UEP OPERATING SYSTEM

TABLE OF CONTENTS

TABLE OF CONTENTS

HEP OPERATING SYSTEM

TABLE OF CONTENTS

TABLE OF CONTENTS

HEP OPERATING SYSTEM

TABLE OF CONTENTS

TABLE OF CONTENTS

HEP OPERATING SYSTEM

TABLE OF CONTENTS

HEP OPERATING SYSTEM

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

HEP OPERATING SYSTEM

TABLE OF CONTENTS

HEP OPERATING SYSTEM

1. EXECUTIVE

The HEP Executive resides on a Digital Equipment Corporation PDP-11 computer. It provides operating system services associated with physical I/O, job preparation, maintenance and debugging. In general, any operating system function whose execution time is not critical is provided by the Executive.

Executive services are provided by Executive tasks, which are described in more detail later in this section. These tasks are coordinated by a mini-operating system in the PDP-11. This mini-OS, called the ROOT, manages memory for the Executive tasks, protects them from each other, and provides certain services to the Executive tasks.

1.1 Root

The Root is the only Executive module written directly in assembly language. The Root is loaded into low PDP-11 memory by the boot-strap process. Other Executive tasks exist as independent disk files, and are loaded by the Root as part of the system initialization. Thus, Executive tasks are separately compiled, and may be changed without rebuilding the entire operating system. Initialization of the disk for system boot is handled by a specialized task - the Disk Builder (DB) and is discussed later under that heading.

1.1.1 Basic Services

Once all Executive tasks are loaded, the Root provides basic operating system services to the tasks. These services are described on the next page.

1.1.1.1. Task Management

Each Executive task is allocated a certain amount of memory for execution. The address space of each task is allocated by the Root as follows:

0-8K	Code-Read Access
8-16K	Code-Read Access
16-24K	Code-Read Access
24-32K	Code-Read Access
32-40K	Data-Read/Write Access
40-48K	Message-Read/Write Access (only 128 bytes used)
48-56K	Message-Read/Write Access (only 128 bytes used)
56-64K	I/O Page-Read/Write Access

The physical memory associated with this 64K byte address space is fixed by the Root at IPL time, and remains allocated forever. Task context switching is handled by the Root, and involves the manipulation of the PDP-11 memory management registers to protect and isolate tasks from each other.

Executive tasks are dispatched strictly in priority order. The priorities are determined by the order of tasks at Disk Build time. Root task management routines always dispatch the highest priority ready task. While there is considerable latitude in the dispatching order of tasks, inappropriate dispatching priorities can result in system failure.

1.1.1.2 Message Routing

Terminal and inter-task communications are handled by messages passed from task to task by the Root. In order to avoid copying of message text, memory mapping registers 5(40 - 48K) and 6(48K - 56K) are used. A task wishing to send a message executes Trap 0, and upon return, the Root has set up mapping register 5 to point

to an available message buffer and general register 0 to address the buffer. The message buffer is a 128 byte area with the following format:

Byte 0	Link
Byte 2	Source Destination
Byte 4	Priority
Byte 6	Length
Byte 8	Type
10-127	Data

The message is transmitted by placing word 0 of the message in general register 0 and issuing Trap 1. The Root locates the destination task using the DEST field of the message.

Tasks 0 - 31 are dummy terminal tasks. Transmitting to these tasks causes the message to enter the terminal service routines. These are described in the next section. Tasks 32 and greater and are actual tasks. Task numbers for these tasks are determined by Disk Build. All real tasks have an input queue into which all messages sent to them are placed. Messages in the queue are maintained in priority order using the priority field of the message buffer. When a task wishes to process a message, it issues Trap 2. If a message is waiting, mapping register 6 and general register 0 are set up to point to the message and the task continues. If no message is waiting, the task enters Message Wait state and the Root dispatches the next ready task. After a task processes a message, it places the link field of the message in general register 0 and executes Trap 4. This releases the message and places it in a list of available message buffers maintained by the Root.

All message buffers are in the first 56K of real memory. The link field of a message is the actual memory address of the message and is used by the Root to manipulate it. No error checking is performed by the Root in message handling. If Executive task violates the message protocol, a complete system crash will eventually result. Most message handling is performed by Executive task runtime library routines, but some tasks manipulate messages directly. This is acceptable, but requires extreme care.

1.1.1.3 Terminal Service

ASCII terminals connected to the Executive computer are handled by terminal service routines in the Root. Input to these terminals is assembled into messages and sent to appropriate tasks. Input messages are type 1 (Command). Output messages from Executive tasks which address tasks less than 32 are routed to the terminal service routines. Output messages should be type 3 (Display Text). The association between task numbers and terminals is compiled into the Root. By convention, task 0 is the console terminal.

To a task, terminals look just like any other task, however, terminal service routines handle messages differently than regular tasks. When a message is input to a terminal, the destination of that message is taken as the first task which sent a message to that terminal. Thus tasks must output at at least one message to a terminal before expecting input. An exception to this is the console terminal. On the console terminal, input may be preceded by a two character task ID and one or two colons. Every task has a task ID assigned by the Disk Builder and console messages are routed using the task ID. If the task ID is followed by a single colon, only the current input is routed to the specified task. If two colons are used, all subsequent input with no specified task is sent to the named task, until a new task ID and two colons is entered.

Since a task can generate lines of output much faster than a terminal can print them, the terminal output routines maintain a count of messages queued for printing by each task. When a task has more than two messages waiting for printing, it is placed in Output Wait state. This is required to prevent a task doing multi-line output from consuming all the message buffers in the system. As messages are printed, the sending tasks are re-activated to generate further output.

Certain terminal interfaces (DZ-11) are capable of programmed baud rate selection. This is controlled by the use of Control-S and Control-Q characters in output messages. If a task sends Control-S to a terminal, the next character is used to set the baud rate, and the terminal is placed in single-character input mode. Each character typed is sent directly to the controlling

task as a separate message. Normal mode is entered when a Control-Q character is sent to the terminal.

The terminal service input routines interpret several characters for control functions. These are:

Control-H	- Delete Last Character Typed
Tab	- Insert Spaces to Next Multiple of 8 Columns
Backslash	- Delete Last Line Typed
Control-S	- Suspend Output
Control-Q	- Resume Output
Carriage Return	- Terminate Input Message

All other control characters are ignored.

1.1.1.4 I/O Interrupt Services

Non-terminal I/O is performed directly by each Executive task. When device latency is short, tasks normally 'busy wait' on I/O completion. Where latency is long, the Root provides I/O interrupt support to enable a task to relinquish the processor until I/O is complete. A task waits for I/O by placing the CSR address of the device being used in general register 0 and executing Trap 6. This causes the task to enter I/O Wait state until I/O is complete.

The Root supports a specific set of I/O devices for interrupt. These are:

Disk (CSR 176700)
Tape (CSR 172522)
Line Printer (CSR 177514)

Other devices must be used without interrupts, except as described in Section 1.1.3 - Switch Interface.

The interrupt service mechanism in the Root records the occurence of interrupts on the supported devices, even if no task is in I/O wait for them. Thus a task may start I/O on a device, enabling its

interrupt, and subsequently issue Trap 6. If the device has already interrupted, the task will immediately continue, otherwise it will wait. There is no timing requirement on the task's issuance of Trap 6 in order to detect the I/O complete.

1.1.1.5 Miscellaneous Services

The Root supports several other Trap codes and features associated with inter-task communication. These are:

Trap 3 - Send and get buffer,
(Trap 2 followed by Trap 0).

Trap 5 - Free a buffer and wait for next buffer,
(Trap 4 followed by Trap 2).

Trap 7 - Test for input buffer waiting.
General register 0 set to 0 if no buffer, set to non-zero if buffer waiting.

Trap 8 - Wait for next tick of the 60 cycle clock (0-16 ms).

Trap 9 - Get task number of task whose two character ID is in general register 0. Task ID is returned in the low byte of register 0. The high byte contains the task issuing Trap 9.

In order to allow tasks to coordinate, if the high byte of the message type of a message is non-zero, the sending task is placed in Reply Wait status, instead of continuing execution. In order to resume the task, the receiving task must load register 0 with the link field of the received message and issue Trap 10. This will make the original sender ready. The two tasks may pass information back and forth through the message. Finally, one of the tasks must release the message using Trap 4. If a task issing Trap 10 wishes to continue, it must clear the high byte of the message before issuing Trap 10, otherwise it will itself enter Reply Wait state.

1.1.2 Executive Debugger

The Executive Debugger is a standard Executive task written in PASCAL. It differs from other tasks only in that it is linked as part of the Root, rather than being loaded from disk like normal tasks. The task ID for the Executive Debugger is 'XD'.

The Executive Debugger may be used to examine and modify the memory of other tasks and may set breakpoints in other tasks. It is also used by the Root to print error messages caused by Executive task malfunctions. The Executive Debugger is accessed from the operator's console using a standardized command sequence. Command line syntax is as follows:

{TYPE ID} {RANGE} . {TASK ID} <=VALUE>

If the "=VALUE" suffix is omitted, the specified item is printed, otherwise it is set to the entered value.

Type ID is a single character describing the item to be examined or modified. Valid types are shown below.

TYPE	RANGE	MEANING
Blank or omitted	0-177776	Computer Memory Address
A	0-6	Memory Address Mapping Registers
B	0-7	Breakpoint Locations
C	-	Count of Waiting Output Terminal Messages
F	-	Flag Showing Dispatching State
P	-	Examine only, Continues From Breakpoint
Q	-	Head of Input Message Queue
R	0-7	General Purpose Registers
S	-	Processor Status Word

Range is entered as a single octal integer or pair of integers separated by a comma. For modify operations only the first range value is used; for examine operations, all locations between the two values are displayed.

Task ID selects the task whose data is referred to. Task ID may be omitted when examining computer memory, and absolute locations are when referred to. Only locations in the first 40K of real memory may be accessed this way, and only for examine.

The Executive Debugger is also used to print messages generated by error traps from other Executive tasks. These messages include the trap type, task ID, PC and status word of the trapping task. Trap types are:

RS - Privileged Instruction (usually HALT)
IL - Illegal Instruction or Nonexistent Memory
OD - Odd Address
MM - Memory Management Violation
BP - Breakpoint
EM - Emulator Trap (not used by this system)
FP - Floating Point
IO - IOT Trap (not used by this system)
PF - Page Fault - Memory Management Error

1.1.3 Switch Interface

A major communications path between the HEP and Executive tasks is the switch interface. This interface appears to the HEP as a set of 16 memory locations, of which three are presently used. A HEP memory access is broken into two parts - a request and a response. The switch interface generates a Root interrupt when a request is received, but does not generate a response. Responses are generated under software control of the responsible Executive task.

In order to facilitate use of the switch interface, three small Executive tasks are incorporated into the Root. These tasks are activated by Root interrupt code when a switch request is received. They read the contents of the switch request and send it to the appropriate Executive task. These Root tasks are described on the next page.

1.1.3.1. Kernel Inbound Task (KI)

The Kernel Inbound Task is used by all Executive tasks wishing to send messages to the HEP Kernel. During HEP IPL, each PEM sends to KI the address of its communications area. This information is saved by KI. After saving its address, the PEM then attempts to read a word from the switch interface. KI holds this request and issues no response. After receiving the read request, KI enters Message Wait state via Trap 2. When an Executive Task wishes to send a message to a HEP processor, it begins by sending a Seize with Reply message (Type 13) to KI. KI places the communications area address for that processor in the message and issues Trap 10 (Reply). This places KI in Reply Wait and activates the original sender. The sender writes data to the communications area and sends an Activate-With-Reply (message type 14) to KI via Trap 10. This causes KI to respond to the outstanding read from the PEM, using the contents of the message as the response data. The PEM process receiving the data uses it to control message processing. After processing, the PEM issues another read request. This causes a Root interrupt which activates KI. KI generates another Reply message to the original sender and enters Reply Wait. This process continues until the transaction is completed. At this point, the sender generates a Release (message type 15) with no reply and sends it to KI via Trap 10. KI frees the message with Trap 4 and issues Trap 2 to get its next input message. The reply mechanism causes KI and a HEP Executive task to run as co-routines during HEP message transmission, and provides an interlock allowing sharing of the switch interface without conflict between multiple senders.

1.1.3.2 Kernel Outbound Task (KO)

The Kernel Outbound Task handles unsolicited messages from the Kernel to the Batch Monitor Executive Task. During initialization, it enables interrupts on the switch location used for this purpose. When an interrupt is received, it assembles the switch data into a message and forwards it as a Switch Message with Reply (message type 12) to the Batch Monitor. When the Batch Monitor completes message processing, it replies to KO, and KO generates a switch response, frees the buffer and reenables interrupts for the next unsolicited message.

1.1.3.3 Request File Task (RF)

The Request File Task is similar to the KO task (in fact, most of the code is common) except that a different switch location is used and messages are sent to the File Manager rather than the Batch Monitor. The RF task is used for communications between HEP supervisor tasks and the file system.

1.2 File Manager

All disk I/O in the HEP System (except during IPL) is performed by the File Manager. Operations supported by the File Manager are:

Logon - Validate User ID

File Open - Locate an Old File or Create a New One

Read a Physical Record

Write a Physical Record

Obtain the Address of an Unused Physical Record

File Close - Close, Delete or Rename a File

In addition, operator commands exist to:

Enter Debug Mode

Leave Debug Mode

Add a User ID

Shut Down the File Manager

For read/write operations, the File Manager merely performs disk control functions and data transfer on behalf of requesting Executive Tasks or HEP supervisor processes. For other operations, the File Manager performs directory search/update functions and search/update of the disk free section tables.

1.2.1 Disk Format

The system disk is a fixed sectored 300Mb moving head disk with a 1.2 Mbyte/second transfer rate. Sectors are 512 bytes (64 HEP words) long.

1.2.1.1 File Format

Files in the HEP system are a doubly-linked list of physical records. Each record contains two HEP words of linkage information, followed by 62 words of data. The linkage information is part of the record and is

made available to and supplied by all software interfacing to the File Manager. The format of a physical record is shown in Figure A.

WORD 0	THIS CYLINDER	THIS TRACK	THIS SECTOR	NEXT CYLINDER		NEXT TRACK	NEXT SECTOR
WORD 1	LAST CYLINDER	LAST TRACK	LAST SECTOR	USER NO.	FILE NO. WITHIN USER	RELATIVE RECORD NO. IN FILE	
WORD 2-63	DATA						

Figure A - DISK RECORD FORMAT

The cylinder, track and sector information is used to chain records together. Maintenance of this information is the responsibility of Executive and supervisor tasks calling the File Manager - it is not checked or modified by the File Manager except during file accesses for internal File Manager purposes.

The "next" fields of the last record in a file contain all zeroes; similarly, the "last" fields of the first record of a file contain all zeroes.

The user number, file number and record number fields are used for file consistency checking and system debug. They should be maintained by all Executive tasks and HEP supervisors.

The format of the "this", "next" and "last" fields is referred to as a "diskaddress" and is the standard format for representing disk locations. Each diskaddress occupies 32 bits (1/2 HEP word).

1.2.1.2 Directory Format

In the HEP system, files are accessed via a tree-structured directory system. At the leaves of the tree are disk file headers. Each file header occupies one record, and contains complete information about a single file. The format of a file header is shown in Figure B.

WORD				
0	THIS ADR.		PREV. ADR.	
1	NEXT ADR.		USER NO.	FILE NO. / RECORD NO.
2	LEN	CKSUM	OCOUNT	MOD.-
3	DATE		DATE	
4		AC DATE		
5	FREC		LREC	
6	UFD		FHP	
7	RECSIZE	ACPRIV	EOFW	FLEN
	FILENAME			

Figure B - FILE HEADER FORMAT

The format of word 0 and word 1 of a file header is standard. These words are used to link all file headers for a particular user into a file named 'HEADER'. This file is automatically maintained by the File Manager. By reading this file, a user program may obtain the name and all pertinent characteristics of all of its files. The remaining fields in the file header pertain to the specific file and are discussed below.

LEN - The length of the file name in bytes.

CKSUM - The exclusive OR of all the character pairs in the file name.

OCOUNT - The number of users who have this file open. If negative, one user has the file open, and additional opens are not allowed.

MODDATE - A 48 bit field containing the date and time this file was last closed by a user with write access. The date is in standard system date format, described in Section 1.11 - Disk Builder.

CRDATE - Date this file was created, in standard format.

ACDATE - Date this file was last accessed.

FREC - The diskaddress of the first record in the file. All files have at least one record, which may contain no data.

LREC - The diskaddress of the last record of the file.

UFD - The diskaddress of the UFD record pointing to this file. UFD records are discussed on the next page. This pointer is used during file delete/rename operations.

FNP - The diskaddress of the file 'HEADER' for this user. Used for file delete/rename operations.

RECSIZE - The record size in HEP words of the records in this file. This informaton is not used by the File Manager, who deals in physical records only.

ACPRIV - Access privileges for this file. The high byte of the field controls public access privileges, while the low byte controls the users own access privileges. Bits in each byte are defined as follows:

.......1	Read Access
......1.	Write Access
.....1..	Extend Access
....1...	Exclusive Access
...1....	Semaphored Access (not used)
..1.....	Delete/Rename Access
.1......	Execute Access (not used)
1.......	Access Change Access

EOFW - End of file word. The word number of the first free word in the last record of the file. All files must be an integral number of words long. All files must contain at least one word; for an empty file, FREC = LREC and EOFW = 0. If a file is an integral number of physical records long, an extra record is present at the end of the file, and EOFW = 0.

FLEN - The record number of the last record in the file (zero relative).

FILENAME - The 1 to 448 character name of the file. The filename is stored in a byte-swapped format within each word. Characters are in the order shown below:

1	0	3	2	5	4	7	6

This is a consequence of the way the Executive computer (a PDP-11) addresses bytes.

In order to speed up searching the file directories, an indexing file, called the User File Directory, or UFD, is maintained for each user. This file resides in the directory of the distinguished user whose ID is '000000000000'. The name of this file is UFD. XXXXXX, where XXXXXX is the ID of the user in question. The format of the records in the UFD file is shown in Figure C(a) and Figure C(b).

WORD 0	THIS		NEXT
WORD 1	LAST		COUNTS
WORD 2	LEN	CKSUM	FHA
WORD 3	LEN	CKSUM	FHA
	. . .		
WORD 63	LEN	CKSUM	FHA

Figure C(a) - UFD ENTRY FORMAT

LEN	CKSUM	CYLINDER	TRACK FHA	SECTOR

Figure C(b) - UFD ENTRY FORMAT

The LEN and CKSUM fields in a UFD entry are duplicates of the corresponding fields in the file header to which it refers. The FHA field is the diskaddress of the fileheader for the file. When searching for a file, the File Manager need only read the file headers of files with corresponding length and checksum fields. Since 62 files may be referred to per UFD record, a considerable saving in open time results. The UFD is automatically maintained by the File Manager, and is not visible or accessible to the user.

In order to permit access to files from multiple users, the User File Directories are pointed to by a higher level directory called the Master File Directory or MFD. This file is also held under the distinguished ID '000000000000'. The format of an MFD record and MFD entry is shown in Figure D(a) and Figure D(b).

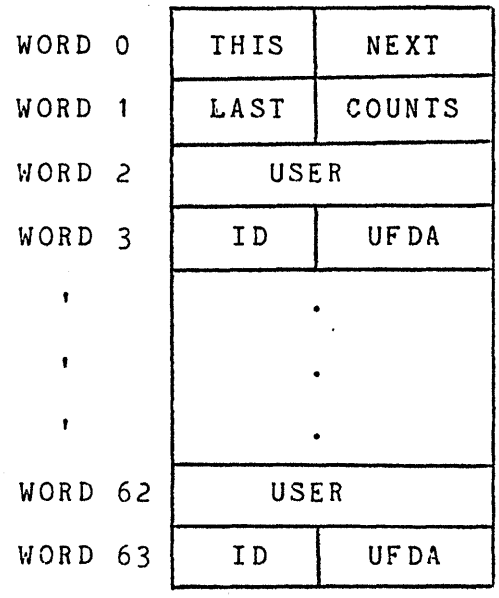


Figure D(a) - MFD RECORD FORMAT

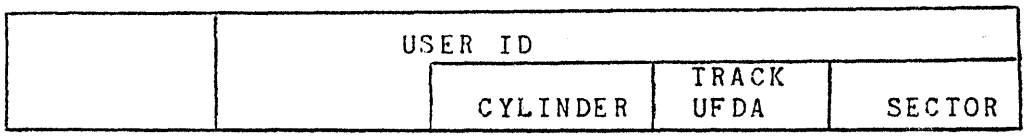


Figure D(b) - MFD ENTRY FORMAT

The user ID is a 12 character (padded with blanks) character string in byte-swapped format as described for file names. The diskaddress points to the first data record of the corresponding UFD. Each user in the system has a single MFD entry.

The UFD's and MFD are maintained as files by the File Manager. Access to their data is not made by normal file access mechanisms. The File Manager searches and updates these files using internal routines not available to other tasks. The MFD, the UFD for the distinguished user, and other files are built by Disk Build during disk initialization.

1.2.1.3 Bitmap and Reserved Sector Format

When additional sectors are required for a file on the disk, an unused sector is allocated using the disk bitmap. The bitmap is a file consisting of one record on each disk cylinder. Bits in the data portion of the record correspond to sectors on the cylinder. Since there are 32 sectors on a track, and 19 tracks per cylinder, 19 two-word pairs are used to represent the cylinder. Bits corresponding to allocated sectors are zero, while unallocated sectors have 1's in their bit position. The bitmap record is on a fixed track and sector on all cylinders. Its location is determined by Disk Build. For convenience, a standard file header is built for the bitmap, under the distinguished user ID '000000000000'. The name of the file is BITMAP.

The File Manager maintains the bitmap record for one cylinder in core at all times. All requests for records are allocated from this cylinder until it is full. At this point, the File Manager moves to the next highest cylinder (modulo the maximum valid cylinder) until available sectors are found. Thus bitmap I/O is minimized, and all files being extended at the same time will go on the same cylinder if possible. This reduces disk latency and improves performance.

Cylinder 0, track 0, sectors 0 and 1 are unique in that they are marked allocated in the bitmap, but are not part of any file. Sector 0 is the hardware bootstrap, and is described in conjunction with the Disk Builder. Sector 1 is the File Manager and IPL pointer sector.

The THIS field of sector 1 points to the first data record of the MFD. The LAST field of sector 1 points to the bitmap. The data portion of sector 1 contains pointers to IPL files and is described with Disk Build.

1.2.2 Basic File Management Routines

1.2.2.1 OBTAIN

OBTAIN is used to get an unallocated sector in which to write data. The sector is marked allocated by OBTAIN.

1.2.2.2 LOOKUP

LOOKUP is used to search a user file directory for a specified file. If the lookup is successful, the file header of the file is made available to the caller.

1.2.2.3 LOGON

LOGON is used to locate a specific user file directory. If the logon is successful, the diskaddress of the UFD is made available to the caller.

1.2.2.4 ENTER

ENTER is used to add a file header to a specified user file directory. An initial data record is allocated and initial values in the file header are supplied. No duplicate file checking is performed.

1.2.2.5 ADDUFD

ADDUFD is used to create a UFD and enter it into the MFD. It is only activated under operator command. No duplicate UFD checking is performed.

1.2.2.6 RELEASE

RELEASE is the opposite of OBTAIN, and is used to free sectors in the bitmap when files are deleted.

1.2.2.7 DELETEFILE

DELETEFILE is used to remove a file header from a UFD, delete the file header record, and queue the file data records for deletion. Since this process may be lengthy, it is handled as a 'demon' during otherwise idle File Manager time.

1.2.2.8 RENAMEFILE

RENAMEFILE updates a UFD and file header to contain a new name. Note that only the name of a file can be changed, not its owning UFD.

1.2.3 Executive Interface

Executive tasks communicate with the File Manager using the standard system message mechanism. Several message types are processed:

TYPE	MEANING
6	File Open
7	File Close
8	Record Read
9	Record Write
10	Obtain a Sector
11	Logon

A common message format is used for open, close, read and write. This format is shown in Figure E.

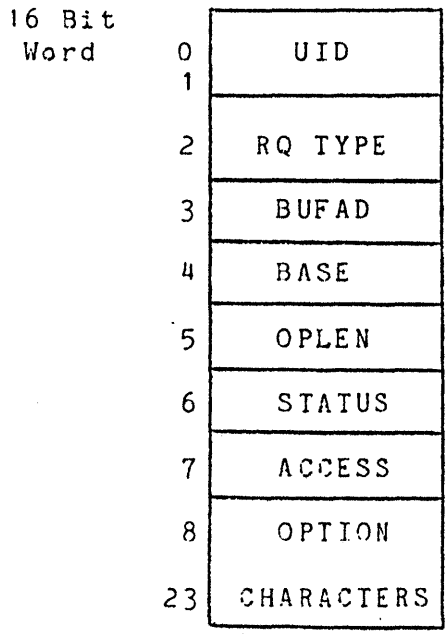


Figure E - OPEN/CLOSE MESSAGE FORMAT

These fields are used as follows:

UID - Diskaddress of user's UFD (open, close).
Supplied by caller.

RQ TYPE - If 0, open for output.
If nonzero, open for input (open, close).
Supplied by caller.

BUFAD - Address in caller's space of disk record (open, close, read, write).
Supplied by caller.

BASE - Base of user stack div 64, (open, close, read, write).
Supplied by caller.

OPTLEN - Length of option characters (open).
Supplied by caller.

STATUS - Result of operation (open, close, read, write).
Set by File Manager.

ACCESS - Requested access codes (open, close)
for this open - set by File Manager. Based on option string on open.
Supplied by caller on close.

OPTION CHARACTERS - ASCII characters specifying (open, close).
Open or close options
Valid options are:

/W = Write Access
/R = Read Access
/A = Append Access (extend plus position to end of file)
/T = Temporary File
/N = New File (delete old if present)

Option processing is provided as a service to Executive Tasks. Not all option bits are used by the File Manager. Positioning to end of file (/A) and file deletion on close (/T) are the responsibility of the caller. Bits are set in ACCESS to indicate these options were specified, but action in these options must be taken by the caller. The low byte of ACCESS has the format described for ACPRIV in the file header. The high byte is as follows:

```
.......1    Temporary (/T)

......1.    Append (/A)

.....1..    New (/N)
```

All message communication with the File Manager uses the Root reply mechanism, and the File Manager responds to Executive requests via Trap 10 (Reply).

1.2.3.1 Executive Open

An Executive task opens a file by issuing message 6. The BUFAD field of the message points to a disk record. In this record, the FILENAME, LEN, and RECSIZE fields are supplied by the user (RECSIZE is only used if the file is to be created). The FILENAME may contain a user ID in square brackets at the start, and may contain access options in parenthesis at the end. Access options are only used if the file is to be created. The open routine USEROPEN strips the user ID and options. If the user ID was present, the File Manager uses LOGON to locate the UFD, otherwise the UFD diskaddress in the open message is used. The access options are processed into the ACPRIV field of the user is file header, and LOOKUP and ENTER are used to locate and/or create the file. Access privileges resulting from the processing of the message option string are stored in the message ACCESS field and checked against ACPRIV in the file header. If valid privileges are requested the actual disk file header is written to the callers disk record, and the message status is set to 0. If not, the message status is non-zero and the file header is not supplied.

The format of the access code string following the file name is; (PRWEXD, URWEXD) where the string beginning with 'P' denotes public access privileges and the string beginning with 'U' denotes user access

privileges. Any or all of the access characters may be supplied:

R - Read Access

W - Write Access

E - Extend

X - Exclusive

D - Delete/Rename

These characters determine the permanent access attributes of the file if it is created by open.

Error returns from open are given in Table F.

STATUS	ERROR
-64	Nonexistent User ID
-65	Bad Message Options
-66	File Existence Conflict (duplicate file or nonexistent file)
-69	Requested Access Denied
-70	Exclusive Access File Already in Use
-71	Disk I/O Error
-68	Attempt to Create File in Another UFD or other Enter Failure.

Table F - OPEN ERROR CODES

1.2.3.2 Executive Close

An Executive task closes a file by issuing message 7. The format of the message is as indicated previously. The file header pointed to by BUFAD must contain the diskaddress of the file header to be closed in the 'THIS' field. If the first character of the option field in the message is 'D' the file will be deleted. If the first character is 'R', the file will

be renamed and the LEN and FILENAME portions of the file header must contain the new name and its length.

Error codes from CLOSE are given in Table G.

STATUS	ERROR
-18	Delete or Rename Not Done - Access Violation or File Open by Other Users (delete only) by Other Users (delete only).
-19	New Name is Duplicate (rename only).
-17	I/O Error

Table G - CLOSE ERROR CODES

1.2.3.3 Executive Read/Write

Executive tasks read and write records via message 8 (read) and message 9 (write). Only the BUFAD and BASE fields are used in these messages. Data is read from or written to the diskaddress specified by the 'THIS' field of the record pointed to by BUFAD. No checking is done on the validity of the address or anything else. This is the responsibility of the calling task.

1.2.3.4 Executive Obtain

If, while extending a file, an Executive task requires an additional disk record, it issues message 10. The File Manager uses the OBTAIN routine to allocate a sector, and the diskaddress of the sector is returned to the caller in the first 32 bits of the data portion of the message.

1.2.3.5 Executive Logon

Several File Manager calls require the caller to specify the diskaddress of a UFD. This address is obtained via message 11. A calling task places the twelve character user ID of a user in the first twelve bytes of the data portion of the message. The File Manager uses the LOGON routine to reach the MFD for the UFD address. If successful, the diskaddress of the UFD is returned in the first 32 bits of the message,

replacing the first 4 characters of the user ID. Logon error codes are:

STATUS	MEANING
-19	No Such User ID

1.2.4 Resident Supervisor Interface

The File Manager provides I/O services to HEP processes in much the same way as it does for Executive Tasks. For Executive tasks, I/O is performed directly into the caller's buffer. Since HEP data memory is not part of the Executive computer's address space, HEP I/O is handled differently.

HEP requests arrive via the Unibus-to-Switch Interface and the File Manager's helper task RF. The message received by the File Manager is a switch message (message 12) and contains one HEP word of data. The high 16 bits of the data word are a request code type with the following values:

0 - Logon

1 - Open

2 - Close

3 - Read Record

4 - Write Record

5 - Obtain Record

The low 32 bits of the word point to a 66 word I/O block. The last 64 words of this block are a disk record in the format previously discussed. The first two words contain I/O parameters and options. These words are described below.

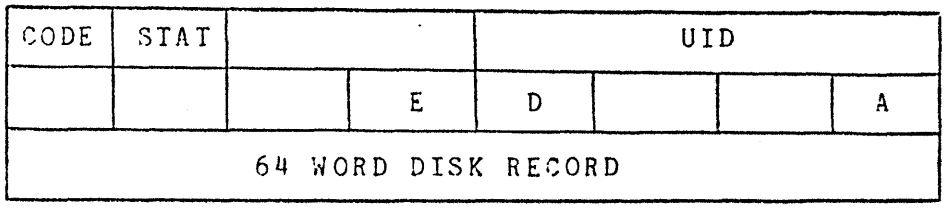


Figure H - HEP I/O REQUEST FORMAT

CODE - Request code, as enumerated above

STAT - Result status supplied by File Manager. Values as indicated for Executive Requests.

UID - Diskaddress of user file directory (open, close).

A - Requested access privileges, format as shown in Table C (open, close).

D - File history (open,close). Possible values are:

0 - Use Old File if Present,
Else Create New File

1 - Delete Old File if Present,
Create New File

2 - Use Old File

3 - Create New File
Fail if Old File is Present

E - File Disposition (close). Possible values are:

1 - Delete File

2 - Keep File

5 - Rename File

The File Manager reads the I/O block into a local buffer using the low speed bus (LSB) Interface to HEP data memory. The amount of data read depends on the request code in the switch message. After performing the request, all or part of the I/O block is written back to data memory with the LSB. Since the LSB is shared with other Executive tasks, the File Manager becomes uninterruptable during this transfer.

1.2.4.1 The Resident OPEN

A HEP supervisor opening a file does so by building a dummy file header in the I/O block. For old files, the file name and name length are required. The file name must be stored in byte swapped format as previously described. For newly created files, the RECSIZE and ACPRIV fields must be supplied. Unlike Executive opens, no option processing is provided. The only optional function is the provision of a user ID in square brackets at the start of of the File name.

If the open is successful, the File Manager copies the file header into the I/O block.

1.2.4.2 Resident CLOSE

Resident CLOSE is the same as Executive CLOSE except that the file disposition field is used to determine close action.

1.2.4.3 Resident READ/WRITE

Resident I/O is the same as Executive I/O. The 'THIS' field of the disk record in the I/O block is used to determine the diskaddress.

1.2.4.4 Resident OBTAIN

Resident OBTAIN uses the standard OBTAIN routine to allocate a disk record. The address of the record is returned in the 'NEXT' field of the disk record in the I/O block.

1.2.4.5 Resident LOGON

Resident LOGON obtains the 12 character user ID from the first 12 bytes of the disk record in the I/O block (word 2 and the high half of word 3). The user ID must be byte swapped. The diskaddress of the UFD is returned in the UID field of the I/O block (second half of word 0).

1.2.5 Operator Interface

The operator may send messages to the File Manager from the console terminal. The supported messages begin with a single character, as shown on the next page:

D - Toggle the debug switch. When debug is on, all received messages are listed on the console in octal.

Z - Shut down. The bitmap is written to disk and the File Manager executes a halt. No files are closed, and the File Manager may be restarted with the Executive Debugger (XD). If a file is being deleted when Z is entered, the message 'BUSY' will result and the File Manager will not shut down.

1.3 PASCAL Runtime Library

The PASCAL Runtime Library provides the interface between PASCAL READ, WRITE and associated I/O statements and the file manager. Components of the PASCAL runtime are linked into all Executive tasks. In addition to I/O, the PASCAL runtime provides the basic runtime environment and service subroutines for PASCAL tasks.

1.3.1 PASCAL Interface

The PASCAL runtime supports a set of I/O calls similar to that provided by standard PASCAL. Certain unneeded capabilities are not supported, and several extensions have been made.

Supported text output procedures are:

WRITE(CHAR:N) WRITE(F,CHAR:N)	Write the character CHAR to the file F or to OUTPUT, followed by N-1 blanks.
WRITE(I:N) WRITE(F,I:N)	Write integer I as a decimal string to the file F or to OUTPUT, followed by blanks to a total width of N. If N is negative, write I as an octal string.
WRITE(S:N) WRITE(F,S:N)	Write the character string S to F or to OUTPUT followed by blanks to a width of N characters. If N is less than the length of S, S is truncated. S may be a literal string.
WRITELN WRITELN(F)	Terminate a line.
BREAK(F)	Terminates a line but does not advance carriage to a new line.

Multiple I/O items may be combined in a WRITE request. If WRITELN is used with output arguments, the line is terminated after the last item.

Real and boolean output are not supported.

Supported text input procedures and functions are:

READ(CHAR) READ(F,CHAR)	Read one input character into CHAR from F or input.
READ(S) READ(F,S)	Read a string of input characters into S from F or INPUT. If the current line is exhausted before filling S, pad with blanks.
EOLN(F)	A boolean function which is true if the next character to be read is the end-of-line character.
EOF(F)	A boolean which is true when there are no more characters to be read.
READLN READLN(F)	Discard remaining characters in the current line (if any) and point to first character of next line.

Multiple I/O items may be combined in a READ request. If READLN is used with input arguments, the rest of the line is discarded after the items are filled.

Integer, real and boolean input are not supported.

The standard procedures GET and PUT may be used with text and non-text files. When used with a file connected to a console, GET exhibits non-standard behavior. The PASCAL standard requires that the first character of input be present immediately after a READLN. In this implementation, the first character is not present until a GET or READ operation is performed. The first GET consumes a dummy blank character. This character is not provided by READ; character strings consumed by READ contain only actual input text.

When used with non-text files, the record definitions acceptable to GET and PUT are restricted. The following record sizes are accepted:

<=136 Bytes
248 Bytes
496 Bytes

Other sizes would require data blocking facilities not present in the runtime routines.

The standard procedures RESET and REWRITE have been extended to allow opening specific disk files by name. The syntax of these procedures is as follows:

```
RESET(F,NAME,OPTIONS,V)
REWRITE(F,NAME,OPTIONS,V)
```

RESET is used to refer to a pre-existing file, while REWRITE causes the creation of a new file. The parameters are:

F — Name of the PASCAL file variable controlling this file.

NAME — A character array or literal string containing the file name. If omitted on a RESET, the file presently open is rewound. If name is a character array, the file name must be non-blank and padded to the right with blanks.

OPTIONS — A character array or literal string containing option characters. Each option is a slash followed by a single character. Available options are:

```
R - Read Access
W - Write Access
A - Append Access
T - Temporary File
N - Force New File
```

If OPTIONS is omitted, default options are supplied. For RESET, /R is the default. For REWRITE, /W and extend permissions are the defaults.

V — Integer variable. On entry, contains the record length to be associated with the file if the file is created. This may differ from the record size of the file variable. The file is processed based on the file variable if non-text, but the line length is is taken from V if the file is text. V is in HEP words (multiples of 8 bytes). On return from RESET/REWRITE, V contains open status. If V<0, the open failed and the value is the error code. If V>=0, it is the record size of the file, in HEP words.

Files may be closed, renamed or deleted by calls to the PASCAL procedures.

```
CLOSE(F)
RENAME(F,NAME,LEN)
DELETE(F)
```

CLOSE(F) is a standard procedure and may be used to close any file. RENAME and DELETE are nonstandard external procedures and must be declared with external declarations of the form:

```
PROCEDURE RENAME(VAR F:TEXT;VAR NAME:<CHARACTER ARRAY TYPE>;
  LEN; INTEGER).
    EXTERNAL;

PROCEDURE DELETE(VAR F:TEXT);
    EXTERNAL
```

F may be declared to be another type than TEXT, but the declaration must agree with the file type to be renamed or deleted.

Files will also be closed if a RESET or REWRITE is issued for their file variable specifying a new file name.

1.3.2 PASCAL Runtime Environment

A PASCAL Executive Task is loaded by the ROOT in a standard fashion. Of the 8 memory pages, the first four are reserved for code. Page 7 is mapped to I/O space. All PASCAL variables and working space is located in page 4. The first locations in this page, starting with location 100000 (octal) are used for control information. This information is shown in Figure 1.3.1.

MNEMONIC LOCATION	MEANING
$KORE=100000	; Top of Heap Space, Base of Stack Space
$FREE=100002	; Start of Linked List of Free Blocks of Length $NEWLN
$RESR5=100004	; INITIAL R5, SAVED BY MAIN
$RESR6=100006	; INITIAL R6
$NEWLN=100010	; Length of Storage Manipulated by New and Dispose
$FILBF=100012	; Start of Linked List of Free File Buffers
$FILTB=100014	; Start of Linked List of Free File Variables
$LOGCY=100016	; Users UFD Location (CYL)
$LOGDA=100020	; Users UFD Location (Track, Sector)
$FILE=100022	; File Variable Address Address
$SPACE=100024	; Dummy Blanks for Get Processing
$FMTSK=100026	; Task ID of File Manager
$MLEN=100030	; Length of Last Queue Message
$SINP=100032	; Standard Input File Variable Address
$SOUTP=100034	; Standard Output File Variable Address
$HEAP=100036	; Start of Heap Space

Figure 1.3.1 - PASCAL WORK AREA BASE

In a running PASCAL task, the register SP points to local variables, and register R5 points to the global variables. During initialization, file variables are allocated in the locations following $HEAP. The number of file variables is a compile-time parameter, normally 6. $KORE is set to point immediately after the file variables, and SP is set to point to location 120000, the top of page 4. The file variables for

INPUT and OUTPUT are initialized, and the main program is started. The main program immediately calls the NEW procedure to get space for global variables. This causes R5 to be set to the value of $KORE, and $KORE is incremented by the size of the globals. Immediately after this call, a task which uses disk files or the NEW/DISPOSE mechanism must call the FINIT procedure. This procedure allocates space for the requested number of file description blocks immediately above the global variables. Since FDB are more than 600 bytes long, only the number absolutely required should be requested. The other effect of FINIT is to define the block size used by NEW/DISPOSE. This size overrides the size specified in the NEW/DISPOSE call to prevent storage fragmentaton.

In operation, local variables of procedures use the stack, which grows downward from location 120000. Calls to NEW use the Heap, which grows upwards from the top of the FDBs. If these two areas collide, unpredictable runtime errors will occur, and the program must be recoded to use less storage.

1.3.3 Files and File Variables

Several types of files are supported by the PASCAL runtime. These are divided into text and non-text files.

1.3.3.1 Non-Text Files

Non-text files are accessed via GET and PUT, and must reside on disk. These files may be word or record files, as described below. For these files, the amount of data transferred by a GET or PUT is strictly determined by the file variable record size. The runtime is only capable of handling spanned records if the record size is less than 17 HEP words, which is why the record sizes are restricted.

Record files consist of a sequence of fixed length records, each an integral number of HEP words long. Word files have no external record structure and are normally processed a word at a time. The PASCAL Runtime ignores word or record structure for non-text files, treating them as record files with the record length determined by the file variable. A permanent record length may be specified by REWRITE which need not agree with the file variable size.

1.3.3.2 Text Files

Text files are of three types: console or queue files, word files on disk and record files on disk.

Console files interact with the system queue mechanism and are used to pass data between tasks or to and from terminals. An input line in the queue can contain a maximum of 118 characters of input. On output, data is broken into multiple messages if the data content exceeds 118 characters. The files INPUT and OUTPUT are console files by default. Other files may be declared as console files by using the distinguished file name 'TI:'. Since there is only one input queue and one output queue, having multiple files as console files yields unusual results. Output characters will be merged on a character by character basis, while input will appear at whichever file variable was most recently accessed.

Word text files contain variable length ASCII text lines. All lines contain a multiple of 8 characters, and are padded with blanks if necessary to become a multiple of 8. Since ASCII characters contain only 7 bits of data, the sign bit of the first character of a line is used to delimit lines. Lines span physical record boundaries. Manipulation of the sign bit is an automatic function of the runtime library, and data visible to the using program never contains a set sign bit. Since the Executive computer packs bytes right to left, while the HEP packs left to right, the runtime library performs a byte swapping operation on every physical record of word text files. This is done on both input and output so that disk data is always in HEP (left-to-right) order. The runtime pads lines with trailing blanks on read string operations, but does not strip trailing blanks on write string operations. There is no restriction on line lengths in word text files.

Record text files contain fixed length ASCII text lines. All lines contain the same number of characters, which must be a multiple of 8. Lines may span physical record boundaries. All 256 possible characters may occur in a record text file. Record lengths may not exceed 488 bytes. Output to a record text file will be truncated or padded with blanks as required to fit the record size. Input will be padded with blanks as truncated on read string operations. Byte swapping on input and output is performed to force disk data to be in HEP format.

BIT NAME	VALUE (OCTAL)	MEANING
S.EOF	100000	File has reached EOF.
S.EOLN	40000	Text file is at EOLN on input.
S.LAST	20000	Disk file is in the last physical record.
S.TXT	10000	File is a text file.
S.ERR	4000	Error encountered while processing file.
S.WAIT	2000	Queue or console file requires a physical read before supplying data.
S.END	1000	Output disk file has terminated a line, but not started a new line.

Table 1.3.3 - FILE VARIABLE STATUS BITS

The saved pointer field (V.SVP) is used if V.PTR is pointing to a space at end of line in order to locate the next actual data character.

V.BUF points to the start of the current logical record for a non-text file, or to the start of the buffer for a text file. If a logical record is spanned, V.BUF points to the psuedo-start of the logical record preceding the actual I/O buffer. V.BUF is advanced by each PUT or GET operation. For console files, V.BUF points to the first character of data.

V.LEN contains the record length to be used for this OPEN of the file. For text files, this number is -1. V.LEN is determined by the size of the record declared in the user's PASCAL program.

V.FDB points to the file descriptor block for this file. For a console file, V.FDB is 0. The I/O control block is described in the next section.

V.EOB points to the end of valid data in the I/O control block. Normally, this is the end of the physical record, but on the last record of the file, V.EOB points to the end of the data. For console files, V.EOB points to the end of the message.

1.3.3.3 File Variables

A file variable occupies 7 16 bit words in the Executive Computer. File variables are compiled into the runtime library. The number of file variables which may be simultaneously active is a compile time parameter in the runtime. An Executive task may have many files declared, but only a limited number of these may be simultaneously open. The RESET/REWRITE procedure establishes a pointer in the user's variables to the actual file variable. The format of the file variable is shown in Figure 1.3.2.

ADDRESS		
0	V.PTR	Pointer to Current Character or Record
2	V.STAT	Status Bits
4	V.SVP	Saved Pointer
6	V.BUF	Start of I/O Record
10	V.LEN	Record Length (-1 if Text File)
12	V.FDB	Pointer to File Description (0 for Console File)
14	V.EOB	Pointer to End of File I/O Buffer

Figure 1.3.2 FILE VARIABLE

V.PTR always points to the current character in a text file, or to a reserved location containing a blank, if the text file is at EOLN. For non-text files V.PTR points to the start of the record, either in the current I/O block, or to a work area used to collect spanned records.

The low byte of V.STAT contains the destination task number if the file is a queue or console file, and is unused otherwise. The high byte of V.STAT contains status bits, defined in Table 1.3.3.

1.3.3.4 File Descriptor Block

All disk files require a file descriptor block. A fixed pool of FDBs is created during PASCAL initialization, and disk file OPEN's in excess of the pool size cannot be accommodated. The format of an FDB is shown in Figure 1.4.

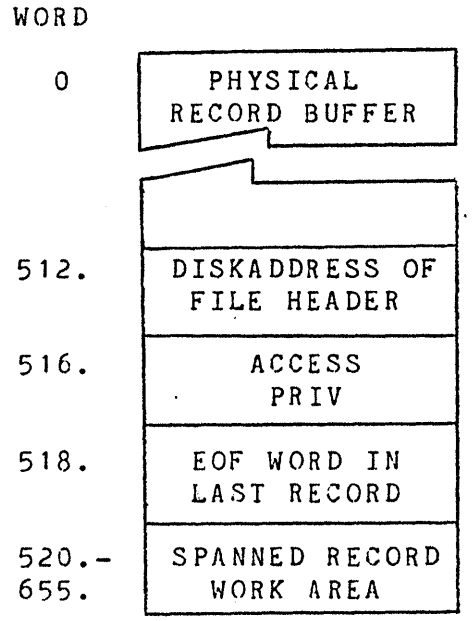


Figure 1.3.4 - FILE DESCRIPTOR BLOCK

1.3.4 Miscellaneous Runtime Support Routines

Several utility routines are included in the runtime package. These are described below.

1.3.4.1 FINIT

Declaration:

```
PROCEDURE FINIT(NFDB,NEWSIZE:INTEGER)
```

FINIT is called once by every main program to define the number of FDBs and the size of the area returned by NEW.

1.3.4.2 LOGON

Declaration:

```
FUNCTION LOGON(VAR UID:ARRAY[0..11]
  OF CHAR)
```

LOGON invokes the file manager to logon as the user UID. If successful, the locations USERDA and USERCY in the runtime base are set up to point to the UFD. If logon is successful, LOGON returns TRUE, else FALSE.

1.3.4.3 LINLEN

Declaration:

```
FUNCTION LINLEN:INTEGER
```

LINLEN returns the length of the last console input line.

1.3.4.4 SETID

Declaration:

```
PROCEDURE SETID(F:TEXT;TSK:INTEGER)
```

SETID sets the destination of the console file F to TSK.

1.3.4.5 GETTSK

Declaration:

```
PROCEDURE GETTSK(VAR N:INTEGER);
```

GETTSK obtains the task number of the task whose two-character ID is placed in N. The value is returned in N as two bytes. The low byte of N is the task number of the requested task. The high byte is the task number of the task calling GETTSK.

1.3.4.6 ERR

Declaration:

```
FUNCTION ERR(F:FILE OF...):BOOLEAN
```

ERR tests the S.ERR bit in V.STAT of the file and returns TRUE if an error has occured, otherwise FALSE.

1.3.4.7 GETLOC

Declaration:

```
PROCEDURE GETLOC(VAR F:TEXT;VAR L:
  ARRAY[0..4]OF INTEGER);
```

GETLOC returns the present file position in the file F. This information may be used in a subsequent call to SETLOC.

1.3.4.8 SETLOC

Declaration:

```
PROCEDURE SETLOC(VAR F:TEXT;VAR L:
  ARRAY[0..4]OF INTEGER);
```

SETLOC sets the file position of F to the information contained in L. SETLOC may only be used for word files with read access only. The information in L must correspond to the file presently open as F. Alteration of the information in L between the call to GETLOC and the call to SETLOC will probably result in system failure.

1.4 Tape Manager

1.4.1 Overview

The Tape Manager runs as a task under the Executive and performs all tape handling functions. The task may be accessed only from the operator's console. Commands to rewind, space over files, and position to end of volume (end of the last file) are supported. Individual files may be copied to or from tape, and entire UFDs may be dumped to or restored from tape. The Tape Manager reads and writes the tape unit directly by manipulating the appropriate I/O locations in the PDP-11 memory.

1.4.2 Tape Format

Two format modes are supported for data on tape: record mode and dump mode.

1.4.2.1 Record Mode

In record mode each physical tape record contains one logical record as defined when the disk file was created. The file is terminated on tape by a file mark. Record mode format is created only when an explicit file name is specified to be copied and the file is defined to be of record type. Record mode format may be read only by explicitly specifying a file name, and results in the creation of a record type disk file whose logical record size is defined to be the length of the tape record read. Therefore it is not necessary (or possible) to supply a record size in the command to the Tape Manager.

1.4.2.2 Dump Mode

Dump mode format is used for copying word type files and for dumping and restoring UFDs. In dump mode each physical tape record is 514 bytes long, except for the last record in a file, which is a short record. Dump mode format is created either by explicitly specifying a file which was created as a word file (logical record length = 0) or by requesting a UFD dump (dump all files in the specified User File Directory).

1.4.2.2.1 Word Files

In the case of copying a specific word file, each 514-byte tape record contains a physical disk record, including the 16-byte block header and followed by 2 bytes whose content is undefined. The last, short record contains the last physical disk record including header, truncated to 2 bytes past the end of the actual file data. Therefore the length of the short record is always 2 bytes more than a multiple of HEP words. In the case where the actual file data exactly fills a physical disk record, the short tape record following the last block will be 18 bytes long (16-byte header plus 2 trailing bytes). The file is terminated on tape by a file mark.

1.4.2.2.2 UFD Dumps

The first record of a UFD dump is an identifying record containing the ASCII UID (User Identification) in the first 12 bytes. The high byte of the UID has its sign bit set to mark this file as a UFD dump. The remainder of the record is undefined. Following the UFD identifying record are all the header records (from file HEADER for this UID) except the header record for file HEADER. These records are used to identify the files contained in the dump. Each header record is a physical disk record, including the 16-byte block header, followed by 2 undefined bytes. Following the last header record is a short record, 18 bytes long, which indicates the end of header records. The content of each file in the UFD follows, in the same order as the header records. Regardless of defined logical record length, each file in the dump has the same format on tape as a word file. Each tape record contains a physical disk record, including the 16-byte block header and followed by 2 bytes which in this case contain the defined logical record length. The last record in each file is a short record, indicating the end of the file. The last file in the dump is followed by a file mark which indicates the end of the dump for this UFD. If multiple UFD's

were specified, the next UID follows the file mark.

1.4.2.2.3 End of Volume

Following the last file on a tape (either single file or UFD dump) is an additional file mark, resulting in two consecutive file marks. This indicates the end of data on the tape. The Tape Manager will not allow reading or positioning past the double file marks. More files may be written to tape, resulting in writing over the second file mark. The new last file is then terminated by two consecutive file marks.

1.4.3 Commands

The Tape Manager is accessible only from the operator's console, directed by commands in the form of messages prefaced by 'MT:' (the task ID for the Tape Manager). A command may specify tape positioning only, may direct that the tape be either read or written with a string of one or more file names or UIDs, or may direct the Tape Manager to read commands from a file.

1.4.3.1 Tape Positioning

Each tape positioning command is prefaced by a slash ('/') to distinguish it from a file name. The command mnemonics and corresponding operations supported are as follows:

'/RW' - Rewind; positions the tape at load point.

'/AP' - Append; positions the tape at the second of the two consecutive file marks which indicate end of volume; at this point, more files may be written to the tape, overwriting the second file mark and resulting in a new end of volume.

'/FFn' - Forward file; positions the tape immediately past the nth file mark forward from the tape's current position; if n is not specified, 1 is assumed; if end of volume is encountered, the command is terminated leaving the tape positioned the same as for '/AP'.

Multiple tape positioning directives may appear in a single command line; each leading slash serves as a delimiter. Each directive is performed in order, left to right.

Examples:

'MT:/AP' - Positions the tape at end of volume.

'MT:/RW/FF2' - Positions the tape at the start of the third file or dump on the tape.

1.4.3.2 Writing a Tape

Writing to tape is indicated in a command line by an equal sign (=) preceding the list of files or UIDs to be copied to tape. Tape positioning commands may precede the equal sign; these will be performed before writing the tape. A UID specification must be enclosed in square brackets ('[]'). A file specification may include its UID (in square brackets), otherwise the last UID appearing in a file specification (in the same or a previous command line) is assumed. Multiple file or UID specifications in a command line must be separated by commas.

Tape positioning commands may be interspersed in the list of file or UID specifications and will be performed in the order encountered. When a tape positioning directive follows a file or UID specification, the slash may serve as the delimiter, so that the comma following the file or UID specification may be omitted.

Examples:

'MT:/AP=[300302]' -

Writes the UFD dump for 300302 at the end of existing data on the tape.

'MT:=[001001]HEPOS,CONTROL/RW' -

Copies the two named files from UID 001001 to tape, then rewinds the tape.

1.4.3.3 Reading a Tape

Reading from tape is indicated by an equal sign ('=') as the last character in the command line. Preceding the equal sign is a list of one or more file or UID specifications, separated by commas. A UID specification must be enclosed in square brackets ('[]'). A file specification may include its UID (in square brackets), otherwise the last UID appearing in a file specification (in the same or previous command line) is assumed.

Tape positioning commands may be interspersed in a list of file specifications and are performed in the order specified. In the case of restoring UFD dumps, the specified UIDs are restored in the order in which they occur on the tape regardless of their order in the command line; therefore tape positioning directives are not useful except at the end of the UID list. When a tape positioning directive follows a file or UID specification, the slash may serve as the delimiter, so that the comma following the file or UID specification may be omitted.

Examples:

```
'MT:/FF,[300302]ABC/FF2,XYZ=' -
```

Copies the second and fifth files from tape into UID 300302.

```
'MT:[001001],[300300],[300302]/RW=' -
```

Restores the files for UIDs 001001, 300300, and 300302, then rewinds the tape.

1.4.3.4 Indirect Command File

An indirect command file is a text file, each line (logical record) of which is a command line as described in the preceding sections on reading, writing, and positioning tapes. The leading 'MT:' found in console commands is ommitted from commands in an indirect command file. The Tape Manager is directed to process the commands in a command file by a message from the operator's console of the form:

'MT:@[UID]file name'.

The Tape Manager then opens the specified file and reads and performs each command until end of file is reached. A command file may contain any legal Tape Manager command except '@...'; that is, indirect command files may not be nested.

1.4.3.5 Terminating Command Processing

When the Tape Manager is finished processing a command line or an indirect command file, the message 'MT:' is displayed on the operator's console. If the Tape Manager encounters an error while processing a command, an error message is displayed on the operator's console and processing of the command line or indirect command file is terminated.

If it is desirable to prematurely terminate the processing of a command line or indirect command file, this may be accomplished by sending any message to the Tape Manager. Processing of the current command or file of commands will halt and the new message will be processed as a command.

1.4.4 Functional Description

In record mode, disk files are read or written with standard calls to the runtime I/O routines. In dump mode, however, since tape records are equivalent to physical disk records, the overhead of unblocking, reblocking, and moving data is avoided by transmitting each record between the I/O buffer and the tape directly. This is accomplished by manipulating the disk file variable (in assembly language) to point to the end of the block after each tape transfer, forcing the next call to the appropriate I/O routine to do a physical read or write of the disk file.

1.4.5 Error Messages

MESSAGE	MEANING
Unknown Account - UID -	The UID in a command is not known to the system.
Command Error -	Command not recognized.
Parameter Error -	Invalid parameter in command.
Illegal Function -	
Bad Tape Format -	The tape format does not match the command being processed.
End of Volume on Tape -	Premature end of volume reached while processing a command.
Tape I/O Error:nnnnnn -	Bad status after a tape operation; nnnn is the status.
Tape on Write Lock -	Attempt to write on a tape which has no write ring in.
UID Table Overflow -	The list of UIDs to be dumped or restored is longer than the program can accomodate; remaining UIDs will not be processed.
Bad Directory -	An error was encountered in processing a file header.
Command Aborted -	The current command line or indirect command file has been prematurely terminated due to error or operator intervention.
Tape Unit Not Ready -	Tape unit is off-line.

For R, C, P and D requests, if =<value> is specified, the contents of the location specified by <Starting Address> are replaced by <Value>.

For all requests, the processor to be accessed is specified by the processor parameter. If this parameter is omitted, processor 0 is assumed.

In the event that the <Filename> parameter is provided, the HEP Debugger will send the input message back to the originating task when the operation is complete. This allows the Baton Monitor to determine when a dump is complete and the HEP resources may be freed.

1.5 HEP Debugger

The HEP Debugger is used to examine and modify HEP memory and examine the PSW queue. It is normally used by the Batch Monitor for taking user dumps, but can also be used from the operator's console.

All HEP Debugger functions use the KI task and code in the HEP Kernel to obtain HEP related information. Thus the HEP Debugger cannot be used if the Kernel or the UNIBUS to switch interface is not working. In these cases, program and data memory may be examined using the IML maintenance task (MP).

1.5.1 Command Format

The HEP Debugger responds to single line text commands with the following syntax:

[<Request Type>]<Starting Address>[,Ending Address][.P<Processor>]

{ [/<Filename>] } (Brackets denote optional parameters)
{ [=<value] }

Valid request types are a single alpha digit drawn from:

R - Register Memory
C - Constant Memory
P - Program Memory
D - Data Memory
S - Program Status Words

If <Request Type> is omitted, and the leading character of the starting address is numeric, data memory is assumed. If <Ending Address> is omitted, it defaults to <Starting Address>. If the /<Filename> qualifier and the =<value> qualifier are absent, for R, C, P, and D request types, all memory locations between <Starting Address> and <Ending Address> are displayed. For S request types, <Starting Address> is taken as a task number, and all PSW's with that task number are displayed.

If /<Filename> is specified, the requested data is written in binary format to the file specified. For S requests to a file, all PSW's are dumped.

1.7 Editor

1.7.1 Overview

The HEP System Editor is a line-oriented text editor which operates on sequence numbered word type or on sequenced or non-sequenced record type files of text. The Editor also performs file utility functions such as listing, copying, renaming, or deleting files, listing the user's directory, and submitting a file as a job.

Options for terminating an edit session include canceling the edit (preserving the original file), saving the updated file under a new name (preserving the original file), and replacing the original file with the updated file. In the latter two cases, the updated file may be saved as a record file, with or without sequence numbers, or as a sequence numbered word file. The Editor supports the inclusion of lines of text from an auxilliary input file, which may be the file being editted.

A separate Editor task services each terminal or port in the HEP system; all the Editor tasks share the same program code but each has its own data space.

1.7.2 Commands

The Editor is the only task which communicates with terminals other than the operator console, therefore messages (commands) to a terminal Editor are not prefaced by any task identifier. However, the operator console is also serviced by an Editor task, and its identifier 'ED:' must preface Editor messages. Commands to the Editor fall into three categories: log on/off commands, file utility commands, and edit commands. Any command which results in substantial output to the terminal may be prematurely terminated by typing a carriage return.

In the following command descriptions the syntax of each command is given. Capital letters in command mnemonics represent the minimum portion of the word to be supplied for command recognition.

1.7.2.1 Log On/Off Commands

When no user is logged on to an Editor task, that task monitors any signal on the line looking for a carriage return at various line speeds to determine the speed at which the terminal is operating. Therefore before logging on, it is necessary to type carriage return until the system responds with a greeting.

1.7.2.1.1 Log On

Hello [uid]

The user whose identification number appears in the backets is logged onto the system. The Editor responds by displaying a greeting with the current date and time.

1.7.2.1.2 Log Off

Bye

The user is logged off the system. If an edit was in progress, it is aborted. The Editor responds by displaying the date and time.

1.7.2.1.3 Assistance

A user who is logged on may obtain a display of all the Editor commands and syntax by typing '?'.

1.7.2.2 File Utility Commands

1.7.2.2.1 List Directory

LD

The name of each file in the logged-on user's file directory is displayed, including the logical record length, the file size in blocks, and the creation, last modification, and last access dates.

1.7.2.2.2 Copy a File

CF file specification, file name

A file, named by the second parameter, is allocated in the user's UFD and the contents of the file represented by the first parameter (which may include a UID) are copied into it. The output file must not already exist.

1.7.2.2.3 Delete a File

DF file name

The named file is deleted from the user's UFD and the space allocated to the file is freed.

1.7.2.2.4 List a File

LF file specification

The contents of the file (which may be in another UFD) are listed on the terminal.

1.7.2.2.5 Rename a File

RF file name, file name

The file in the user's UFD named by the first parameter is renamed to the second parameter. The second file name must not already exist.

1.7.2.2.6 Submit a Job

SUbmit file specification

The Editor builds and sends to the Reader task a message containing the UID and file name specified, to be submitted as a job stream. If the file specification does not contain a UID, the logged on user's UID is used. If the parameter is not provided, the updated version of the file being editted is submitted.

1.7.2.3 Edit Commands

In the following syntax descriptions, parameters appearing in square brackets are optional. The letters N, M, L, and I represent numbers.

1.7.2.3.1 Edit a File

Edit file name

The Editor initiates an edit session for the specified file. This includes copying the file to a word file if it is a record file and assigning sequence numbers if it is not sequenced.

1.7.2.3.2 Copy Lines

Copy N[-M][/file specification],L[,I]

The Editor copies line numbers N through M from the specified file, inserting them in the file being editted, starting at line number L and incrementing by I. If M is omitted, only line N is copied from the file. If the file specification is omitted, the lines are copied (replicated) from the file being editted. If I is omitted, the last increment specified in any command is used.

1.7.2.3.3 Move Lines

Move N[-M],L[,I]

Lines N through M are inserted starting with line number L, incrementing by I, and the lines are deleted from their former position. If M is omitted, only line N is moved. If I is omitted, the last increment specified in a command is used.

1.7.2.3.4 Insert a Sequence of Lines

Sequence L[,I]

Sequence insert mode is established, starting with line number L and incrementing by I. The user is prompted for each line of text by a display of the next line number in sequence. The process is terminated by typing a carriage return as the only character on a line, or by overlapping an existing line number in the file.

1.7.2.3.5 Replace a Text String

Replace /string 1/string 2/[N[-M]]

The first occurence of string 1 is replaced with string 2 in every line in the range. If M is omitted, the range is the entire file. The slash delimiter may be any character.

1.7.2.3.6 Delete Lines

Delete N[-M]

The specified line or range of lines is deleted.

1.7.2.3.7 Direct Insert

N text

Line N is placed in the file, containing the text following the line number N. If line N already exists, its contents are replaced by the new text.

1.7.2.3.8 Direct Delete

N

Line number N is deleted from the file.

1.7.2.3.9 Find a Text String

Find /string/[N[-M]]

The Editor searches the specified range of lines for occurences of the string and displays on the terminal all lines containing the string. The slash delimiter may be any character. If M is omitted, only line N is searched; if N is also omitted, the entire file is searched.

1.7.2.3.10 List Lines

```
List [N[-M]]
```

All lines in the file in the specified range are displayed on the terminal. If M is omitted, only line N is displayed; if N is also omitted, the entire file is displayed.

1.7.2.3.11 Renumber the File

```
Number N[-M],L[,I]
```

The Editor renumbers the specified range of lines, assigning new sequence numbers starting with L and incrementing by I. If M is omitted, only line number N is renumbered. If I is omitted, the last increment specified in a command is used. If the renumbering of the range would cause the file to be out of sequence (i.e. lines following the range have lower numbers, or lines preceding the range have higher numbers), then no renumbering is done.

1.7.2.3.12 Save the Changed File

```
SAve [file name][,Rn][/S]
                or
SAve [file name][,W]
```

The updated version of the file being editted is saved. If file name is specified, the file is saved under that name, and the original file is retained, otherwise the updated file replaces the original file. If Rn is specified, the file is saved as a record file with a logical record length of n/8 HEP words; if /W is specified, the file is saved as a word file; otherwise the file is saved as the same file type as the original file. If /S is specified on a record file, the sequence numbers are saved as part of the file and the logical record length must allow for the 1-HEP-word sequence number per record. If /S is not specified, then the sequence numbers are saved only if the original file was sequenced. The save command does not take the user out of edit mode; updating may continue from the point of saving the intermediate file.

1.7.2.3.13 End the Edit Session

Off

The Editor terminates an edit and closes all files. If the edit file has been modified, a save command must be performed before the off command is accepted.

1.7.2.3.14 Cancel the Edit

ABort

The current edit is terminated and the updated file is deleted. The original file is retained.

1.7.3 Functional Description

The Editor applies updates to a temporary copy of the file; the actual file is not affected until the user issues the save command. It is always possible to cancel an edit session, leaving the file as it was before the edit started.

During editting, an input and an output file are open. The input file is initially the original file unless the original file is a record file; the output file is a temporary file. None of the file being editted is kept in memory; the first time the input file is read, a table of disk addresses is built, evenly distributed over the file. Then for each command, the disk address from the table whose sequence number is nearest to but less than the line number referenced is used to position the file. From that point the file is read sequentially to process the command. Updated lines are kept in a table in memory and override the corresponding lines in the file. Inserted lines are also kept in the table. When the table becomes full, a portion of the input file is written to the output file, incorporating the changes in the table, until the change table is partially empty. From this point on, the output file is also accessed for input, and the corresponding portion of the input file is ignored. The renumber command, however, results in the entire updated file being written to output with the new sequence numbers. Any time the last of the updated file is written out, the old input file is closed, the output file becomes input, and a new output file is allocated. All files allocated by the Editor are allocated as temporary files so that they are deleted when closed.

When the user saves an edit, the original file is deleted if necessary and the final output file is renamed to the desired name, making it a permanent rather than a temporary file.

When a record file is editted, it is initially copied to a temporary word file. Line numbers are assigned unless the record file contains embedded sequence numbers at the end of each logical record. Embedded sequence numbers are identified by the high-order bit being on in the first byte of the 8-byte number. If present, the embedded sequence numbers are extracted and used as the line numbers as the file is copied to a word file. The sequence numbers in the text are replaced by blanks in the word file.

1.7.4 Running a Job From the Editor

When the submit command is used to cause a file which is a job stream to be submitted as a job, the Editor builds and sends a command to the Reader task. The Editor uses the message buffer for terminal output and builds all fields of the message directly. The message text consists of a UID (either supplied in the submit command or taken from the user's log-on) and a file name. The Editor does the trap to send the message, then does a trap to obtain a new message buffer. This new message buffer becomes the terminal output buffer.

When a job submitted by an Editor has completed execution, a completion message is received by the same Editor from the Batch Monitor task. Informative messages of this sort are distinguished from user commands by the presence of a backslash (line cancel character) as the first character of the message. These messages are simply displayed on the terminal.

1.8 Batch Monitor

1.8.1 Overview

The Batch Monitor runs as a task under the Executive. It is responsible for job scheduling, resource management, and processing status changes for jobs in execution. The Batch Monitor receives job descriptor messages from the Reader task and maintains a queue of jobs to be run. As resources are available, the Batch Monitor removes jobs from the queue and initiates execution; a queue of jobs in execution is also maintained. At job step termination, the Batch Monitor initiates the dump function if requested. The Batch Monitor sends completed jobs to the Writer task for printing. Operator commands are available to examine the queues, re-order the job queue, suspend or cancel jobs, and change HEP partition sizes. The operator can also instruct the Batch Monitor to quiesce the system by shutting down the Reader task and not starting any new jobs.

1.8.2 Commands

The Batch Monitor is accessible only from the operator's console, directed by commands in the form of messages prefaced by 'BM:' (the task ID for the Batch Monitor). There are two categories of commands: job-related and system-related.

In the following command descriptions the syntax of each command is given. Capital letters in command mnemonics represent the minimum portion of the word to be supplied for command recognition.

1.8.2.1 Job-Related Commands

In the following syntax descriptions 'nnnn' represents a four-digit system-assigned job number which uniquely identifies the job.

1.8.2.1.1 Move Job to Top of Queue

Next nnnn

Job number nnnn is moved to the top of the job queue so that it will be the next job started.

1.8.2.1.2 Suspend Job Execution

SUspend nnnn

If job number nnnn is in execution, the Batch Monitor sends a suspend message to each task in the job which is currently active.

1.8.2.1.3 Resume Job Execution

Resume nnnn

If job number nnnn is in execution, the Batch Monitor sends a resume message to each task which is currently paused.

1.8.2.1.4 Cancel a Job

CAncel nnnn

If job nnnn is in execution, the Batch Monitor sends a cancel message to each task in the job. If job nnnn is in the job queue, it is removed so that it will not be executed.

1.8.2.2 System-Related Commands

1.8.2.2.1 Set HEP Partition Sizes

Partition p,m,s1,s2,s3...

The Batch Monitor sends a message to the Kernel in PEM number p requesting that memory type m be partitioned according to the sizes s1, s2, etc. The number of sizes specified must be less than or equal to the maximum number of partitions allowed in the memory type. The sum of all the sizes must be less than or equal to the physical amount of the specified memory on the specified PEM. Partitions may be reconfigured while jobs are running as long as the boundaries of the active partitions are not affected. If the requested partitioning cannot be performed because of violation of any of the above criteria, the partitions message is modified to reflect the actual current partition sizes. In any case, the message is returned to the Batch Monitor and is then displayed in tabular format.

If the operator simply wants to know what the current partition sizes are, an easy method is to type a set partitions command with a single size field which is larger than the total amount of memory of that type.

1.8.2.2.2 Set Control Card Processor

CC [uid]file name

The Control Card Processor is always loaded from a file. If the user wishes to specify some file other than the system default (i.e. to try out a new Control Card Processor program), this command must be used to direct the Batch Monitor to use the desired file for loading the Control Card Processor task. The named file remains the Control Card Processor file until the Batch Monitor receives another CC command.

1.8.2.2.3 Display the Job Queue

Display Queue

The Batch Monitor lists on the operator's console all jobs waiting to be run, showing the job number, name, memory requirements, and process count requirements.

1.8.2.2.4 Display the Jobs in Execution

Display Active

The Batch Monitor lists on the operator's console all jobs in execution, showing the job number, name, data memory partition number, and the PEM number, task number, partition numbers, process count, and task status for each task in the job. Task status can be 'L' for loading, 'A' for active (running), 'P' for paused, or 'D' for dormant (no code in this task).

1.8.2.2.5 Quiesce the System

STOp

The Batch Monitor does not start any more jobs from the job queue, and sends a message to the Reader to stop reading job streams. Other Batch Monitor activities proceed normally.

1.8.2.2.6 Resume Normal System Operation

STArt

This command is the reverse of STOP; the Batch Monitor sends a message to the Reader to continue reading jobs, and the Batch Monitor resumes initiation of jobs from the job queue.

1.8.3 Inter-task Messages

In addition to commands from the operator's console, the Batch Monitor receives (and sends) other types of messages. These messages support three basic Batch Monitor activities: sending and receiving HEP messages, sending and receiving jobs, and taking dumps of jobs.

1.8.3.1 HEP Messages

When the HEP sends a status change message on behalf of some job, the Batch Monitor receives a "switch message" from the interface task. A switch message consists of one HEP word of data, followed by 32 bits of control information. In this case, the HEP word contains the data memory address of the HEP message header. The Batch Monitor then reads the HEP message header via the low speed bus, sets good status in the header, and writes the header back to data memory via the low speed bus. The second word of the HEP message header contains the length and start address of the message data, if any. If there is data, the Batch Monitor reads it via the low speed bus, then releases the original switch message by sending it back to the interface task as a reply.

The Batch Monitor then processes the HEP message according to its type. The types of messages that might be received from the HEP are pause, normal termination, abnormal termination, and system error. The message header contains, in addition to message type, the PEM

number and task number from which it came. Except in the case of a system error message, the PEM and task number are used to identify the job associated with the message, by searching the execution queue. Since HEP messages come from the supervisor task, 8 is subtracted from the header task number to get the corresponding user task number. If there is any message data it is displayed on the operator console and written to the job's log file. If the message was a normal or abnormal termination, the Batch Monitor sends a cancel to any other tasks in the job to force them to terminate.

In the case of a system error message, the Batch Monitor displays the message data on the operator's console and halts.

When the Batch Monitor wants to send a message to the HEP, it must first seize control of the interface. This is done by sending a a switch message to the appropriate interface task (KI) with the target PEM number in the first 16 bits of the data word. The interface task replies with the data memory address of the interface area in the switch message data word. The third and fourth words of the interface area are where HEP-bound message headers are written. The Batch Monitor writes its message header into this area and sends an activate type switch message.

SUMMARY OF BATCH MONITOR CONSOLE COMMANDS

Job Related Commands

Next nnnn	- Move job number nnnn to top of job queue.
SUspend nnnn	- If nnnn is in execution, suspend all tasks in the job.
REsume nnnn	- If nnnn is suspended, resume execution of its tasks.
CAncel nnnn	- IF nnnn is in execution, cancel all of its tasks.

System Related Commands

Partition p,m,S1,S2,S3...S7 -

Set partitions in processor p, memory type m to the values in S1...S7.

CC[uid]Filename	- Set the name of the CONTROL CARD PROCESSOR load module equal to [uid] filename.
Display Queue	- Display the contents of the Job Queue (jobs waiting to be run).
Display Active	- Display the status of all active jobs.
STOp	- Do not process any more new jobs.
STArt	- Reverse of STOP, allow Batch Monitor and Reader to continue processing new jobs.

Other Console Commands

RD:[uid]Filename	- Read the jobfile specified and submit it for execution.
PR:[uid]Filename	- Enter the file specified in the print queue.
PR:CAncel nnnn	- Remove job nnnn from the print queue.
PR:CAncel	- Stop printing only of file currently being printed.
PR:COntinue	- Resume printing (after printer has gone off-line for some reason).

HEP DEBUGGER - HD:

Examine/Modify Memory -

[<Memtype>]<Start Addr.>[,Endaddr.][,Pn] [/<Filename>]
[=Value]

Memtype = R, C, P, D or S - Default = D

Pn = Processor Number - Default = P0

/<Filename> = Optional Output to <Filename>

'= Value' = Modify Word to Value Specified

TAPE MANAGER - MT:

/RW - Rewind tape.

/AP - Append.

/FFn - Forward file, n file marks . 'n' default = 1.

= <[uid]Filename> - Write <[uid]Filename> to tape. Default [uid] = last [uid] encountered or [001001].

<[uid]Filename>= - Copy <[uid]Filename> from tape. Default [uid] = last [uid] encountered or [001001].

= [uid] - Write UFD dump for [uid] to tape.

[uid] = - Copy UFD dump from tape to [uid].

Tape Manager directives may appear in any order in a command line, except that only one '=' may appear. This must be either at the front of the command line (for writing to a tape) or at the end of the command line (for reading from a tape).

EDITOR - ED:

Editor commands related to file manipulation and batch processing of jobs. Operator must be logged on to the Editor.

BYE	- Log off the Editor.
CF F1,F2	- Copy file F1 to file F2.
DF F1,<F2,...Fn>	- Delete files specified. More than one file may be named on a command line.
HEL N1	- Log on the Editor with usercode N1.
LD	- List file directory for your usercode.
LF F1	- List the contents of File F1.
RF F1,F2	- Change the name of File F1 to F2.
SUbmit F1	- Submit the (Control Card) File F1 as a job.
?	- Print complete summary of Editor commands.

NOTE: Cut out this '16' and tape it as a subscript on page 2 under HEP DEBUGGER section, the line "'= Value' = Modify word to value xx specified" where the xx's are.

ALSO, draw a bracket on the left hand side between ,Pn] and [/<Filename>] in the third line under HEP DEBUGGER and on the right hand side after [/<Filename>].

1.11 Disk Builder

The Disk Builder is a utility task used to initialize and reconfigure the disk file system. It is not normally part of an operational system system, but is always part of the bootable system on a distribution tape. The Disk Builder is unique in that it accesses the disk directly, rather than through the File Manager. The Disk Builder contains no command error checking of any description, and errors or misuse of its commands may destroy the file system. The File Manager in a system must be shut down while the Disk Builder is used. Disk Builder commands are described below. All commands are single letters.

1.11.1 Format Disk

The F command causes the disk to be formatted and verified. Sectors with read errors are flagged.

1.11.2 Initialize Disk

The I command causes the disk allocation bitmap, the master file directory and the user file directory for user 000 000 000 000 to be built. This readies the disk for use by the File Manager

1.11.3 Create User File Directory

The U command prompts for a user ID, and creates a UFD for this user. No check is made for duplicate UFD's.

1.11.4 Logon

The L command prompts for a user ID, and uses that ID for all subsequent file references.

1.11.5 Build Bootstrap Sectors

The B command builds the bootstrap sectors (cylinder 0, track 0, sectors 0 and 1). Sector 0 contains a hardware bootstrap for the ROOT, while sector 1 contains pointers to the MFD, the bitmap and the disk addresses and boot parameters for the separately compiled Executive tasks. Sector 1 is used by Root initialization to load the Executive tasks. The B command obtains the Sector 0 bootstrap from the file "HWBOOT.TSK" in the current user ID. The command prompts for a series of task ID's and boot parameters. For each task, the following boot parameters must be supplied:

Filename - The name of an existing file in the current UFD or ^ taskid, where taskid is a previously defined task. If ^ taskid is entered, the bootstrap entry is flagged so that the code for taskid will be shared with this task and not loaded again.

Debug Mode - If debug mode is Y, the initial flags of the task will offset so that the task will not automatically start when the system is booted.

If the bootstrap sector already contains boot information, the user is prompted for each existing entry to determine whether to change that entry. After all existing entries are processed, new entries may be added. Note that since the boot sectors contain absolute disk addresses rather than file names, it is necessary to rebuild the boot sectors whenever an Executive task or the Root is changed.

1.11.6 Set Date

The D command prints the current date (MM/DD/YY) from the calendar clock and accepts a new date. Typing an invalid date will hang the task and require a reload.

1.11.7 Set Time

The T command displays the current time from the calendar clock (HHMM:SS) and accepts a new time (HHMM). If an invalid time is is entered, the task will hang and the system must be reloaded.

1.11.8 Make Distribution Tape

The M command will write a magnetic tape consisting of the tape bootstrap contained in the file "MTBOOT.TSK", followed by a core image of the current incore system. This tape may be hardware booted. For the tape to be useful, the system must be inactive while the tape is being made, and the File Manager must be shut down.

1.11.9 Read Absolute Sector

The R command allows the operator to read disk sectors. The command prompts for the cylinder head, and sector to be read. After the sector is read, the command prompts for an output mode, starting data word and word count to print.

Legal modes are A (ASCII), D (Decimal), H (Hexadecimal) and O (Octal).

1.11.10 Set Indirect File

The @ command prompts for a file name and obtains subsequent commands from that file until EOF or error.

1.11.11 Shut Down

The Z command writes out the current bitmap in preparation for activation of the File Manager.

1.11.12 Disk Build Procedure

This section describes the sequence of operations required to build a disk from the distribution tape.

a) Boot the distribution system from tape;

b) Set the date and time;

c) Format the disk (if required);

d) Initialize the disk;

e) Define required UFD's, including 001001;

f) Using the Executive Debugger, set the flags of the task FM and MT to zero, allowing them to run;

g) Using the Tape Manager, copy the contents of UFD 001001 from the distribution tape;

h) Shut down the File Manager;

i) Logon as 001 001;

j) Use the Boot command to initialize the bootstrap sectors;

k) Shut down the Disk Builder;

l) Halt and reboot the system from disk.

1.11 Disk Builder

The Disk Builder is a utility task used to initialize and reconfigure the disk file system. It is not normally part of an operational system system, but is always part of the bootable system on a distribution tape. The Disk Builder is unique in that it accesses the disk directly, rather than through the File Manager. The Disk Builder contains no command error checking of any description, and errors or misuse of its commands may destroy the file system. The File Manager in a system must be shut down while the Disk Builder is used. Disk Builder commands are described below. All commands are single letters.

1.11.1 Format Disk

The F command causes the disk to be formatted and verified. Sectors with read errors are flagged.

1.11.2 Initialize Disk

The I command causes the disk allocation bitmap, the master file directory and the user file directory for user 000 000 000 000 to be built. This readies the disk for use by the File Manager

1.11.3 Create User File Directory

The U command prompts for a user ID, and creates a UFD for this user. No check is made for duplicate UFD's.

1.11.4 Logon

The L command prompts for a user ID, and uses that ID for all subsequent file references.

1.11.5 Build Bootstrap Sectors

The B command builds the bootstrap sectors (cylinder 0, track 0, sectors 0 and 1). Sector 0 contains a hardware bootstrap for the ROOT, while sector 1 contains pointers to the MFD, the bitmap and the disk addresses and boot parameters for the separately compiled Executive tasks. Sector 1 is used by Root initialization to load the Executive tasks. The B command obtains the Sector 0 bootstrap from the file "HWBOOT.TSK" in the current user ID. The command prompts for a series of task ID's and boot parameters. For each task, the following boot parameters must be supplied:

Filename - The name of an existing file in the current UFD or ^ taskid, where taskid is a previously defined task. If ^ taskid is entered, the bootstrap entry is flagged so that the code for taskid will be shared with this task and not loaded again.

Debug Mode - If debug mode is Y, the initial flags of the task will offset so that the task will not automatically start when the system is booted.

If the bootstrap sector already contains boot information, the user is prompted for each existing entry to determine whether to change that entry. After all existing entries are processed, new entries may be added. Note that since the boot sectors contain absolute disk addresses rather than file names, it is necessary to rebuild the boot sectors whenever an Executive task or the Root is changed.

1.11.6 Set Date

The D command prints the current date (MM/DD/YY) from the calendar clock and accepts a new date. Typing an invalid date will hang the task and require a reload.

1.11.7 Set Time

The T command displays the current time from the calendar clock (HHMM:SS) and accepts a new time (HHMM). If an invalid time is is entered, the task will hang and the system must be reloaded.

1.11.8 Make Distribution Tape

The M command will write a magnetic tape consisting of the tape bootstrap contained in the file "MTBOOT.TSK", followed by a core image of the current incore system. This tape may be hardware booted. For the tape to be useful, the system must be inactive while the tape is being made, and the File Manager must be shut down.

1.11.9 Read Absolute Sector

The R command allows the operator to read disk sectors. The command prompts for the cylinder head, and sector to be read. After the sector is read, the command prompts for an output mode, starting data word and word count to print.

Legal modes are A (ASCII), D (Decimal), H (Hexadecimal) and O (Octal).

1.11.10 Set Indirect File

The @ command prompts for a file name and obtains subsequent commands from that file until EOF or error.

1.11.11 Shut Down

The Z command writes out the current bitmap in preparation for activation of the File Manager.

1.11.12 Disk Build Procedure

This section describes the sequence of operations required to build a disk from the distribution tape.

a) Boot the distribution system from tape;

b) Set the date and time;

c) Format the disk (if required);

d) Initialize the disk;

e) Define required UFD's, including 001001;

f) Using the Executive Debugger, set the flags of the task FM and MT to zero, allowing them to run;

g) Using the Tape Manager, copy the contents of UFD 001001 from the distribution tape;

h) Shut down the File Manager;

i) Logon as 001 001;

j) Use the Boot command to initialize the bootstrap sectors;

k) Shut down the Disk Builder;

l) Halt and reboot the system from disk.

2. RESIDENT OPERATING SYSTEM

The HEP computer contains four different types of memory: program, register, constant, and data. Programs executing on the machine are allocated a "task" in which to run. Each "task defines a contiguous region of each type of memory. The hardware restricts each user to his own region of memory, and restricts the type of access he may make to each memory type. Program memory is execute-only; constant memory is read-only; and register and data memory are read/write.

A task may contain one or several processes, which are executable code sequences. Several processes may be simultaneously executing in the HEP, unlike conventional computers. Processes are implemented by a set of hardware registers, of which there is a fixed number; thus an error condition (create fault) exists when too many processes come into existence in the processor. Since existing processes can create new processes at will, processes must be allocated to tasks and managed just as memory must be allocated and managed.

All of the sixteen hardware implemented tasks in the HEP are not equivalent. Tasks 0-7 are user tasks. In these tasks, privileged instructions are forbidden. In tasks 8-15, privileged instructions are allowed. These tasks, called "supervisors", perform system services for the user tasks. User tasks request these services with "supervisor call" (SVC) instructions. These instructions generate a "trap", creating a process in a supervisor task and suspending execution of the user. The hardware forces user traps to a particular supervisor task, for example, task 2 traps to task 10. In general, task k(k<8) traps to task k+8.

Supervisors may also generate traps. All traps from a supervisor create a process in task 8. A supervisor trap suspends the supervisor in the same way a user trap suspends the user. Note that a trap suspends ALL processes in a task, not just the process causing the trap.

The HEP computer is interfaced to I/O devices and the user via the Executive computer. In hardware, the Executive can read and write HEP program and data memory and certain control registers via a low speed (actually quite high speed) bus (LSB) interface. Privileged instructions on the HEP can also manipulate certain of these registers. In addition, the Executive and the Resident Operating System are interfaced via the passive interface, which allows supervisory processes to interact with the Executive. One of the main supervisory functions of the HEP O/S is to control the flow of data through these interfaces.

The Resident Operating System is organized into two main components: the Kernel and the Supervisors. The users (in tasks 1-7) make service requests (via SVC) of their corresponding Supervisors. In the event of user errors, the Supervisors contain error handling routines. The Supervisors run in tasks 9-15, and execute privileged instructions to carry out user requests. When a user request requires I/O with the Executive computer, the Supervisors communicate with the Executive File Manager via the passive interface. The Kernel handles error conditions arising in the Supervisor code, and handles the majority of operator interface functions. In addition, since the hardware traps all create fault conditions to task 8, the Kernel handles these also. Note that since the task using the last process (and getting create fault) may not be the one using too many, the Kernel must find the offender with software and take appropriate action. This is the reason that create faults come to the Kernel rather than the Supervisors. Supervisors have control ONLY over their associated user. All supervisor code is rentrant, so that only one copy is present in each PEM.

2.1 Kernel

The Kernel of the resident operating system occupies task 8 in every PEM. The base and limit registers of the Kernel are set to allow it to address all of memory. The Kernel is logically divided into three parts - the Inbound Kernel, which responds to directives from the Executive; the Outbound Kernel, which responds to traps from Supervisors; and the Create Fault Handler, which responds to create fault traps. Each of these Kernel sections is described below. Data structures used by the Kernel are described in Section 2.1.1, along with Kernel initialization.

2.1.1 Kernel Data Structures and Initialization

Each Kernel accesses two types of data structures: private data structure unique to each PEM, and shared data structure accessed by all PEM's in a system. The shared data structures are located at the base of data memory, and relate to the control and status of data memory. The private data structures follow the shared data structure. During initialization, each Kernel uses the RCLK instruction to obtain its own unique processor number. This number is used to offset its private data structure area into a unique region of data memory. In addition, each processor examines and updates a shared cell defining the base of user data memory. Each Kernel sets this cell to the end of its own private area, if the previous content of the cell is not already higher than that. Thus, after all processors are

initialized the cell contains the correct base of user data memory, regardless of the order in which the processors were initialized.

2.1.1.1 Memory Management Data Structures

Memory is managed using a partition table for each memory type. The length of each table is an assembly time parameter of the Kernel. The partition table for data memory is in the shared area, while the tables for register, constant and program memory are in the private area.

A partition table consists of a number of partition descriptors, followed by a terminator word. The format of the table is shown in Figure 2.1.

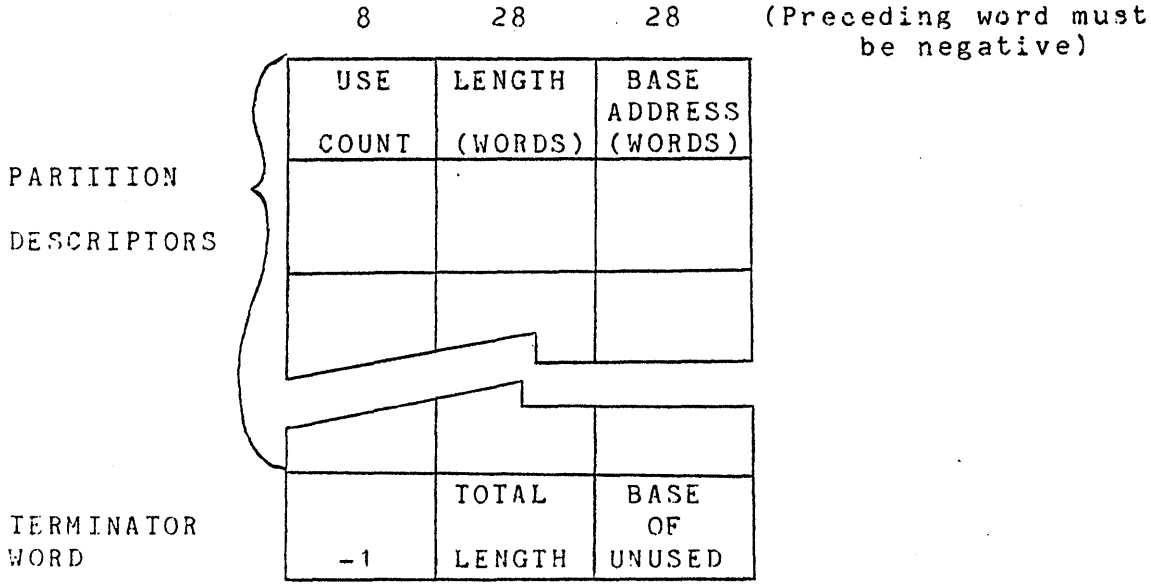


Figure 2.1 - PARTITION TABLE FORMAT

The base address or length of a partition descriptor may only be changed when its use count is zero.

2.1.1.2 Task Management Data Structures

Tasks are managed with a task table, process table and task status words. Since tasks are local to PEMs, these structures reside in private areas or in Kernel registers.

The task table is an array of seven task descriptors, each corresponding to a user/supervisor task pair. The format of a task descriptor is given in Figure 2.2.

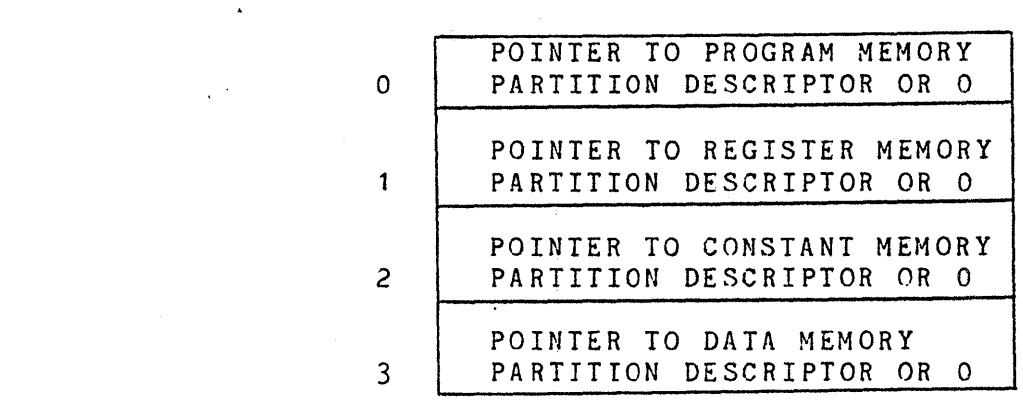


Figure 2.2 - TASK DESCRIPTOR

Each element of the task descriptor contains the byte address of the partition descriptor for the memory used by the task. The use count in the partition description is the number of task descriptors, system wide, which refer to the partition. For all but data memory, the use count cannot exceed 1.

The process table is an array of seven elements followed by a termination word. Each element contains the number of processes allocated by software to the corresponding task. The termination word contains the maximum total number of processes allowed by software.

Task status is contained in several registers. Each register uses one bit per task to record status. The three task states are VALID, ACTIVE and LOADED. A task becomes VALID when a program loading is begun in the task, and becomes not VALID when the program terminates normally or abnormally. A task becomes ACTIVE when it is

started (after loading) or resumed (after a pause). The task is not ACTIVE during program loading and after a pause. A task becomes LOADED upon completion of program load, and becomes not LOADED upon normal or abnormal termination. These states affect the type of directive which the Kernel will accept for tasks.

2.1.1.3 Communications Data Structure

Communications between the Executive and the Kernel is accomplished using a pair of message headers located in the Kernel private data structures. One message header is used exclusively by the Inbound Kernel for communication with the Executive task KI. The other is used exclusively by the Outbound Kernel for communication with the Executive task KO. A message header is a two-word block whose format is given in Figure 3.3.

8	8	8	8	8	8	8	8	WORD
PROC. NUMBER	SOURCE/ DEST. TASK	-	-	-	STATUS	MESSAGE DATA EXT.	MESSAGE TYPE	1
MESSAGE LENGTH (Words)				MESSAGE ADDRESS (Byte Address in HEP Data Memory)				2

Figure 3.3 - MESSAGE HEADER FORMAT

For the Inbound Kernel, message length and all the fields in word 0 are supplied by KI, after which the Kernel fills the address halfword with the address of the Kernel data buffer, located in the pivate data area. If the message is invalid, the status field is set to a non-zero value and the buffer address is not supplied. It is then the responsibility of the Executive to not transmit any data. The destination of all outbound messages is the Kernel (task 8).

For the Outbound Kernel, all fields are supplied by the Kernel. The data length and message address are obtained from the supervisor which trapped to request the message be sent. The message address is relocated by the Kernel to an absolute data memory address. Since no recovery is possible, the status field is not checked by

tne OUTBOUND Kernel. Tne source task field is tne task number of tne user task (1-7) wnose supervisor trapped.

2.1.1.4 Initialization

When tne Resident Operating System is first loaded, tne Executive causes an IPL trap wnicn begins execution of tne Kernel initialization code. Tnis code identifies tne processor number and sets up tne private data structures. Partition, task and process tables are initialized to tne inactive and unassigned states. All tasks are marked not VALID, not ACTIVE and not LOADED.

Initialization communicates witn tne Executive by sending tne base address of tne private data area, wnicn is also tne Executive communications area, to tne first Unibus-to-Switcn Interface location witn a STO instruction. Tnis address is received by tne Executive KI task and is maintained by tnat task for tne duration of system operation. After tne Executive nandsnake, tne initialization code brancnes to tne directive reception code of tne inbound Kernel.

2.1.2 Inbound Kernel

Tne Inbound Kernel processes directives received from tne Executive KI task. Tnese directives originate witn either tne Executive Batcn Monitor or tne HEP Debugger. Tne Inbound Kernel is a single process in task 8 wnicn normally waits in tne Unibus-to-Switcn interface for tne result of a memory read instruction. Wnen a directive is to be processed, tne Executive KI task provides eitner a "0" or a "1" as tne result of tne read. Tnis relinks tne Inbound Kernel in tne PEM process queue and execution begins (resumes).

Tne message type in tne inbound message neader is cnecked for reasonability and tne destination task is cnecked to verify tnat it is task 8. If tnese conditions are met, tne data value returned by tne Executive is used to enter pnase 1 or pnase 2 directive processing.

If tne data value was 0, pnase 1 processing is entered. During pnase 1, tne specific message code is validity cnecked, tne message lengtn is cnecked and message specific cnecking is performed on tne neader. If pnase 1 is successful tne message status is zeroed, tne Kernel buffer address is supplied and tne read to tne Unibus-to-Switcn interface is reissued.

When the read is again satisfied, the data value is again checked. If the value is not zero, phase 2 processing is entered. Phase 2 processing rechecks the message type and status. If satisfactory, the contents of the Kernel buffer are used for message specific processing, and may be altered to display results. After Phase 2 processing is complete, the read against the Unibus-to-Switch interface is reissued to await a new directive.

A total of 13 directives, some with varients, are accepted. These are briefly described below.

2.1.2.1 Examine Directive - Type 21 (16)

The Examine Directive causes the Kernel to read the contents of a single word of memory and place its value in the Kernel buffer. The message data extension in the header specifies the memory type, as shown in Table 2.4

VALUE		MEMORY TYPE
0	-	PROGRAM
1	-	REGISTER
2	-	CONSTANT
3	-	DATA

Table 2.4 - MEMORY TYPE CODES IN DIRECTIVES

The first word of the Kernel buffer contains the address to be examined. Upon completion, the next word will contain the register descriptor or 0, and the following word will contain the value.

2.1.2.2 Modify Directive - Type 1 (16)

The Modify Directive causes the Kernel to replace the contents of a single word of memory with a new value. The format of the directive is the same as the examine directive, except that all fields are supplied by the Executive.

2.1.2.3 Cancel Directive - Type 2 (16)

The Cancel Directive specifies a user task number (1-7) in the message data extension. This task must be VALID. The message causes the Kernel to create a process

in the task with an all-ones PC and UTM. When the task next becomes active, the process will trap to its supervisor, which will recognize the unusual PSW and terminate the task.

2.1.2.4 Suspend Directive - Type 3 (16)

The Suspend Directive specifies a user task number in the message data extension. The task must be VALID and ACTIVE. The message causes the Kernel to create a process in the task with a PC of 0 and an all-ones UTM. When this process traps to its supervisor, the supervisor will leave the task dormant and send a PAUSE message through the Kernel. This will mark the task not ACTIVE.

2.1.2.5 Resume Directive - Type 4 (16)

The Resume Directive specifies a user task number in the message data extension. If the data length of the message is non-zero, the message data is placed in the data memory of the task starting at user location 0. The task is then activated. Resume requires that the task be VALID and not ACTIVE. The task is set ACTIVE by Resume.

2.1.2.6 Load Directive - Type 5 (16) and Type 7 (16)

The Load Directive specifies a user task number in the message data extension. The task must be not VALID. The task descriptor for the task is checked to ensure that memory of all types is assigned. If these conditions are met, the TSW registers for the task are initialized using the partition descriptors. If the Load Directive is type 7 (system load), the task memory is cleared and initialized, and the contents of the load data buffer are copied into low task data memory. This data normally contains the file name of the system control card processor used to set up files for the user task.

For all load directives, runtime constants are initialized in the supervisor space of the user task, the task is marked VALID and not ACTIVE, PSW's having the tasks' task number are erased from the hardware PSW queue, and the loader process is created in the appropriate supervisor.

2.1.2.7 Miscellaneous Examine Directive - Type 22 (16)

The miscellaneous examine directive copies various information into the Kernel data buffer. The message data extension specifies what data to copy; as shown on the next page.

MDE VALUE	ACTION
0	Read PSW queue into buffer.
1	Read all TSW's into buffer.
2	Read CFU control into buffer.
3	Read ECC register and clock into buffer.
4	Read partition descriptors and process count for all tasks into buffer, in order by task, and memory type, 5 words per task.

2.1.2.8 Set Partitions Directive - Type 23 (16)

The Set Partitions Directive resets the partition descriptors in a partition table to reflect new partition sizes. The message data extension field contains the memory type of the table to be modified, as described in Table 2.4. The Kernel buffer is set by the Executive to contain an image of the partition table, in which only the length fields of the partition descriptors is significant. The directive builds an updated table in the Kernel buffer, filling in the correct base addresses and use counts. The updated table is then compared to the previous table to make sure that total memory is not exceeded, and that no descriptor with a non-zero use count has been changed. If these conditions are met, the updated table replaces the previous table.

At the conclusion of the directive, the then current partition table (previous or updated) is copied to the Kernel buffer for possible examination by the Executive.

2.1.2.9 Set Task Directive - Type 24 (16)

The Set Task Directive selects partitions descriptors from each memory type and places pointers to them in the task descriptor for the user task whose number is in the message data extension. The task must be not VALID. The first four words in the Kernel buffer specify the partition descriptor index (1 relative) in program, register, constant and data memory partition tables, respectively. If an index is zero, no memory is assigned of the corresponding type. If a task descriptor pointer was previously non-zero, the use count of the referenced partition descriptor is decremented before the pointer is moved. When the pointer is set to a new partition descriptor, the descriptor's use count is incremented.

If the reassignment was successful, the first word of the Kernel buffer is set to zero, otherwise it is set to -1.

2.1.2.10 Create Process Directive - Type 6 (16)

The Create Process Directive unconditionally creates the process whose PSW is in the first word of the Kernel buffer.

2.1.2.11 Dump Directive - Type 25 (16)

The Dump Directive is identical to the Examine Directive, except that the Kernel buffer is filled with 63 register descriptor/value pairs starting at the address contained in the first word of the Kernel buffer. The data begins at the second word of the buffer, and the last word of the buffer is not used.

2.1.2.12 Set Process Directive - Type 26 (16)

The Set Process Directive specifies the new contents of the process table. The new process counts are summed, and if the sum does not exceed the software process count limit, the process table is copied from the Kernel buffer. If the limit is exceeded, the first word of the Kernel buffer is set to -1 and the process table is not changed.

2.1.3 Outbound Kernel

Outbound Kernel processes are produced by error and SVC traps from supervisors. While multiple outbound processes may exist simultaneously only one such process per PEM makes progress at a given time, since all Outbound Kernel processes interlock on a single Kernel register. This requirement is imposed because Outbound Kernel code is not re-entrant.

2.1.3.1 SVC Processing

Normal entries into the Outbound Kernel are produced by SVC calls from supervisors. Four such SVC's are recognized as shown in Table 2.5.

SVC	NAME	MESSAGE CODE-HEX	ACTION
0	STOP	C2	Task set not VALID, not LOADED, not ACTIVE
1	PAUSE	C6	Task set not ACTIVE
2	CROAK	C3	Task set not VALID, not LOADED, not ACTIVE
3	LOAD COMPLETE	C2	Task set LOADED, not ACTIVE

Table 2.5 - Kernel SVC Codes

For all SVC entries, the data parameter descriptor; contained in the register 0 is relocated to an absolute memory address and is placed in the second word of the outbound message header. The first word of the outbound message header is set to the message type indicated in Table 2.5, and the source task is set to the user task number of the requesting task. The processor number is placed in the processor field.

The Executive is notified by storing the address of the Kernel private data structures into the second Unibus-to-Switch interface location. This produces an interrupt handled by the Executive KO task and results in the Baton Monitor reading the message header and data. After the Executive has read the data, the store is

responded to, and the Outbound Kernel resumes execution, unlocks the synchronization register and quits.

As part of all SVC preamble processing, the trapping process is vectored to a quit instruction and the task is reactivated. Thus the supervisor issuing the SVC continues with other processing in progress, but the process issuing SVC does not continue after the SVC.

2.1.3.2 Error Processing

Normal system operation does not generate error traps from the supervisors. If any occur, or if an illegal SVC is issued by a supervisor, it is a fatal system error. The trapping PSW, the Kernel PSW of the trap handling process, and the instruction causing the trap are captured and stored in a fixed 3-word buffer in the Kernel. A type C7 message is generated by the Kernel, pointing to the 3 word buffer. The trapping supervisor is left dormant. Occurence of this error will probably result in corruption of open disk files and should be followed by an immediate system shutdown and reload.

2.1.4 Create Fault Handler

If the total number of processes in all supervisor or all user tasks exceeds a hardware limit, a create fault trap occurs. All user or supervisor processes are placed in a quasi-dormant state. The create fault process determines if the fault was a supervisor or user overflow. If the fault was a supervisor overflow, the create fault condition is reset and operation continues. This should not occur, and is a potentially fatal system error. However, since supervisor processes cannot be abnormally terminated without causing system damage, there is no corrective action possible.

If the fault was a user overflow, the create fault handler examines the PSW queue to count the number of processes used by each user task. This count is compared to the limit specified in the process table. For all tasks whose actual process count exceeds the limit, all such processes are vectored to a quit. After these processes terminate, a process with a cancel PSW is created in the offending tasks. The create fault condition is cleared and normal processing resumes.

2.2 Loader

The program loader's function is to read a disk file containing the load module for an executable program and place the contents of the load module into program, register, constant and data memory. The loader runs as a supervisor task (9-15). As with all supervisor code, the loader's constant and program base are 0, allowing code and constants to be shared by all supervisors. Register and data base and limit restrict the loader to the memory allocated for the task to be loaded. The first 10 registers and 64 data memory words are reserved for the loader and I/O supervisor, as is the memory left above a loaded program. During Kernel initialization for a load, certain loader registers are loaded with control values for the load. In addition, certain low data memory locations within the supervisor task space are preinitialized.

2.2.1 Initialization

Upon startup, the loader determines if this is a "system load" or a "user load". For a system load, the loader takes a filename passed in the base of its data memory as the name of the file to load. It issues an open request to the Executive File Manager using the third Unibus-to-Switch location. This file is opened as logical unit 0, using standard I/O supervisor protocol. If the load is a "user load" logical unit 0 is presumed already open.

With logical unit 0 open, the loader begins to process load module records. This process continues until the end of file, the end of module record is encountered, or the relative task number field in the load module record descriptor changes. While processing load module records, several record types may be encountered. Processing of each type is described below.

2.2.2 Header and Checksum Records

Header records and checksum records are ignored.

2.2.3 Task Record

The task record contains the program size in program, register, constant and data memory. Only the register requirement field is used. This field, together with the process count for the task supplied by the Kernel, determine the number of user registers required. The fixed register

requirement in the task record determines the base of the register environment pool. This pool is set up during task record processing and is intended to allow 40 registers per process. The top of the environment pool becomes the user register limit, and the TSW is updated to reflect that fact. Remaining registers are formed into a pool of 4 register blocks for use in concurrent I/O processing. These register environments are also set up by task processing. The task record must precede the start record.

2.2.4 Start Record

The start record contains a PC value at which to begin execution. Start record processing combines this PC into a PSW with the register index of the first register environment. The resultant PSW is created, however since the user task is dormant, execution does not start until the Kernel subsequently activates the task.

2.2.5 Data Record

Data records consist of a memory type, word count, memory address and a block of data words to be loaded. All words in a block go into the same memory. The memory address in the data record is a user relative address, and must be relocated before loading. For program memory and constant memory, the loader base is zero, so the total offset of the base of the task must be added to the address. For register and data memory, the loader base is the bottom of the partition, so only the length of the supervisor reserved section at the partition base must be added. In all cases, the address to be loaded is checked against the limit for that memory. If the limit is exceeded the loader enters error termination processing.

2.2.6 Loader Termination

The loader terminates upon encountering an error, encountering a change of task or end of module record or upon reaching EOF.

If the loader encounters a change of task, the load file is left open and positioned to the first record of the new task. The loader terminates with a return code of 0 in the data block passed to the Kernel via SVC3. Key supervisor locks are reset to allow normal operation of the I/O supervisor.

In all other cases, the file is closed, and a non-zero return code is supplied. Table 2.6 describes these codes.

CODE	MEANING
X'10'	I/O error, reading past EOF or file not found.
X'20'	Unrecognized record type.
X'30'	End of module record reached.
X'40'	Program memory overflow.
X'41'	Register memory overflow.
X'42'	Constant memory overflow.
X'43'	Data memory overflow.

Table 2.6 - LOADER ERROR CODES

2.3 I/O Services

User I/O requests are handled by the corresponding supervisor process via SVC calls. When a user process executes an SVC instruction the task is made dormant, and a supervisor process is activated. In general, the supervisor will reactivate the user immediately after validating and copying the user's parameter block. This allows all of the processes in that task to proceed except for the process which actually issued the SVC instruction. When the supervisor has finished processing the SVC it will re-create the user process at the instruction location following the SVC, and it will proceed from that point. The exceptions to this are SVC 7 (Stop SVC) and 8 (Pause SVC), which are not reactivated. In the case of Stop the task will not be activated until after a KILL instruction has been issued by the Supervisor, and in the case of Pause, the activate will be issued by the Kernel when, and if the operator sends a Resume message from the system console.

2.3.1 SVC's

The services which the I/O Supervisor can perform for the user are the following:

SVC 0 - OPENLU - Allocate and open a disk file at a specified logical unit.

SVC 1 - CLOSELU - Close, rename or delete a disk file on a specified logical unit.

SVC 2 - BUFFERIN - Read a record.

SVC 3 - BUFFEROUT - Write a record.

SVC 4 - BACKSPACE - Backspace one record.

SVC 5 - REWIND - Reposition a disk file to the first record.

SVC 6 - ENDFILE - Write END OF FILE, close a file.

SVC 7 - STOP - Cancel the user task and print a message on the system console.

SVC 8 - PAUSE - Suspend the user task, and print a message on the system console.

SVC 9 - INQUIRE - Inquire regarding the OPEN/CLOSE status of an LU. If open, return the record length and options word.

*SVC 10 - GETENV - Acquire a supervisor LU buffer. *May be executed by CONTROL CARD PROCESSOR only.

SVC 11 - LOGON - Logon to FILE MANAGER using the user ID supplied.

SVC 12 - GETCORE - Returns the address of the first word above the user's Data Memory.

Refer to Figure A (next page) for Parameter Block Formats.

A - Requested Access Privilege
B - Public Access Privilege
C - Owners Access Privilege
D - History
E - Disposition
F - I/O Direction
G - Buffers

OPENLU SVC 0/CLOSELU SVC 1

STATUS	LUNO	RECLEN (WORDS)					
↑ VOLUME ID		↑ FILENAME					
X	G	F	E	D	C	B	A

BUFFERIN SVC 2/BUFFEROUT SVC 3

STATUS	LUNO	0
LENGTH (WORDS)		↑ BUFFER START
		0

BACKSPACE SVC 4/REWIND SVC 5/ENDFILE SVC 6

STATUS	LUNO	0
		0
		0

STOP SVC 7/PAUSE SVC 8

STATUS	0	0
TEXT LENGTH (WORDS)		↑ TEXT
		0

INQUIRE SVC 9

STATUS	LUNO	RECLEN (RETURNED)					
							0
X	(G)	(F)	(E)	(D)	(C)	(B)	(A)

RECLEN and A-G fields interpreted as with OPEN and CLOSE, returned if LU is OPEN.

GETENV SVC 10*

	0
	0
	0

*Executable only by Control Card Processor.

LOGON SVC 11

STATUS	
USER ID CODE (12 BYTES)	
	0

GETCORE SVC 12

	0
0	(USERDTOP RETURNED)
	0

Figure A - HEP SVC Parameter Block Formats

In the case of all SVC's , the user's indexed register number one (R1:I) is assumed to be a pointer to the parameter block. If R1 does not point to a valid user Data Memory address the result will be an abnormal termination (ABTERM) (see Section 2.4 - ERROR HANDLER for a description of ABTERM). Other errors associated with SVC's which will result in Abnormal Termination are issuing an illegal SVC number, and invalid text pointer in a STOP SVC. All other errors will be reflected in the status field of the SVC parameter block. Normal return status is zero, anything else indicates some abnormal condition.

Traps which result in the creation of a Supervisor process can be caused by a user process issuing a supervisor call (SVC) or by the detection of an error condition. These traps are received by the Kernel, which examines the PSW to determine which type of trap it is. It then creates a process in the appropriate supervisor task to process the trap. Error traps are processed by the User Error Handler, and SVC traps are processed by the I/O Supervisor.

When a supervisor process is created it will have control of a set of ten global registers which are shared by the Kernel and all supervisors in that task. In order to avoid conflicts only one supervisor at a time is allowed access to these registers. No other supervisor processes will be created until these registers have been released, and once they have been released a supervisor may not attempt to seize them again.

There also exists a similar job-wide set of common Data Memory which is similarly seized and freed by the supervisors in each task.

When the I/O supervisor is awakened it will already have control of the global registers. Its first act is to seize the global Data Memory for that job. It will not proceed until it has successfully acquired this Data Memory. Having acquired an operating environment it will then copy the SVC parameter block from user space and examine it. After verifying that the user has issued a valid SVC and the parameter block which is pointed to by the user's indexed register one (R1:I) is correct, the I/O supervisor will re-activate the user task, for all but STOP and PAUSE SVC's. This allows the user task to proceed with as short an interruption as possible. For those SVC's which require I/O Buffers the supervisor will then acquire one, and then will acquire a set of temporary local registers. Having acquired a

working environment where necessary the supervisor can now release the locks on the global Data Memory and registers and proceed with the processing of the SVC. From this point on the supervisor is completely re-entrant and, with the exception of multiple concurrent I/O to the same logical unit, can operate completely in parallel with any number of I/O supervisor processes, in any number of tasks. The local register environment consists of a set of four registers obtained from a pool. The Data Memory environment (LU environment) is 96 words in length. Figure B is a diagram of an LU environment.

OPENLU - SVC 0

OPEN SVC acquires an LU environment. If no buffer is available it will return an error status to the user. An LU environment is made available by the issuing of a GETENV SVC (SVC 10) by the Control Card Processor. Once an LU environment is seized its location is recorded in the LUTABLE, a job-wide table of open LU's. Entries in this table indicate that the LU in question is either 1) not open, 2) in use, or 3) available for use. The LUTABLE entry is marked in use. OPEN LU builds an open message from the information in the SVC parameter block, which it sends to the file manager. If the open is unsuccessful the LU environment is returned, the LUTABLE entry marked not open, and an error status is returned to the user. If the open succeeds, OPENLU will check the I/O direction field of the options word. If the direction specified is forward, the first record of the file will be read. If append is specified the last record will be read from the file and the position pointer set to the end of the last logical record. Even if a file is empty (i.e. contains no records as yet) there will always be one physical record in it, and the end of file pointer will point to the beginning of that record.

The contents of the options word will be saved in the LU environment to be referred to by successive I/O requests. The LUTABLE entry will be marked available, and the user process will be created at one past the PC of the SVC instruction. If at any time during the processing of this SVC an error condition is detected, the LU environment will be returned, the LUTABLE marked unopened, and an error status returned to the user. Figure C shows the format of the open message to the file manager.

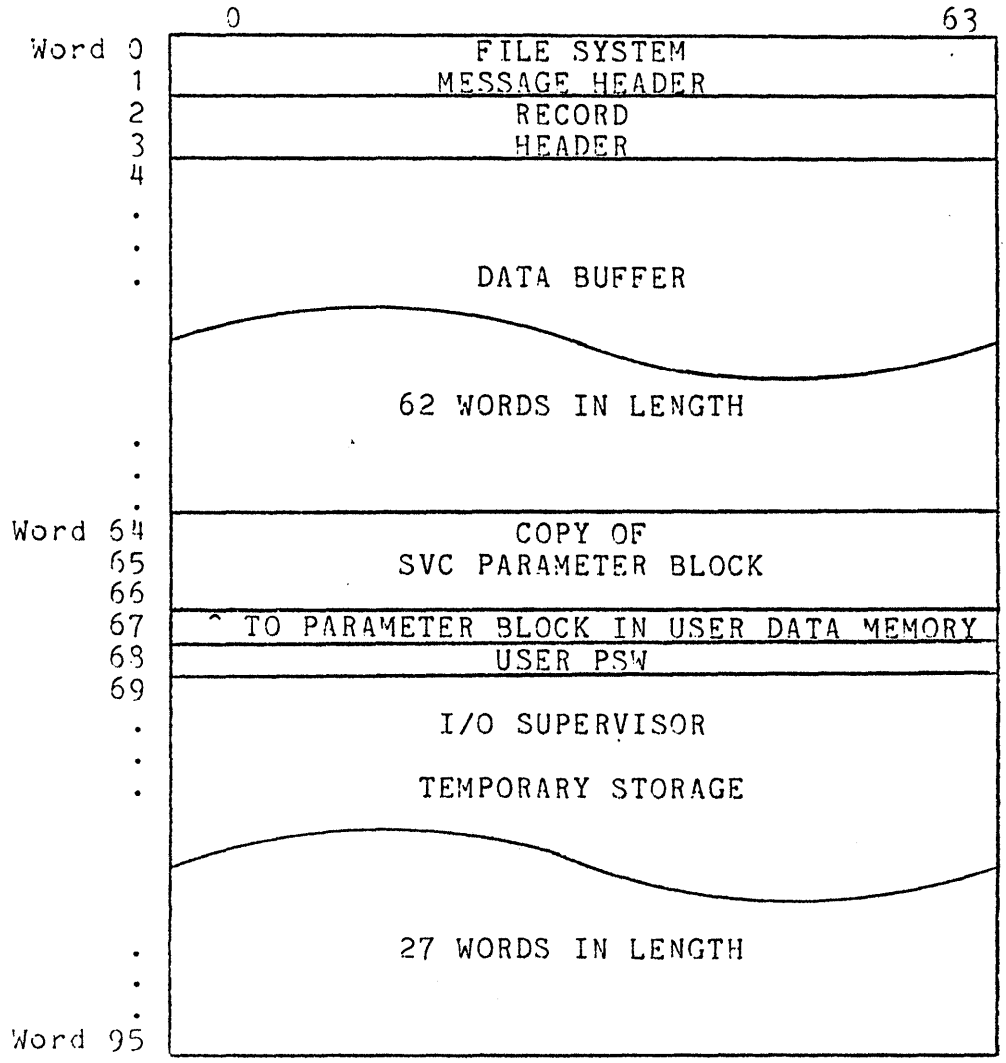


Figure B - HEP I/O SUPERVISOR LU ENVIRONMENT

FILE MANAGER MSGCODE = 0 - LOGON
1 - OPEN
2 - CLOSE
3 - READ
4 - WRITE
5 - OBTAIN

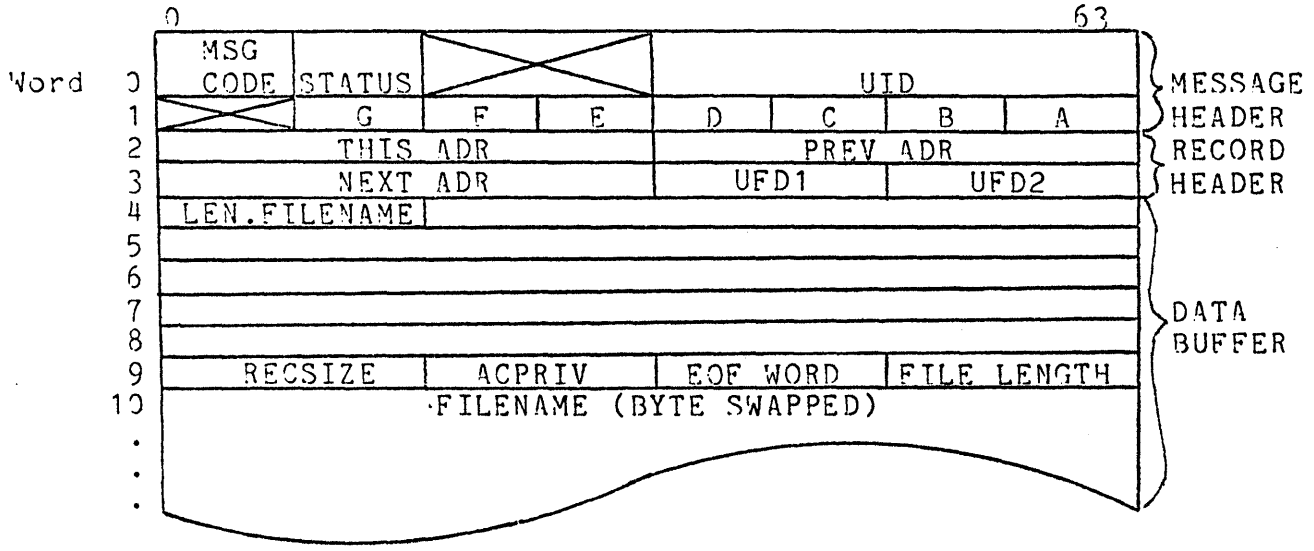


Figure C - OPEN/CLOSE FILE SYSTEM MESSAGE BLOCK

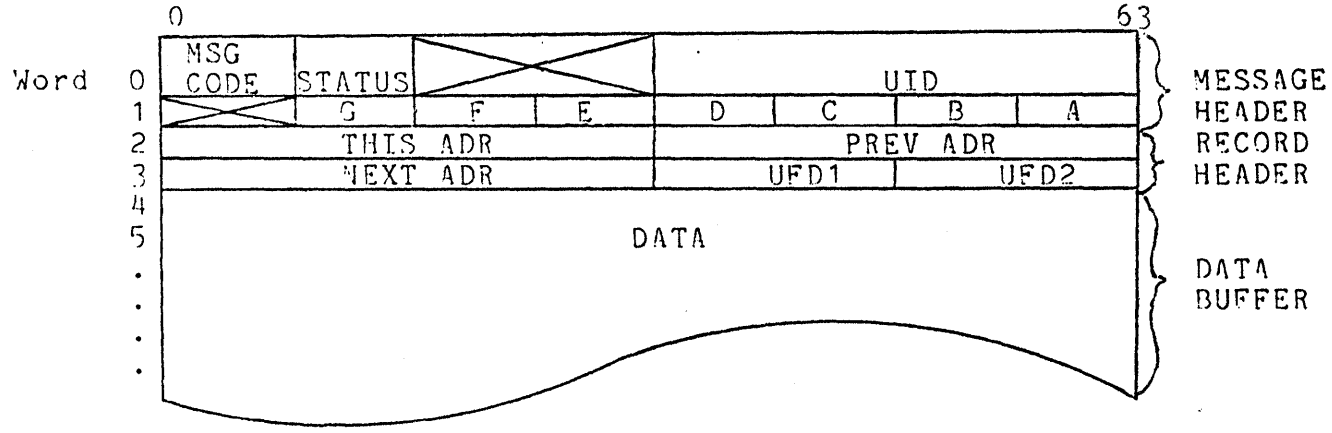


Figure D - READ/WRITE/OBTAIN FILE SYSTEM MESSAGE BLOCK

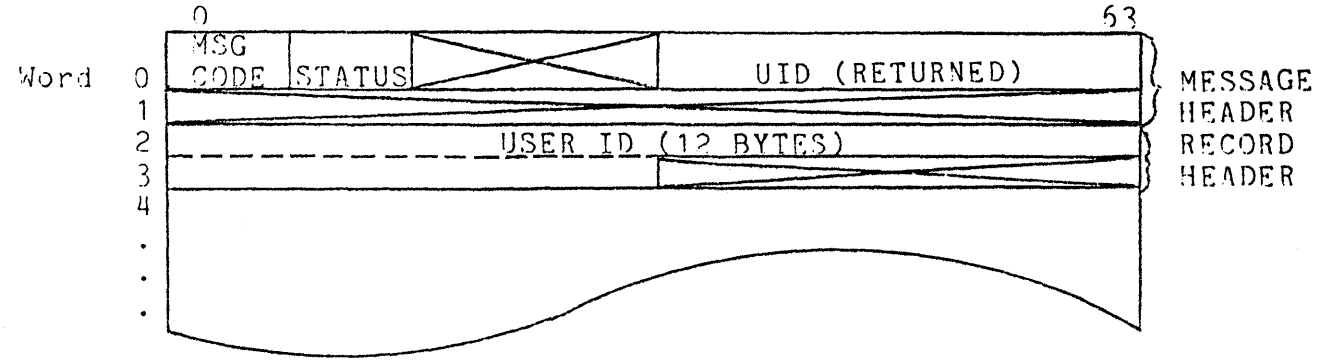


Figure E - LOGON FILE SYSTEM MESSAGE

CLOSELU - SVC 1

CLOSE SVC acquires an LU environment from the LUTABLE. If the entry in the LUTABLE indicates that the file is not open error status will be returned to the user. After it determines that the LU is open, CLOSE SVC will compare the options word which has been saved from the open with the options word sent in the CLOSE SVC parameter block. Changes indicated will be copied into the close message. If the file has been opened with write access the current record will be written and the EOF pointer will be updated. If the rename bit in the ACCPRIV field is on, and a file name is provided, this name will be copied (byte swapped for the PDP-11) into the message block. Finally a CLOSE message will be sent to the file manager. The LU environment is then returned and the LUTABLE entry marked not open. Status returned by the file system is returned in the SVC parameter block. The user process is re-created and the local registers are returned by the supervisor. Figure C shows the format of the close message to the file manager.

BUFFERIN - SVC 2/BUFFEROUT - SVC 3

BUFFERIN and BUFFEROUT perform the read and write operations respectively to disk files. Except for the direction of the I/O they are essentially identical. BUFFERIN/OUT acquires an LU environment from the LUTABLE. If the LUTABLE indicates the LU is not open an error status is returned to the user. EOF condition is checked for both operations, and if true, EOF status is returned, except where extend and write access has been granted on the open of the file. Data is copied to/from the LU Buffer and user Data Memory, one word at a time until the logical record length specified has been consumed. When a logical record crosses the boundary of a physical record a physical I/O is performed. In the case of BUFFERIN this is just a read. In the case of BUFFEROUT, if the current physical record is not the last record of the file the current record is written and the next record read into the buffer. If the current record is the last record in the file an OBTAIN message is sent to the file system processor to acquire the address of the next record, and have it assigned to this file. This address is copied into the next address field of the current record header, and the record is written.

Upon completion of the I/O the LUTABLE entry is marked available, status is placed in the parameter block, the user process is recreated, and the supervisor returns the local registers. Figure D shows the format of the Read, Write and Obtain messages to the file manager.

BACKSPACE - SVC 4/REWIND - SVC 5

BACKSPACE and REWIND each acquire an LU Buffer. As with BUFFERIN and BUFFEROUT, if the LUTABLE indicates the LU is not open error status is returned.

BACKSPACE/REWIND must check the current access privileges. If write access is included, the current record must be written, in case it has been modified. For BACKSPACE the current position pointer is decremented by the logical record length. If it is decremented beyond the beginning of the current logical record, as many reads in reverse direction as necessary are performed until the position pointer is in the current buffer.

For REWIND the operation consists of simply reading the first record of the file and setting the position pointer at the beginning.

Error status could be returned if an I/O error were to occur of if an attempt is made to BACKSPACE beyond the beginning of the file.

Status is returned in the SVC parameter block. The LU Buffer is returned, the LUTABLE entry marked available, the user process recreated and the supervisor local registers returned.

ENDFILE - SVC 6

ENDFILE, in this implementation, has the effect of a call to CLOSELU with the default options specified.

STOP - SVC 7

STOP does not acquire an LU Buffer, it does not obtain a local register environment and the user is not immediately re-activated. When the I/O supervisor is entered for a STOP SVC the user has already been de-activated as a result of issuing the SVC. The supervisor verifies the address of the message text in the parameter block. It issues a KILL followed by an

ACTivate instruction against the user task to make certain that the user task has no outstanding SFU requests. It then issues a call to CLOSEALL, which closes all open LU's. Finally, the supervisor issues a STOP SVC to the Kernel with a pointer to the message passed by the User. The STOP SVC to the Kernel is not the same as a user STOP SVC, a supervisor STOP request is SVC 0.

PAUSE - SVC 8

As with STOP, PAUSE does not acquire an LU Buffer, nor a local register environment, and the user is not re-activated. The user is already dormant, therefore PAUSE simply verifies the address of the text passed by the user in the parameter block, and issues a PAUSE SVC to the Kernel with a pointer to the same text string. A supervisor PAUSE request is SVC 1.

INQUIRE - SVC 9

INQUIRE acquires an LU Buffer. If it is found to be a new buffer, i.e. the LU in question is not open, the supervisor returns a non-zero status. If the buffer is not a new buffer then the LU has already been opened, and the supervisor returns zero status, and the record length and options word in the parameter block. The purpose of INQUIRE is to allow a user to ask the supervisor if an LU is open before attempting to do I/O, open or close it. The LU could have been opened by the Control Card Processor, or by another task within its own job (in a multi-PEM environment).

GETENV - SVC 10

A GETENVIRONMENT call is executable only by the Control Card Processor. For any other user an illegal SVC ABTERM will result. GET ENV does not require a local register environment, and does not obtain an LU Buffer. The supervisor simply decrements the user's Data Memory limit by the length of an LU environment and rewrites its TSW.

LOGON - SVC 11

LOGON acquires an LUBUFFER and a set of local registers. Even though it is not associated with an LU, LOGON requires a Data Memory work area. In order to maintain the re-entrant nature of the I/O supervisor it

must acquire this environment for the LUTABLE. For this reason there must be at least one LU environment available at the time a LOGON SVC is issued. The Control Card Processor issues a GETENV SVC (SVC 10) before it issues a LOGON. If a user finds it necessary to LOGON using a user ID other than the one used to open the jobfile, he must ensure that a buffer is available.

LOGON copies the twelve character user ID specified in the SVC parameter block, and issues a LOGON message to the file manager. If the file manager accepts this ID it will return the UIC code. This code will be used for all successive opens from this task until a new user ID is supplied. Files already opened under another UIC will remain under that UIC. If the LOGON is rejected by the file manager the error status will be copied into the SVC parameter block. The LU Buffer and local register environment are returned, and the user re-created. Figure E shows the format for a LOGON message to the file manager.

GETCORE - SVC 12

GETCORE does not require a local register environment or LU Buffer. It does not return status. If the pointer to the SVC parameter block contained in the user's indexed register one (R1:I) is not a valid address the supervisor will issue an ABTERM. Otherwise, the address of one greater than the last word of user Data Memory will be returned in the SVC parameter block.

2.4 Error Handler

Hardware-detected error conditions result in traps to a set of low-core addresses. All such errors in user tasks are processed by the User Error Handler, which is a Supervisor process running in the corresponding task (User Task Number +8). When the hardware detects an error condition the task is made dormant. The first act of the supervisor is to issue a KILL instruction followed by an ACTivate on the user task. This will insure that all processes in the task are terminated. Then the supervisor will call an I/O Supervisor CLOSEALL to insure that all opened files are closed. The Supervisor then builds an abnormal termination (ABTERM) message which consists of three words containing:

a) Trap PSW;

b) User PSW at time of error;

c) Instruction generating the trap.

This message is sent to the Kernel, and is normally printed on the System Console by the Batch Monitor.

When a Cancel Task message is sent from the Host, the Kernel creates a process in the user task with all bits on except the PS field. It is then treated as any other ABTERM.

When the Kernel receives a Suspend Task message from the Host, it creates a process in the user task at location zero, with all UTM bits on. This is a special case in which the Supervisor simply vectors the PSW which trapped to the quit at instruction zero, and issues a Pause request message with no text to the Kernel.

The User Error Handler shares the global supervisor registers and data memory with the I/O Supervisor. It must therefore observe the same semaphoring conventions on those resources in order to avoid conflicts.

3. SYSTEM SOFTWARE

3.1 Control Card Processor Overview

HEP Control Card Processor (CCP) is a system program which processes certain records in the user job file. CCP is responsible for allocating and opening all disk files for the user Partition as a result of submitting a job. After processing the records in the Job File, CCP terminates, leaving all files open, with the user load file assigned to LU zero.

If CCP encounters an error in the Job File, such as an illegal command or not being able to open a file with the requested privileges, all open files will be closed, and the Job will be terminated.

All CCP commands must begin in column one and unless otherwise stated, must be followed by at least one blank. Commands recognized by CCP are:

JOB - Job Specification Record

ASN - File Allocation and Assignment Record

DMP - Conditional Dump Record

RUN - Run Record - Specifies Load File and and End of Job Step

// - End of Job Record - Terminates All Job Steps

'*' - Comment - Any Record Beginning With an '*' is Treated as a Comment

3.1.1 Control Card Processor Command Syntax

The following commands are accepted by CCP. Even though other commands may be recognized by the READER, by the time CCP receives access to the Job File they should be commented out.

3.1.1.1 Job Record Syntax

JOB<Jobname Followed by Job Environment Requirements>

This record is copied into Logfile with no further processing required by CCP. The first record in a Jobfile must be a Job Record.

3.1.1.2 Assign Command Syntax

ASN LU,FILENAME[,Logical Reclen, Accpriv, Owners Accpriv, Public Accpriv, I/O Direction, File History, File Disposition, Buffer Count]

[] - Indicates optional parameters, not order dependent.

Example: ASN 5,CARDFILE,REC:80,ER,F,OLD, KEEP:DELETE

Accpriv - A single letter for each access privilege, may be specified in any order.

Prefix O - Indicates Owners Privileges

Prefix P - Indicates Public Privilges

No Prefix - Privileges For This Open

R - Read Access

W - Write Access

X - Extend Access

E - Exclusive Access

S - Semaphored Access - (Implemented in HSFS Only)

D - Delete/Rename Access

If file is being newly created, O and P Accprivs become the permanent attributes of the file.

If file already exists, O and P Accprivs are ignored.

DEFAULT - (No Accpriv Specified)
For this open - R - read access
For owner if creating file - WXED
Write, Extend, Exclusive, Delete/Rename
For public if creating file - No privileges

I/O Direction - A Single Letter

F - Forward - Open at beginning of file, do I/O in forward direction

B - Backward - Open at end of file, do I/O in forward direction (implemented in HSFS only).

A - Append - Open at end of file, do I/O in forward direction (X - Extend Access must be allowed for appending).

DEFAULT is F - Forward I/O.

File History -

USE - Use old file if present, else create new file.

NEW - Create new file, delete old file if present.

OLD - Use old file, fail if not present.

CREATE - Create new file, fail if old file present.

DEFAULT - (No history specified at all) is USE.

File Disposition -

DELETE - Delete on any close.

KEEP - Keep on any close.

DELETE:KEEP - Delete on normal (user) close - Retain on abnormal (system) close.

KEEP:DELETE - Keep on normal (user) close - Delete on abnormal (system) close

DEFAULT - (No disposition specified) is: If old file used, KEEP on any close - If newly created file, DELETE on any close.

Logical Reclen -

REC:n or n

Where n = Desired logical record length in HEP words.

If file is being created, n becomes the default RECLEN for the file and if n = 0, or is not specified, a word file is assumed.

If file already exists -
If n = 0 it is treated as a word file for this open.
If n is not specified the default RECLEN for the file is used.

Buffer Count -

Number of buffers allocated in HEP Data Memory for this open of this file (implemented in HSFS only).

BUF:m

Where m - Number of Buffers Desired.

DEFAULT is 2 Buffers.

(NOTE: In standard file system implementations Buffer Count defaults to one, even if BUF is specified in a Control Card.)

3.1.1.3 Conditional Dump Command Syntax

DMP[ALWAYS][(MEMORY TYPES)]

ALWAYS - Specifies dump after any termination. Default is dump after abnormal termination.

(MEMORY TYPES) - P - Program
R - Register
C - Constant
D - Data
Memory types to be dumped, enclosed in parenthesis.
Default is (RD) Register and Data Memory dump.
Memory type characters may be in any order, and should not be separated by any other characters.

Example:

No DMP Record - No dump will be taken.

DMP ALWAYS (P) - After any termination, Program Memory is dumped.

DMP (CRD) - After abterm, Constant, Register and Data Memory are dumped.

DMP - After abterm, Register and Data Memory are dumped.

3.1.1.4 Run Command Syntax

RUN<LOAD FILENAME>[OPTIONAL RUN PARAMETERS]

<LOAD FILENAME> - Name of user task to be run.

[OPTIONAL RUN PARAMETERS] - Copied as ASCII text, left justified into user Data Memory, beginning at word 0 relative to user Data Memory base.

If optional run parameters are included, the first ten words of user Data Memory will be initialized to zero. Then the text string will be copied, beginning at byte 0, and running to the end of the input line.

Example:

RUN MYFILE.TSK - Load and run MYFILE.TSK, no run parameters.

RUN HEPTASK A B C D E F - Copy the string 'A B C D E F' into user Data Memory beginning at word zero and blank filled to the the end of the input line. Load and run HEPTASK.

3.1.1.5 End of Job Record Syntax

// -or- End of File on Jobfile.

Causes termination of a HEP job. Any open files are closed.

3.1.1.6 Comment Record

Any string beginning with an asterisk ('*') in column one.

The HEP Control Card Processor (CCP) is implemented in such a way as to make it very easy for a user to add features to it, or if desired, write a new one. The following describes the Runtime Environment expected by the CCP.

3.1.2 Runtime Environment

CCP runs in the User Partition just prior to the execution of a User Task. In the case of a multi-task job, CCP is loaded in the first partition large enough to hold it.

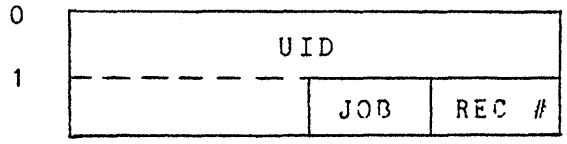
CCP Run Parameters

In User Data Memory beginning with word 0 is the following information:

User ID Code - 12 Bytes, ASCII

Job Number - 2 Bytes, Binary

Jobfile Record Number - 2 Bytes, Binary



REC #<0 indicates a dump has been taken.

The following filenaming conventions apply to CCP:

Jnnnn.HEP = Jobfile Name

Lnnnn.HEP = Logfile Name

Dnnnn.HEP = Dumpfile Name

Pnnnn.HEP = Dump Formatter Print Filename

Where 'nnnn' = Job Number

All of the above filenames can be fabricated using the Job Number passed to CCP as a run parameter.

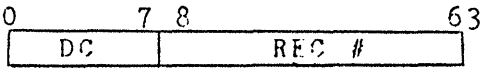
Jobfile Record Number (word 1, 4th quarter) is the number of the last record of the Jobfile read by CCP for the current job.

IF REC #=0 then this is the first step of this job. Begin processing from the first record of Jobfile.

IF REC #>0 then this is not the first step of this job. Begin processing from REC # + 1 of Jobfile.

IF REC #<0 i.e. the high bit of this field is set, then this is not the first step of this job, and furthermore, a dump has been taken of the HEP memory.
CCP must Open the Dumpfile at LU1, Open the Dump print file at LU2, and Open the Dump Formatter Load Module (DMFMT.HLL) at LU0.

When CCP terminates normally, it passed a binary stop code via SVC 7 to the Baton Monitor, of the form;



WHERE DC = Dump Code - Applies only to this Job Step.

Dump Code Conditions - Bits, 2, 3

No Dump	0 0	0 (16)
Dump on Abterm	0 1	1 (16)
Dump on Normal Term	1 0	2 (16)
Dump Always	1 1	3 (16)

Memory Type to be Dumped - Bits 4,5,6,7

Program Memory	1 0 0 0	8 16
Register Memory	0 1 0 0	4 16
Constant Memory	0 0 1 0	2 16
Data Memory	0 0 0 1	1 16

Memory Dumped will be the inclusive Or of these bits, e.g.:

X'1F' - Dump all memory on abterm.

X'35' - Dump Register and Data Memory on any termination.

X'00' - No dump.

REC # = Number of jobfile record last processed. If end of file on the jobfile has been encountered the number returned is a zero.

In processing LU assignment records the CCP must perform the following sequence of Supervisor Calls:

1) LOGON SVC 11 - Log on using the user ID in Data Memory bytes 0 - 11.
This must be done once at the beginning of execution, before any files, including the Jobfile are opened.

2) INQUIRE SVC 9 - Inquire regarding the status of an LU. If LU is already opened, SVC 9 will return its current default record length, and options. If an LU is open an attempt to open it again, whether for the same file or not, will result in an error.

3) GETENV SVC 10 - Get an LU buffer environment. This must be done prior to opening a file. SVC 10 is allowed only to CCP. Once gotten, an LU buffer will not be returned. If files are opened, and subsequently closed before another is opened, an extra SVC 10 is not required. The number of SVC 10's issued must be greater than or equal to the number of LU's open at any time. SVC 10 will not return a status, if it fails, the next open will be unsuccessful.

4) OPENLU SVC 0 - Open a file at the specified logical unit.

If the program to be run is a FORTRAN program, or an Assembly Language program which uses the FORTRAN I/O Formatter, it is necessary that the OPEN and any Input or Output be done via calls to the I/O Formatter, as it maintains internal buffers for the opened Logical Units. Refer to HEPFMT documentation for a complete description of I/O Formatter routines.

3.2 HEP Dump Formatter

The HEP Dump Formatter is a system program which runs in a user partition immediately following the execution of a job for which a memory dump is taken. The purpose of the Dump Formatter is to translate the binary dump file into a printable and more legible format. The Dump Formatter is loaded by the Control Card Processor at the instruction of the Batch Monitor, immediately after the dump is taken. In the case of a multi-step job, if a dump is taken it will be formatted before the next step in the job is executed.

A typical dump will contain the UIC (User ID Code), job number, jobfile record number, job name, processor and task numbers for each task in the user job, user and supervisor TSW's for each task, system table entries, user and supervisor PSW's and the contents of the entire partition for each memory type specified.

A user may request that a dump be taken either after an Abnormal Termination (ABTERM), or after any termination of his job. He may also specify which memory types are to be dumped. This is accomplished by the DMP command in the jobfile. Depending on the information in the DMP command, the Batch Monitor initiates a dump upon receiving a job complete message from the Kernel. This binary dump will be output to a newly created file with a name of 'Dnnnn.HEP', where 'nnnn' is the job number. This is a record file with a logical record length of 129 words. It will contain one header record followed by one task record and one PSW record for each task. Then will come the memory dump records for the memory types specified, in the following order: Data Memory, then Register Memory, Constant Memory and Program Memory for the first task, followed by the Register, Constant and Program Memory for the second task, and so on. Figure A contains a diagram of the dump file record formats.

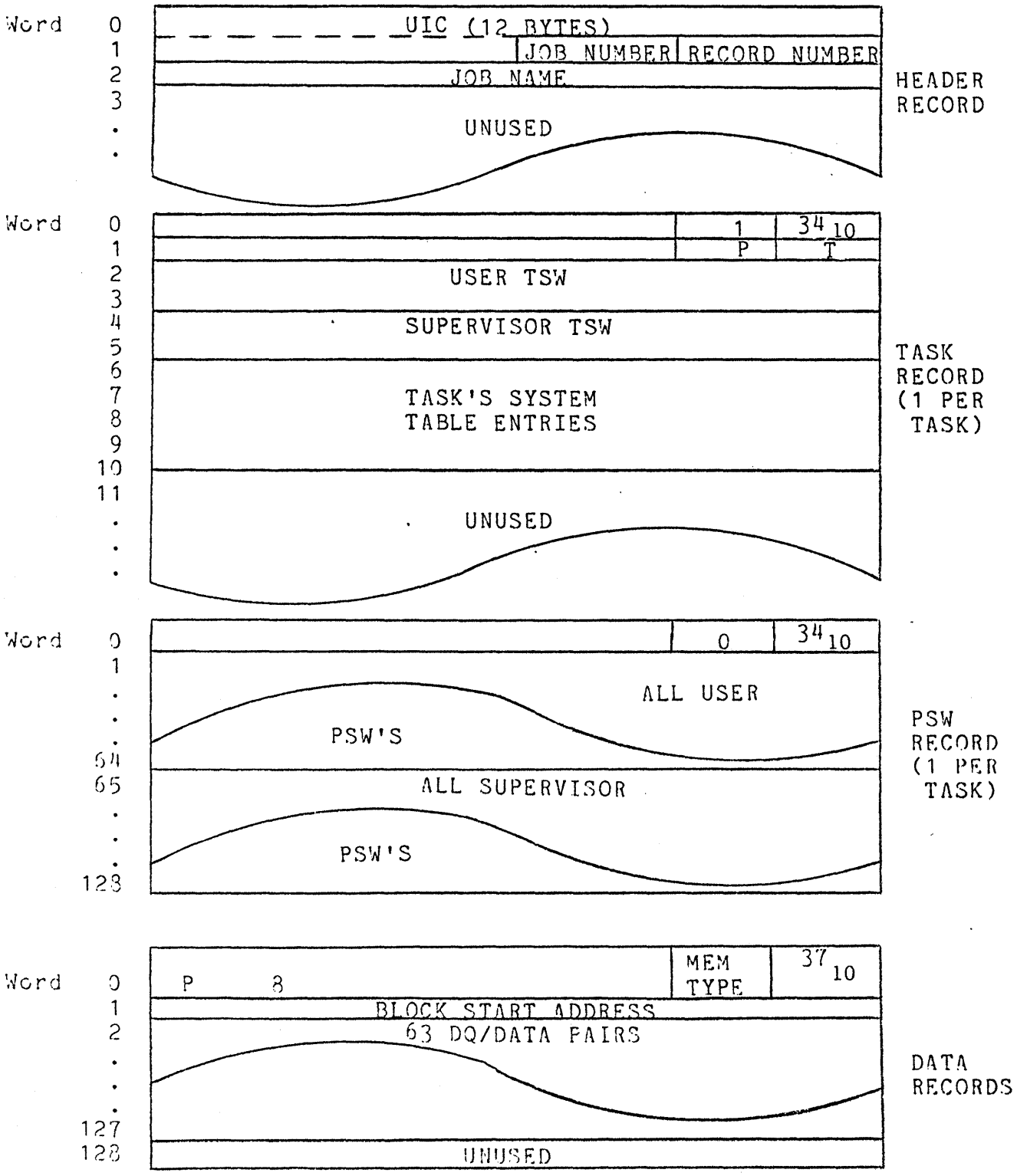


Figure A - DUMPFILE RECORD FORMATS
Logical Record Length = 129 Words

Figure A - DUMPFILE RECORD FORMATS
Logical Record Length = 129 Words

The following describes the processing necessary for each record type.

Header Record - Contains UIC, job number, job file record number and job name.

Task Records - (1 per task) - Contain processor number, task number, user TSW, supervisor TSW and system table entries. The system table entries give the starting address and length of the Data, Register, Constant and Program Memory partitions, and the maximum number of processes allowed for the task. The user TSW contains the base and limit addresses for the user's memory partitions. Using this information a table of supervisor base, user base, user limit and supervisor limit is set up for each memory type. This table will be referred to in processing the Data Records.

PSW Records - (1 per task) - Contain all of the PSW's in the PEM at the time of the dump. The first 64 are user PSW's; the last 64 are supervisors. The Dump Formatter scans through this record comparing the PT field with the PT specified in the Task Record. Those PSW's which belong to the task in question are output to the printfile.

Data Records - Using the information in the first two words, the absolute address of each word in the record is calculated. This address is compared with the base and limit Table entries for the appropriate memory type to determine whether it is a supervisor or user, and within the partition. If the end of the memory partition falls within a record buffer, the buffer is filled with as many words as necessary beyond the end of the partition.

In a Data Record the word immediately proceeding a memory is its register descriptor. The following processing is necessary for each memory type: (see Figure B)

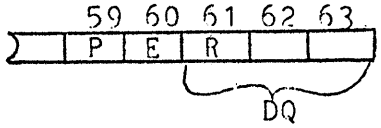
Register : Empty/full bit and reserved bits are checked, and 'E'/'F' and 'R' are printed,. The number representing data quality is printed, and parity is calculated. If the parity bit in the DQ is not correct an '*' is printed.

Data : Empty/full bit is checked, and 'E'/'F' is printed. The data quality number is printed and parity is checked. If parity is incorrect an '*' is printed.

Constant : Parity is calculated. If the parity bit is incorrect an '*' is printed.

Program : The Register Descriptor field is ignored for Program Memory.

Finally, the address, memory type, supervisor/user's status, register descriptor field and memory contents are printed, four words to a line. If the supervisor/user's status or memory type is different from the previous word, the current line is terminated, and several lines skipped to dilineate the change. If more than two lines in a row would be identical except for the address, the message '**** THROUGH ****' is printed, and all but the first and last lines are left unprinted.



P = Parity Bit

E = Empty Bit = 1 - Empty, = 0 - Full

R = Reserved Bit = 1 - Reserved

DQ = Bits 61=63 Represent Data Quality

Register Descriptor Word Format
Figure B

3.4.2 HEP I/O Formatter

The HEP I/O Formatter is a set of system subroutines which is included in the user Load Module. It provides an interface between the FORTRAN program and the I/O Supervisor. The I/O Formatter is responsible for performing formatted and unformatted I/O, opening and closing logical units, backspace, rewind and endfile, and issuing STOP and PAUSE requests. It is primarily record and logical unit oriented. If a user wishes to write an Assembly Language program which utilizes the I/O Formatter, it must be done in the same manner as a FORTRAN program. The following describes the interface between the user program (FORTRAN or Assembler) and the I/O Formatter.

OPEN and CLOSE are called using the standard HEP FORTRAN calling sequence, with parameters as follows:

```
CALL OPEN (LU #, Filename, Logical Record Length,
           Options Word)

CALL CLOSE (LU #, filename, Logical Record Length,
            Options Word)
```

Parameter blocks for all other I/O Formatter routines have special formats (See Figure B). In all cases, as with any FORTRAN subroutine, indexed register zero (R0:I) contains a pointer to the user's Data Memory Base, indexed register one (R1:I), a pointer to the parameter block, and the low 32 bits of indexed register two (R2:I) contain the return PC. The word immediately following the parameter block must contain a -1. This is beeause OPEN and CLOSE may be called with a variable number of parameters, and the end of a parameter block is designated by a -1.

F%READ/F%WRITE

↑ FORMAT	LU NUMBER
END=RETURN PC	ERR=RETURN PC
0	

F%IOLST

0	LU NUMBER
LENGTH -1(ARRAY ELEMENTS)	↑ I/O ITEM
0	

F%STOPI

0	LU NUMBER
0	
0	

F%BSPAC/F%RWIND/F%WEOF

0	LU NUMBER
0	
0	

F%STOP/F%PAUSE

TEXT LENGTH (HEP WORDS)	0
TEXT	

F%BUFIN/F%BUFOU

0	LU NUMBER
END=RETURN PC	ERR=RETURN PC
↑ ARRAY	LENGTH (HEP WORDS)

Upon CALL

R0 = ↑User's Data Memory Base

R1 = ↑Parameter Block

R2 = Return PSW

Figure B
HEP FORTRAN - I/O Formatter Interface
Parameter Block Formats
Revision 3/30/81

I/O Formatter Entry Points
(Refer to Figure B for parameter block formats.)

OPEN/CLOSE

OPEN acquires an LU Buffer, and issues an SVC 0, to open the file specified, at the LU specified. If a record length is not specified (i.e. the third parameter negative) the default record length for the file is used. If record length is zero the file is treated as a word file. Any other record length supplied is considered the record length for this open. WARNING: Attempts at formatted I/O on a word file may have unpredictable side effects. The I/O Formatter is record oriented.

CLOSE frees the LU Buffer, and issues an SVC 1 to close the file specified. If a filename parameter is specified it will attempt to rename the file, and if record length or options word parameters are included, these will also be copied into the SVC parameter block. Close may be called with only an LU number parameter if desired.

A user may open and/or allocate a file by name, and close or delete or rename a file using the OPEN and CLOSE subroutines in the I/O Formatter. These activities are accomplished by means of the Options Word parameter. This word is copied directly into the SVC parameter block by the subroutine. It is divided into several one byte fields which have the following meanings:

A - Requested Access Privileges For This Open

B - Public Access Privileges

C - Owners Access Privileges

D - History

E - Disposition

F - I/O Direction

G - Buffer Count

Access Privileges - Fields A, B, and C

If the file is being created (open) the privileges requested in these fields become part of the permanent attributes of the file. On a close, if the high bit of each field is set, these become the new attributes of the file.

Bit Definitions

.......1 Read Access.

......1. Write Access - Update Records.

.....1.. Extend Access - Add Records.

....1... Exclusive Access - No Other Concurrent Opens Allowed.

* ...1.... Semaphored Access - May Consume and Fill Records.

..1..... Rename/Delete Access.

.1..... Undefined.

1..... Change Privileges.

* Semaphored access is unimplemented in the standard file system.

For field A, current access privileges, the default privilege is read access only.

For field B, public access, the default is no public access.

For field C, owner's access, the default is read, write, extend, exclusive and rename (00101111) access.

File History - Field D

Determine whether to use an old file or create a new file.

Values:

0-Use old file if present, else create new file - this is the default.

1-Create new file, delete old file if present.

2-Use old file, fail if not present.

3-Create new file, fail if old file is present.

File Disposition - Field E

Specifies the disposition of the file upon close. Entries in this field on open are kept until the close. If no entries are specified on close those specified with the open will be used.

0-Keep old file, delete new file - default.

1-Delete file on close.

2-Retain file on close.

*3-Retain file on system close (i.e. ABTERM), delete on user close (normal termination).

*4-Retain on user close, delete on system close.

**5-Retain and rename file.

* These have meaning only on open.
** Valid for close only.

In the case of a file opened several times the last disposition specified (open or close) in chronological order determines the file disposition which will be used.

I/O Direction - Field F

Controls the initial positioning of the file, and the direction for sequential access. This field is ignored by close.

0-Forward - Open file positioned at the beginning of the first record, do I/O in forward direction - default.

1-Backward - Open file positioned at the beginning of the last logical record, do I/O in reverse direction - this is unimplemented in the standard file system.

2-Append - Open file positioned at the end of the last logical record, do I/O in forward direction - (appending to a file requires extend privilege).

Buffer Count - Field G

Number of physical records to be held in I/O Cache at any time. this feature is unimplemented in the standard file system. All entries in this field will be ignored by the I/O supervisor.

R%READ/F%WRITE

READ and WRITE seize the LU Buffer for the LU specified, and prepare it for I/O, marking it busy, and setting up the ERR= and END= return addresses, and the format pointer, if they are specified.

F%IOLST

IOLISTITEM performs whatever processing is necessary for the I/O Item (Scalar or Array) specified. When the buffer is full (in the case of WRITE) or empty (in the case of READ) it issues the appropriate SVC.

F%STOPI

STOPIO finishes processing of the format statement if necessary, and marks the LU Buffer available for another I/O.

F%BUFIN/F%BUFOU

BUFFERIN and BUFFEROUT combine the actions of READ/WRITE, IOLIST and STOPIO for unformatted I/O to (from) a single IO item (Scalar or Array). They acquire an LU Buffer, marking it busy, issue the SVC's required for the I/O requested, and upon completion return the LU Buffer and mark it available.

F%STOP/F%PAUSE

STOP and PAUSE issue the appropriate SVC (7 for Stop, 8 for Pause) with a pointer to the text supplied in the parameter block.

F%BSPAC/F%RWIND/F%WEOF

BACKSPACE, REWIND and ENDFILE each acquire the LU Buffer specified, and issue the appropriate SVC (4, 5, or 6 respectively).

A typical call sequence for a formatted Read or Write would consist of 1) a call to F%READ (/F%WRITE), followed by 2) one or more calls to F%IOLST, one call for each I/O list item, and terminated with 3) a call to F%STOPI (see Example 1). An unformatted I/O operation could be accomplished in the same manner, omitting the format pointer in the parameter block for the F%READ/F%WRITE call. A more efficient method however is to issue a single call to F%BUFIN or F%BUFOU (see Example 2). These result in a single Call/Return sequence, instead of a minimum of three which would be required if I/O is done as with formatted I/O.

Example 1: The FORTRAN Statements

```
DIMENSION A(2), B(10)

READ (6, 1000, END=2000, ERR=3000) A,B,C
```

would generate the following series of subroutine calls:

1) A call to F%READ with a parameter block of

↑ LABEL 1000	6
↑ LABEL 2000	↑ LABEL 3000
0	0

2) A call to F%IOLST for the Array A.

0	6
1	↑ A
0	0

A second call to F%IOLST for the Array B.

0	6
2	↑ B
0	0

A final call to F%IOLST for the Scalar C.

0	6
0	↑ C
0	0

3) A call to F%STOPI to finish the READ and mark the LU available.

0	6
0	0
0	0

Example 2: The FORTRAN Statements

```
DIMENSION A(6)

BUFFERIN (6, END=2000, ERR=3000) A
```

would generate a subroutine call to F%BUFIN with a parameter block of:

0	6
LABEL 2000	LABEL 3000
A	6