16
10        $         IDENT
20        $         USERID
30        $         LOWLOAD
40        $         OPTION    SAVE/X,NOGO,NOSETU
50        $         ENTRY     ASCASC
60        $         GMAP      COMDK,NDECK
70        $         SELECT    DEESHIP/KTZAA
80        $         SELECT    LIBRARY/TDS
90        $         SELECT    K64BAIR/SCAF
100       $         EXECUTE
110       $         PRMFL     H*,R/W,R,DEESHIP/ASCASC
120       $         ENDJOB
```

Explanatory notes:

Line 50 - Specifies the SYMDEF entry point of the program to be placed on H*.

Line 70 - Directs GCOS to obtain from the user's file space a file called KTZAA, which may be a GMAP source file or a COMDK file.

Line 80 - Directs GCOS to obtain from the system's file space a file called TDS. (Object code)

Line 90 - Directs GCOS to obtain from another user's file space a file called SCAF. (Object code)

Line 100 - Directs the General Loader to load the three previous programs and, because of the NOGO option in line 40, not transfer control to the H* file produced by the loader.

Line 110 - Directs the General Loader to save the H* file on a random file called ASCASC in the user's file space.

It is assumed that the main program KTZAA includes SYMREFs to SCAF and TDS and that at least one XED TDS exists in the program.

TDS 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 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; 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 parameters. Using the Offset function, the user will have set the offset value.

The SA form specifies an absolute value for the address parameters.

The parameters are as follows:

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 parameters (Patch)
or
PA parameters (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 patch-location parameter.

The parameters are as follows:

aaaØbbb where aaa (1-6 octal digits) is the patch location. Field bbb is the octal patch 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. Tne patch field (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.

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

form:	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, allowing another request to be given.

Response: M parameters (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:	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, allowing 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 user must not attempt to insert a breakpoint at a location affected by a repeat-type instruction or containing one of the instructions listed below:

RPT	XEC
RPD	XED
RPL	MME
STC1	DIS
STC2	DRL

The permissible forms of the Breakpoint request are:

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

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

Response: Dxxxxx (Delete breakpoint)

This request indicates that the user wishes to delete a breakpoint established with the TDS Breakpoint option by replacing it with the original instruction. The effective address for the request is offset + xxxxx. If the effective address is offset + 0, xxxxx may be omitted.

When this request has been satisfied, the TDS subroutine responds with a question mark, allowing 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 R form of this request, the offset is added.

TDS Error Indications and Messages

1. ILLEGAL INPUT - RETYPE -- message appears 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 -- message appears in response to parameters not preceded by a function-type indicator.

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

4. NO ENTRY -- message appears when an attempt is made to delete a breakpoint at a location that does not contain one.

SUBSYSTEM DUMP FACILITY

Dump Procedure

If the user wishes his subsystem to be dumped when an unexpected fault occurs (or when, at his discretion, he calls for a dump via DRL ABORT) he does the following:

1. Creates a permanent linked file named ABRT of sufficient length to hold his entire subsystem.

2. Before calling the subsystem into execution, accesses the file named ABRT. This can be done with the ACCESS subsystem, GET command, etc.

The subsystem will now be dumped to this file when either a fault occurs which the subsystem does not handle or a DRL ABORT is executed by the subsystem. After this occurs, the user can inspect his dump with the subsystem called SABT (Scan Abort File), described below.

SABT (Scan Abort File) Subsystem

When a fault occurs in a subsystem which does not handle such faults, or a DRL ABORT is executed, and the user has a file named ABRT opened, the aborted subsystem is copied to the file. By means of the SABT subsystem, the user may scan the ABRT file by snapping portions of it at the terminal. (The ABRT file must have been accessed before calling SABT, or it must be a non-password, quick-access file named ABRT.)

SABT is called as a system selection:

```
SYSTEM? SABT
OFFSET?
```

The user may specify an offset to be added to all addresses requested. Designation of areas to be snapped may be given as in the following examples (all numbers are octal and will have offset; if any, automatically added to them).

	Meaning
?1235	snap 1 word at 1235
?172,14	snap 14 words starting at 1672
?2354-2367	snap from 2354 through 2367
?(carriage return)	done, return to subsystem-selection level.

Output is typed in the following form:

```
loc word1 word2 word3 word4
```

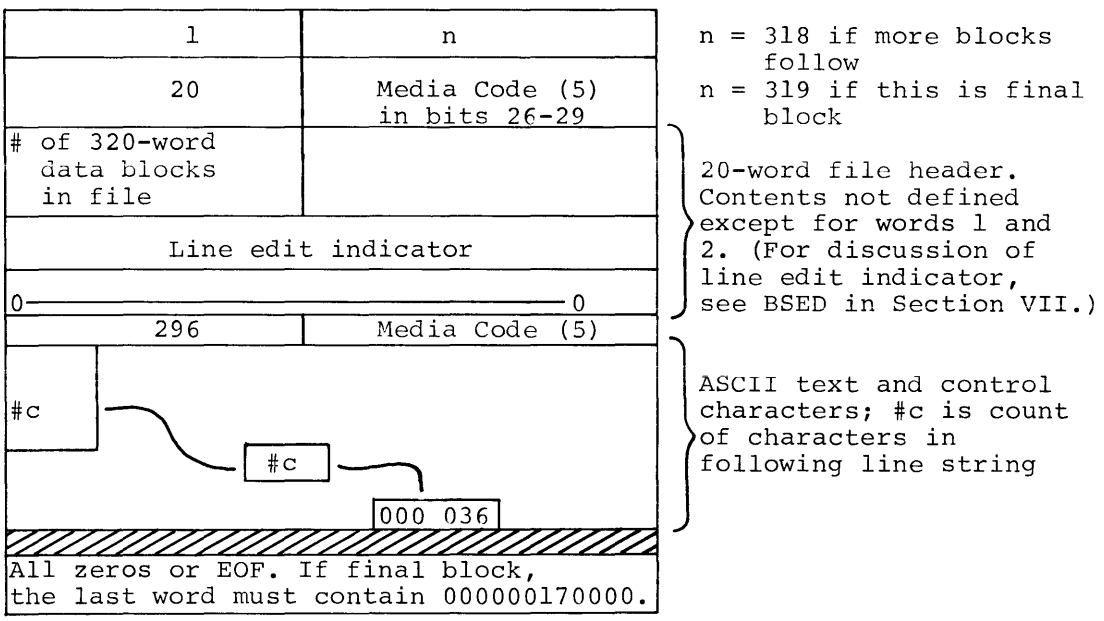
SECTION VI

FILE FORMATS

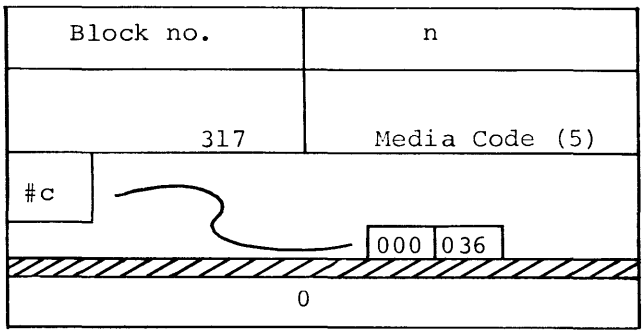
SOURCE (*SRC) FILE FORMAT

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 File and Record Control. The standard source file used by Honeywell-released subsystems (the current file) is named *SRC. Its format is as follows:

Initial 320-word block:



Second and succeeding 320-word blocks:



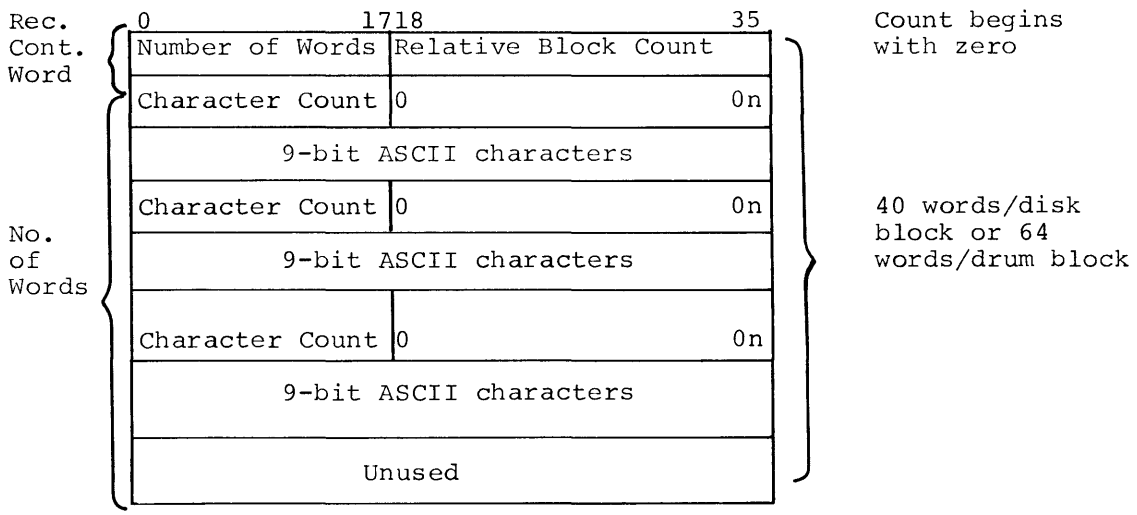
n = 318 if more blocks follow
n = 319 if this is final block

ASCII text and control characters;#c is count of characters in following line string

The ASCII text 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. A character count of zero followed by a character of 036 (octal) indicates that 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. The last word of the last block must also be 000000170000 (octal) to indicate EOF. A character string does not extend from one block to another.

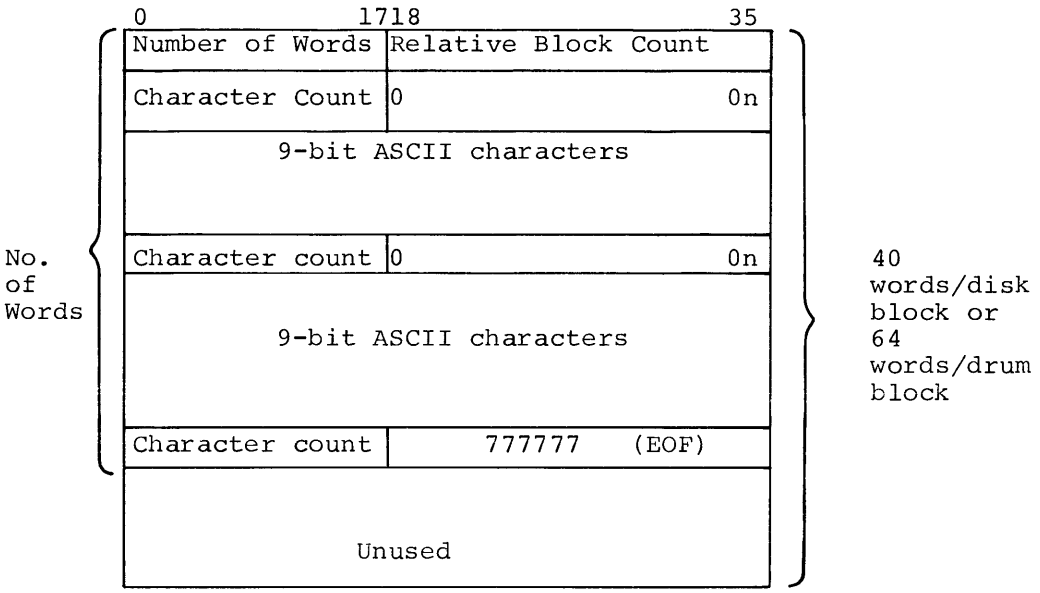
SY**FILE FORMAT

All nonempty records except the last:



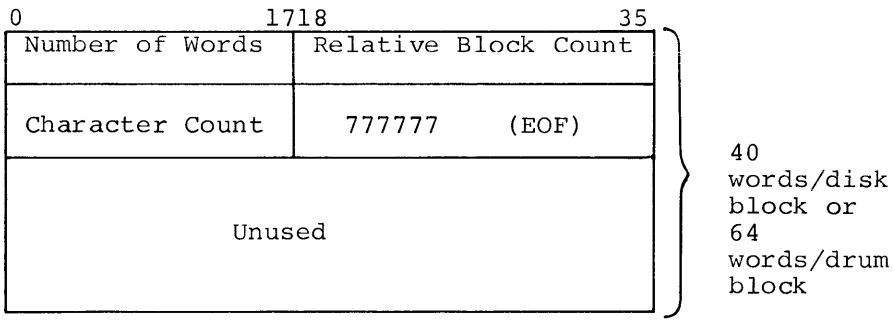
n (bit 35) = 1 if the string contains 80 characters with no carriage return; = 0 otherwise.

Final nonempty record:



n (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)



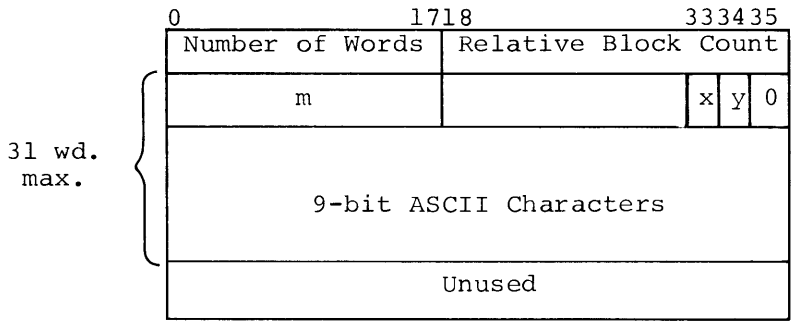
Note:

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

TAP* FILE FORMAT

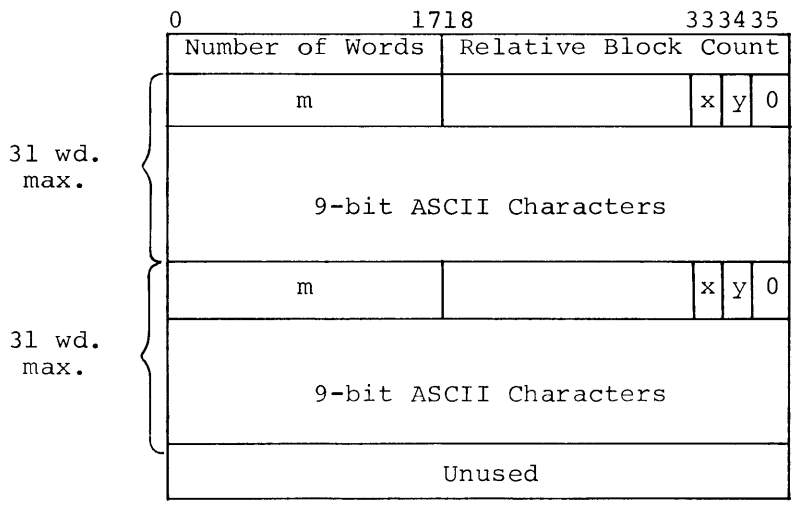
TAP* is the punched paper tape (PPT) collector file which contains the unedited PPT input. It is a random file, with a maximum of 2 links.

Format from Disk - 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

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

SECTION VII

HONEYWELL-SUPPLIED SUBSYSTEMS

A significant part of the time-sharing system is implemented in subsystem form, rather than as part of the Executive. For example, command-language processing is performed by subsystems, one subsystem for each command. These subsystems are, for the most part, available for use as part of a user-designed system within TSS. Functional descriptions of several subsystems are included in this section.

HONEYWELL SUBSYSTEM TYPES

There are three general types of Honeywell-supplied subsystems:

1. Working subsystems -- these are not directly callable by the terminal user, either by system-selection or by command, but are called by another subsystem as subroutines, to perform specific functions. An example is the BSED (Basic Edit) subsystem, which is called when editing of raw, line-numbered input, from file SY** to file *SRC, is required.

2. Command-processor subsystems -- these subsystems are called to process specific commands given by the terminal user in build mode. Certain of these also fall under type (3).

3. Direct mode subsystem -- these subsystems normally are called by the user at the subsystem-selection (SYSTEM?) level and communicate directly with him; ACCESS is an example. Certain of these also fall under type (2), since they may also be called at the command level -- for example, the SCAN subsystem. These subsystems are documented separately in other Series 6000 TSS manuals.

HONEYWELL SUBSYSTEM DESCRIPTIONS

The following definitions are pertinent to descriptions of certain commands and user responses that appear in the subsystem descriptions.

- filename -- a 1- to 8-character name of a permanent file

- filedescr -- this symbol can be replaced by one of the following:

  1. filename

  2. filename$password

  3. userid/catalog$password/.../
     catalog$password/filename$password

     If the file to be described emanates from the user's own master catalog, the USERID may be omitted and the file description begun with an initial slash:

     /catalog$password/.../filename$password

- permissions -- this may be one of the following:

READ (or R)	READ and WRITE
WRITE (or W)	READ and APPEND
APPEND (or A)	EXCLUSIVE
EXECUTE (or E)	

  The default interpretation for permissions is generally READ, WRITE (that is, for example, if READ permission only is desired, it must be specified).

- altname -- a 1- to 8-character alternate name.

  The altname parameter, enclosed in double quotes, is given when the file filename is referred to by an alternate name during the current session at the terminal.

- (i,j) -- i and j are line numbers within a specified file; thus, filename (i,j) specifies a file segment.

BSED (LINE EDITOR) SUBSYSTEM

Purpose:

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

Usage:

BSED is called as a result of commands issued while in the BASIC, FORTRAN, or CARDIN subsystems. When one of the commands--such as RUN, LIST, DONE, or SAVE--is entered 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 current file is assumed to be in proper order.

Word 2 of the header for each source-text file is used by BSED. (See Source (*SRC) File Format in Section VI.) This word is zero if the file is an old file not yet updated by the current BSED. Otherwise, the word contains one of the following:

- The number (binary) of the first line in the last block of the file.
- A negative number.

If word 2 contains a negative number, it indicates that the file has been altered and that it may contain empty or loose blocks or that the number of the first line of the last block may have changed. Thus, any subsystem that alters the contents of a source text file and creates empty or loose blocks should place a negative value in the second word of the header. This will indicate to BSED that such an alteration has occurred.

Error Comments:

<51> COLLECTOR FILE--I/O STATUS xx Unrecoverable read error occurred on SY**; xx is the hardware status returned.

<51> CURRENT FILE--I/O STATUS xx Unrecoverable read error occurred on *SRC; xx is hardware status returned.

<51< CURRENT FILE--I/O STATUS xx Unrecoverable write error occurred on *SRC; xx is hardware status returned.

<53> LINES IGNORED BY EDIT --- LINES... Indicated lines were not merged into current file because they lacked line numbers.

<54> SYSTEM MALFUNCTION--CURRENT FILE ERROR The *SRC file not in standard TSS source-file format.

BUILD SUBSYSTEM

Purpose:

Before entry to a subsystem not dependent on line numbers, reformats keyboard input on SY** or punched paper tape input on SYT* and writes this to *SRC, appending the input to any already-existing information.

Usage:

BUILD is called as a result of a first-column carriage-return command, a #TAP command, or an overflow (automatic-dump) of SY** or SYT*.

Error Comments:

<50> CURRENT FILE -- NO EOF
No logical EOF detected on read of *SRC.

<51> CURRENT FILE -- I/O STATUS xx
COLLECTOR FILE -- I/O STATUS xx
Unrecoverable read error occurred on *SRC or SY**, as specified; xx is the hardware status returned.

<51< CURRENT FILE -- I/O STATUS xx
Unrecoverable write error occurred on *SRC; xx is the hardware status returned.

<54> SYSTEM MALFUNCTION -- CURRENT FILE ERROR
*SRC not in standard TSS source-file format.

OLDN (OLD/NEW FILE REQUEST) SUBSYSTEM

Purpose:

The OLDN subsystem allows the user to specify the file he wants to work with as his current file.

Usage:

The OLDN subsystem is called whenever a user selects a subsystem, such as BASIC or EDITOR, that requires a current file and whenever a NEW, SAME, OLD, or LIB command is subsequently encountered.

The first response from the time-sharing system after such a subsystem is selected comes from the OLDN subsystem:

```
OLD OR NEW-
```

The legitimate responses to this demand are:

- NEW

  a. NEW

     The OLDN subsystem will see if a current file has been defined (opened) and if it has, it will initialize the file. If a current file has not yet been defined, it will be defined and initialized. the next response will be

     ```
     READY
     ```

  b. NEWP filedescr (permissions applicable)

     A permanent file filedescr is created and defined as the current file. It remains as the current file until the next NEW, NEWP, NEWP#, OLD, OLDP, OLDP#, or LIB command is given. The file is then deaccessed but remains in the user's catalog until released.

     If permissions are not specified, the file is created without permissions.

  c. NEWP# filedescr (permissions applicable)

     Execution is the same as for NEWP, except that only with the next NEWP, NEWP#, OLDP, or OLDP# is the file deaccessed. It remains as the current file as long as the subsequent commands are NEW, OLD, SAME, or LIB.

- SAME

  This response allows a user to keep his current file intact when changing from one system to another. The OLDN program will verify that the user already has a current file. The next response will be

  ```
  READY
  ```

- OLD

  a. OLD filedescr, permissions, "altname"

     (If filedescr is not given, the user will be asked OLD NAME.) OLDN will see if the named file has been previously accessed (opened), and if not, it will attempt to access the file. 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 current file and will copy the named OLD file to it. The next response from the time-sharing system will be

     READY

  b. OLD filedescr, permissions, "altname" (i,j)

     The specified file segment becomes the current file. The file must be a line-numbered file.

  c. OLD filedescr, permissions (i,j)1;...;
     filedescr, permissions (i,j)n

     The files and/or file-segments specified by filedescr (1 through n) are adjoined in the order listed and become the current file. Adjoining of BASIC files should be done with caution (line numbers are also statement numbers).

     If the list is too long for one line, the subsystem will request more input if a comma is the last nonblank character before the carriage return. The altname parameter may be used as required in this form of the response also.

  d. OLD f(i,j)1:...:f(i,j)n (permissions and altname applicable)

     The n files or file segments are merged together by line numbers, and become the current file, where f is a filedescr (colon-separated). If duplicately numbered statements appear in two or more files, each such statement will appear in the order specified by the file list. The asterisk designating the contents of the current file (or segment thereof) may appear as a filedescr anywhere in the file list.

e. OLDP filedescr (permissions applicable)

Existing permanent file filedescr becomes the current file. It remains as the current file until the next OLD, OLDP, OLDP#, NEW, NEWP, NEWP#, or LIB command is given. The file is then deaccessed but remains in the user's catalog until released.

If permissions are not specified, all permissions are automatically requested for the file.

f. OLDP# filedescr

Execution is the same as for OLDP, except that only with the next OLDP, OLDP#, NEWP, or NEWP# command is the file deaccessed. It remains as the current file as long as the subsequent commands are NEW, OLD, SAME or LIB.

- LIB filedescr

  The file in the system library specified by filedescr becomes the current file.

When NEW is subsequently given, OLDN will reinitialize the current file and the next response will be:

READY

Other responses from the OLDN subsystem 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.

- PASSWORD - filename - ?

  Followed by a mask over which the password is to be typed. The user has failed to supply a required password.

APPENDIX A.

SYSTEM MACROS

The time-sharing system modules and subsystems require many definitions of the Communication Region, UST contents, DRLs, etc. These have been formulated into a set of macros. They are loaded by

```
LODM .G3TSn
```

where n depends upon the catalog name currently assigned.

The macros and their use are as follows:

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

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

.SSDRL — Normal subsystems 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.).

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	∅	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	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	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	TLD
037	US	077	?	137	_	177	DEL

DEFINITIONS

Communications Control

ACK	Acknowledgment
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 Block
ETX	End of Text
NAK	Negative Acknowledgment
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	^
DEL	Delete (Rubout)
EXP	!
GRA	`
GTN	>
LBK	[
LBR	{
LTN	<
NULL	Null
RBK	]
RBR	}
RSL	\
SI	Shift In
SO	Shift Out
SP	Space
TLD	~
VTL	Vertical Line

APPENDIX C.

Series 600/6000 Standard Character Set

Character Set				
HONEYWELL	CARD	OCTAL	ASCII	
			0 - -	1 - -
0	0	-00	NULL	@
1	1	-01	SOH	A
2	2	-02	STX	B
3	3	-03	ETX	C
4	4	-04	EOT	D
5	5	-05	ENQ	E
6	6	-06	ACK	F
7	7	-07	BELL	G
8	8	-10	BSP	H
9	9	-11	HT	I
[	2-8	-12	LF	J
#	3-8	-13	VT	K
@	4-8	-14	FFD	L
:	5-8	-15	CR	M
>	6-8	-16	SO	N
?	7-8	-17	SI	O
♭	(blank)	-20	DLE	P
A	12-1	-21	DC1	Q
B	12-2	-22	DC2	R
C	12-3	-23	DC3	S
D	12-4	-24	DC4	T
E	12-5	-25	NAK	U
F	12-6	-26	SYN	V
G	12-7	-27	ETB	W
H	12-8	-30	CAN	X
I	12-9	-31	EM	Y
&	12	-32	SUB	Z
.	12-3-8	-33	ESC	[
]	12-4-8	-34	FS	\
(	12-5-8	-35	GS	]
<	12-6-8	-36	RS	^
\	12-7-8	-37	US	_
↑	11-0	-40	♭	`
J	11-1	-41	!	a
K	11-2	-42	"	b
L	11-3	-43	#	c
M	11-4	-44	$	d
N	11-5	-45	%	e
O	11-6	-46	&	f
P	11-7	-47	'	g
Q	11-8	-50	(	h
R	11-9	-51	)	i
-	11	-52	*	j
$	11-3-8	-53	+	k
*	11-4-8	-54	,	l
)	11-5-8	-55	-	m
;	11-6-8	-56	.	n
'	11-7-8	-57	/	o
+	12-0	-60	0	p
/	0-1	-61	1	q
S	0-2	-62	2	r
T	0-3	-63	3	s
U	0-4	-64	4	t
V	0-5	-65	5	u
W	0-6	-66	6	v
X	0-7	-67	7	w
Y	0-8	-70	8	x
Z	0-9	-71	9	y
←	0-2-8	-72	:	z
,	0-3-8	-73	;	{
%	0-4-8	-74	<	|
=	0-5-8	-75	=	}
"	0-6-8	-76	>	~
!	0-7-8	-77	?	DEL

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form*

TITLE:	SERIES 600/6000 GCOS TIME-SHARING SYSTEM PROGRAMMERS' REFERENCE MANUAL	ORDER No.:	BR39, REV. 1
		DATED:	NOVEMBER 1971

ERRORS IN PUBLICATION:

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

(Please Print)

FROM: NAME _____ DATE: _____

COMPANY _____

TITLE _____

_____

_____

ALONG LINE

*Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here. ☐

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS,
MASS. 02181

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

POSTAGE WILL BE PAID BY:

HONEYWELL INFORMATION SYSTEMS
60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATTN: PUBLICATIONS, MS 050

Honeywell

The Other Computer Company:

Honeywell

HONEYWELL INFORMATION SYSTEMS

3880
1.5372
Printed in U.S.A.

In the U.S.A.: 200 Smith Street, MS 061, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario

BR39, Rev. 1