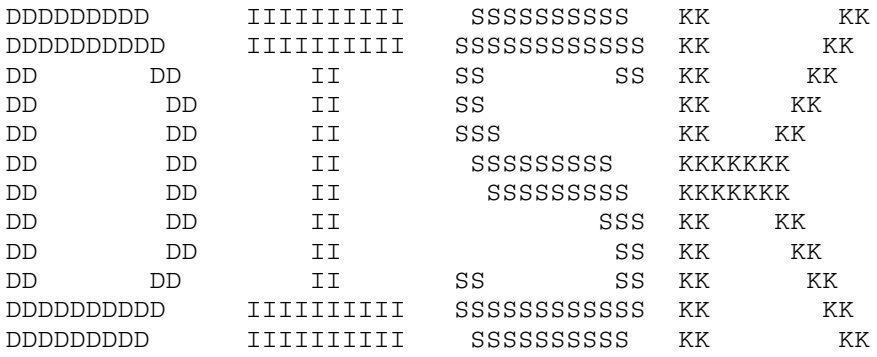
UM D6.0



April 1987

(I n t e r n a l D o c u m e n t a t i o n)

Table of Contents

Disk General Reference Data

The MTS file system works with two general types of disks: count-key-data format and fixed-block format. The two formats differ in the details of how each file system record is accessed, but the size of each record is the same for both types--4096 bytes. A 4096 byte record, also known as a page, is the unit of data which is read or written by the file system.

Independent of the type of disk which is used, the file system considers each disk to be a linear vector of 4096 byte records. Every record is accessed by the file system by specifying a device where the record is and a relative record number (starting with zero) on the device. (This abstraction comes from the TSS operating system, and is called the VAM2 format. It has been augmented by MTS and is known as VAMX in this form.)

Every file system disk adheres to the following conventions:

- there is a structure called the PAT (Page Assignment Table) which describes how every page on the disk is being used. The beginning of the PAT is pointed to by the volume label. The PAT is a linear vector of bytes, with each byte in one-to-one correspondence with a page on the disk. Hence, the PAT is as large as the number of pages on the disk. Since it must be written in 4096 byte chunks just like any other file system data, the PAT size is always an integral multiple of 4096 bytes. The remaining space at the end of the PAT in the last page is used for relocation entries (see below).

  Every page fits into one of the following five categories:
  - -data page
  - -free page
  - -bad page or special page
  - -PAT page
  - -DSCB page

  The PAT entry codes used in MTS are:
  - -X'00' means free page
  - -X'01' means data page
  - -X'80' means DSCB page (<12 slots used)
  - -X'82' means DSCB page (12-15 slots used)
  - -X'83' means DSCB page (all slots used)
  - -X'C0' means bad page, or special page
  - -X'7F' means PAT page
  - -X'FF' means end of PAT entries

  The last page in either a single- or double-density pack contains the regular PAT data plus relocation entries. Relocation entries describe pages which have been found bad during previous disk operations and whose data has been moved to another page on the pack. The bad pages are marked as error pages, and relocation

entries are placed at the end of the PAT describing which pages were bad and where they have been reassigned to. Whenever a disk page is read or written, the relocation entries are checked to see if the page is in the list of relocated pages. If so, the relocated page replaces the originally assigned page in the channel program constructed to effect the disk operation.

The format of relocation entries follows. If there are no relocation entries the last word of the PAT will be zero. If there are relocation entries the last word of the PAT will appear as:

-bits 0-15 contain number of relocation entries
-bits 16-31 contain X'FFFF'.

The relocation entries themselves are eight bytes in length and start at the penultimate fullword in the last PAT page and work backwards. The format of a relocation entry is:

-bytes 0-3 contain number of error page
-bytes 4-7 contain number of new page

- the first few pages (generally the first track's worth of pages but at least three) on every pack contain IPL data and the volume label (this is an IBM convention). They are marked in the PAT as bad pages. The last 3 pages of every pack are always marked as bad also (again an IBM convention).

- the format of the data area of the volume label for VAMX is as follows:

CL4'VOL1'	ID
CL6	VOLUME LABEL (E.G: MTS001)
XL1'F0'	SECURITY CODE
FL2	PAT RELATIVE PAGE ADDRESS (NOT ALIGNED)
CL4'3330'	DEVICE TYPE[1]
XL1'20'	VOLUME STATUS INDICATOR[2]
H	PVN (PUBLIC VOLUME NUMBER)
XL1'00'	
10C' '	MANUFACTURER NAME
10C' '	ASA NUMBER
10C	OWNER ID
CL2'VX'	VOLUME FORMAT (VAMX)
CL9' '	NOT USED, BLANK
XL1'00'	VTOC RELIABILITY INDICATOR - IF LOW ORDER BIT IS 1, SPACE ALLOCATION INFORMATION ON THE PACK IS INCONSISTENT
XL4	POINTER TO CATALOG (IF PVN=1)
FL4	MAXIMUM RELATIVE PAGE NUMBER ON THE PACK
CL9' '	NOT USED, BLANK

---

[1]Can be any of: 2311, 2314, 3330, 333B, 3340, 3344, 3350, 3370, 3375, 3380, 3390, or 6280.

[2]Can be any of X'00' (private), X'20' (public), X'40' (paging).

The volume label is record 3 on cylinder 0, head 0, of every count-

key-data type pack (this track is not page formatted), and has a key value of C'VOL1'. On a fixed-block pack the label is at block 1.

- the IPL information on a disk is located in records 1, 2, and 4 of track zero. The disk hardware implements a READ IPL command which causes track zero record one to be read. Record 1 is a 24 byte record initially written by the MTS DASDI program. It is a no-op record and causes the machine to go into wait state if IPLed. IPLINIT will replace this with something more reasonable. It changes it to a record for bootstrapping record 2. Record 2 is a standard record for bootstrapping record 4, which is the machine code image for the IPLBOOT program. (Continuing the IPL, IPLBOOT then reads in IPLREAD, which is located in a sequential file called *IPLAREA but which is not in the catalog. *IPLAREA is described by DSCBs, however.) Records 1 and 2, like the volume label, are unusual in that they have keys of IPL1 and IPL2, respectively.

- DSCB (Data Set Control Block) pages contain 16 DSCB slots -- DSCBs are always 256 bytes long. DSCBs tell where files are on the disk. They contain a file name and a list of where all the pages comprising the file reside. DSCBs come in two flavors: type E and type F. Each file has exactly one type-E DSCB associated with it. If the file is greater than 38 pages then it will also have one or more type-F DSCBs associated with it (the type-Fs are chained from the type-E). The format of the DSCB's is documented elsewhere (see DSCBDSCT macro).

- the hardware characteristics for each type device differ, but their general character is summarized here.

  Disk packs look like a stack of phonograph records, and are read by a comb-like access mechanism which sticks between the platters. Each tooth of the comb has one or more read/write heads on it dedicated to accessing a given platter's surface. The platters are thin aluminium substrate with magnetic oxide on both sides. Data is stored on each surface as a series of magnetized bits in concentric rings starting at the outside edge and going inward towards the spindle. Each ring is known as a cylinder, and each platter in a ring is known as a track.

  Depending on the type of disks, this stack of platters may or may not be removable. The different types have different numbers of platters. Data is not necessarily written on every platter. For example, removable disks (2311s, 2314s, and 3330s) which are not shrouded (like 3340s) do not use either the top nor the bottom surface to record data (they are protective). Other surfaces may be dedicated to servo information, used by the access mechanism to position itself properly in order to read or write data. Newer model disks have this servo information magically encoded within the data and do not require a dedicated servo surface.

  The hardware only knows how to access data based on cylinder and track. Fixed-block disks are smarter in that they need only be told

a relative block number on the device and do the conversion to cylinders and tracks themselves.

Since the rotation speed of the disk determines how fast data can be transferred to or from the device, the rotational rate and data recording density determine the data transfer rate. These vary depending on the disk type and manufacturer (all the following are IBM advertised values):

type	rev. per minute	latency (msec)	transfer rate (Mb/sec)
2311	2400	25	0.156
2314	2400	25	0.312
3330	3600	16.7	0.806
3340	3000	20.24	0.885
3350	3600	16.7	1.198
3370	3000	20.2	1.859
3375	3000	20.2	1.859
3380	3600	16.7	3.0
3390	4250	7.1	4.2
6280	4000	15.2	1.859

- disk capacity table:

type	cylinders	tracks	pages /track	capacity (pages)	default pat start	default pat len
2311	203	10	<1	1624	800	1
2314	203	20	2+	6496	3200	2
	406			12992	6400	4
3330	411	19	3	23427	11400	6
	815			46455	22800	12
3340	349	12	2	8376	4200	3
	698	12	2	16752	8400	5
3344	698	12	2	16752	8400	5
3350	560	30	5+	74480	37240	19
3370	959	12	7+	69750	4?	18
3375	959	12	8	92064	46032	23
3380	886	15	10	132900	66400	33
	1771	15	10	265650	132835	65
	2656	15	10	398400	199200	98
3390	1113	15	12	200340	100170	49
	2226	15	12	400680	200340	98
6280	840	20	6	100800	50400	25

Note that the capacity figures only represent the maximum number of pages which could be fit onto a disk pack. The actual values are lower due to the information on track zero which is not page

formatted, and the pages which are occupied by the PAT.

- A summary of disk format on the various disk types follow.

2311: Each group is five tracks, comprising eight records and defining four pages.

record 1 track 0 is a bytes of page 1
record 1 track 1 is 4096-a bytes page 1
record 2 track 1 is b bytes of page 2
record 1 track 2 is 4096-b bytes of page 2
record 2 track 2 is c bytes of page 3
record 1 track 3 is 4096-c bytes of page 3
record 2 track 3 is d bytes of page 4
record 1 track 4 is 4096-d bytes of page 4.

2314: Each group is five tracks, comprising twelve records and defining eight pages.

record 1 track 0 is page 1
record 2 track 0 is 2920 bytes of page 2
record 1 track 1 is 1176 bytes of page 2
record 2 track 1 is page 3
record 3 track 1 is 1592 bytes of page 4
record 1 track 2 is 2504 bytes of page 4
record 2 track 2 is page 5
record 3 track 2 is 207 bytes of page 6
record 1 track 3 is 3889 bytes of page 6
record 2 track 3 is 3136 bytes of page 7
record 1 track 4 is 960 bytes of page 7
record 2 track 4 is page 8.

3330: Each group is one track, comprising five records and defining three pages.

record 1 is page 1
record 2 is a 102 byte gap
record 3 is page 2
record 4 is a 102 byte gap
record 5 is page 3

3340/3344: Each group is one track, comprising two records and defining two pages.

record 1 is page 1
record 2 is page 2

3350: Each group is eleven tracks, comprising fifty-nine records and defining nineteen pages.

record 1 track 0 is page 1
record 2 track 0 is page 2
record 3 track 0 is page 3

```
record 4 track 0 is page 4
record 5 track 0 is 1945 bytes of page 5
record 1 track 1 is 2151 bytes of page 5
record 2 track 1 is page 6
record 3 track 1 is page 7
record 4 track 1 is page 8
record 5 track 1 is 3890 bytes of page 9
record 1 track 2 is 206 bytes of page 9
record 2 track 2 page 10
record 3 track 2 is page 11
record 4 track 2 is page 12
record 5 track 2 is page 13
record 6 track 2 is 1554 bytes of page 14
record 1 track 3 is 2542 bytes of page 14
record 2 track 3 is page 15
record 3 track 3 is page 16
record 4 track 3 is page 17
record 5 track 3 is 3499 bytes of page 18
record 1 track 4 is 597 bytes of page 18
record 2 track 4 page 19
record 3 track 4 is page 20
record 4 track 4 is page 21
record 5 track 4 is page 22
record 6 track 4 is 1163 bytes of page 23
record 1 track 5 is 2933 bytes of page 23
record 2 track 5 is page 24
record 3 track 5 is page 25
record 4 track 5 is page 26
record 5 track 5 is 3108 bytes of page 27
record 1 track 6 is 988 bytes of page 27
record 2 track 6 page 28
record 3 track 6 is page 29
record 4 track 6 is page 30
record 5 track 6 is page 31
record 6 track 6 is 772 bytes of page 32
record 1 track 7 is 3324 bytes of page 32
record 2 track 7 is page 33
record 3 track 7 is page 34
record 4 track 7 is page 35
record 5 track 7 is 2717 bytes of page 36
record 1 track 8 is 1379 bytes of page 36
record 2 track 8 page 37
record 3 track 8 is page 38
record 4 track 8 is page 39
record 5 track 8 is page 40
record 6 track 8 is 381 bytes of page 41
record 1 track 9 is 3715 bytes of page 41
record 2 track 9 is page 42
record 3 track 9 is page 43
record 4 track 9 is page 44
record 5 track 9 is 2326 bytes of page 45
record 1 track 10 is 1770 bytes of page 45
record 2 track 10 page 46
```

record 3 track 10 is page 47
record 4 track 10 is page 48
record 5 track 10 is page 49.

3375: Each group is one track, comprising eight records and defining eight pages.

record 1 is page 1
record 2 is page 2
record 3 is page 3
record 4 is page 4
record 5 is page 5
record 6 is page 6
record 7 is page 7
record 8 is page 8.

3380: Each group is one track, comprising ten records and defining ten pages.

record 1 is page 1
record 2 is page 2
record 3 is page 3
record 4 is page 4
record 5 is page 5
record 6 is page 6
record 7 is page 7
record 8 is page 8
record 9 is page 9
record 10 is page 10.

6280: Each group is one track, comprising six records and defining six pages.

record 1 is page 1
record 2 is page 2
record 3 is page 3
record 4 is page 4
record 5 is page 5
record 6 is page 6.

3370: There are eight blocks per page. Though the disk is not count-key-data, the position of each record on the disk is determined for slot sorting to optimize access order. There are 62 blocks on each track. Each group is four tracks, comprising 248 blocks and 31 pages.

blocks 0-7 track 0 are page 1
blocks 8-15 track 0 are page 2
blocks 16-23 track 0 are page 3
blocks 24-31 track 0 are page 4
blocks 32-39 track 0 are page 5
blocks 40-47 track 0 are page 6
blocks 48-45 track 0 are page 7

blocks 56-61 track 0 are 3072 bytes of page 8
blocks 0-1 2 track 1 are 1024 bytes of page 8
blocks 2-9 track 1 are page 9
blocks 10-17 track 1 are page 10
blocks 18-25 track 1 are page 11
blocks 26-33 track 1 are page 12
blocks 34-41 track 1 are page 13
blocks 42-49 track 1 are page 14
blocks 50-57 track 1 are page 15
blocks 58-61 track 1 are 2048 bytes of page 16
blocks 0-3 track 2 are 2048 bytes of page 16
blocks 4-11 track 2 are page 17
blocks 12-19 track 2 are page 18
blocks 20-27 track 2 are page 19
blocks 28-35 track 2 are page 20
blocks 36-43 track 2 are page 21
blocks 44-51 track 2 are page 22
blocks 52-59 track 2 are page 23
blocks 60-61 track 2 are 1024 bytes of page 24
blocks 0-5 track 3 are 3072 bytes of page 24
blocks 6-13 track 3 are page 25
blocks 14-21 track 3 are page 26
blocks 22-29 track 3 are page 27
blocks 30-37 track 3 are page 28
blocks 38-45 track 3 are page 29
blocks 46-53 track 3 are page 30
block 54-61 track 3 are page 31.

3390: Each group is one track, comprising 12 records and defining 12 pages.

record 1 is page 1
record 2 is page 2
record 3 is page 3
record 4 is page 4
record 5 is page 5
record 6 is page 6
record 7 is page 7
record 8 is page 8
record 9 is page 9
record 10 is page 10
record 11 is page 11
record 12 is page 12.

Console Error Messages

Whenever a fatal error occurs, the 3330-compatible unit check routine (UCDISK) produces a message on the operator's console. Disk error messages have the following format:

```
CL4           DEVICE NAME (E.G.: D261)
C' '
CL6           VOLUME LABEL (E.G.: MTS004)
C' ERROR '
C             ERROR CODE (SEE NEXT PAGE)
C' '
CL8           FLAGS
C' '
CL2           GLOBAL SENSE FLAG (SHOULD BE X'01')
C' '
CL53          SENSE DATA (24 BYTES)
C' '
CL8           CCCCHHHH (CYLINDER AND HEAD)
C' '
CL2           FAILING CCW OPERATION CODE
```

See an operator manual for an interpretation of the sense data.

The 'flags' word is the input and output flag bits to UCDISK - of interest only to programmers using the unit check routines.

If an abbreviated message appears on the console (just consisting of the device name and the error code), this indicates that CMDSTAT drop areas are full. Cmdstat may not be running, or it may be running but unable to reach the the disk. To start the CMDSTAT task (if it has stopped running) issue an MTS *CMD command at the console.

Unit Check Error Message Codes

? - invalid parameters[1]

0 - insufficient sense data (less than the full 24 bytes)

1 - all relevant sense bits are zero

2 - equipment check

3 - command reject[1]

4 - file mask violation (unexpected)[1]

9 - bus-out check

A - uncorrectable data check

B - overrun - 10 tries

C - unwanted track overflow[1] (CKD) or block size exception[1] (FBA)

D - unexpected end of cylinder[1]

E - intervention required (2301 or 3805)

F - overflow incomplete

G - unable to find define extent parameters (FBA)

H - check data error (FBA)

I - incorrect length

J - invalid device type[1]

K - no record found[1]

L - channel detected error: program check, protection check, channel protect check, interface control check, or offline control unit

M - low spares on 3805 or 3825

P - track condition check (3340 or 3344)

Q - intent violation (3380)

R - seek check (3340 or 3344)

S - PCI fetch mode and correctable data check[1] - everything

except 2301s

T - intervention required

U - channel data check

V - chaining check

W - environmental data presented

X - truncation error during data check correction

Y - program interrupt looking at user's CCW list[1]

Z - incremented seek address past a cylinder boundary and the file mask forbids seeks[1]

- regular recording of buffered log of activity on this disk

[1]Probably a software problem, not flakey hardware.

Structure of the Catalog

The CATALOG in MTS is composed of a number of FILES (special, to be sure) named *MASTER.CATALOGn where n=0, 1, 2, ...255. These special files are also called EXTENTS. These extents may all be on the same disk VOLUME or they may be scattered across different disk volumes. For reasons of efficiency, they should be scattered across volumes (and even control units). Each *MASTER.CATALOGn is linked to the next *MASTER.CATALOGn+1. The above structure is generally determined at the time the catalog is initially built, and except for facilities provided to dynamically expand when necessary,[1] this structure does not change. Thus, it is important if one is building the catalog to know ahead of time on what volume(s) one wants the catalog to reside and to "direct" the building process in that direction.

In general, each catalog file (extent) has the following structure. The first PAGE[2] or RECORD of each extent is the EXTENT HEADER. This header contains a 4 byte id ("*EH*"), a 4 byte count of the number of pages in this extent (maximum of 816), a 4 byte link to the next extent,[3] and a 4 byte data set control block (DSCB) type E address[4] for this *MASTER.CATALOGn.

The remainder of the extent header is composed of a variable number of FREE SEGMENT DESCRIPTORS. The number of free segment descriptors is equal to the number of pages in the extent. The free segment descriptor indicates which segments (as defined below) in the corresponding page are available for use. A free segment descriptor is composed of a 4 byte page address followed by a 1 byte bit map describing the free segments.

---

[1] The expansion is open-ended in the sense that the catalog will always create a new *MASTER.CATALOGn+1 (given available space, of course).

[2] The catalog uses 4096 byte page size physical records as does the regular file system.

[3] Catalog addresses take the form of a 12 bit public volume number and a 20 bit relative page number (starting at zero) within the volume.

[4] The DSCB type E is not used by the catalog routines.

The remaining pages in the extent have the following structure. Each page has a 16 byte PAGE HEADER, also called a RECORD HEADER, containing a 4 byte id ("*RH*"), the 4 byte address of the extent header, a 4 byte relative page number within this extent (starting at 1 since page 0 is used for the extent header), and the 4 byte address of this page. The remainder of the page is broken up into 6 SEGMENTS, each 680 bytes long.

Each segment has a 20 byte SEGMENT HEADER containing the following: the 4 byte userid to whom the segment has been assigned, a 4 byte link to the next segment assigned to this userid,[5] a 1 byte count of the maximum number of DESCRIPTORS (see below) that can be contained in this segment, a 1 byte descriptor length, and 10 unused bytes.

A segment can be assigned to the MASTER INDEX, SYSTEM FILE CATALOG, SCRATCH FILE CATALOG, or a USER CATALOG. If a segment is assigned to the master index, it may contain a maximum of 55 MASTER INDEX DESCRIPTORS each 12 bytes long. If a segment is assigned to the system file catalog, the scratch file catalog, or a user catalog, it may contain a maximum of 10 FILE DESCRIPTORS and/or SHARING DESCRIPTORS each 66 bytes long.

The master index contains a descriptor for every userid that has permanent private files in the system. This master index descriptor contains a 1 byte flag, a 4 byte userid, a 4 byte address of the first segment of the user catalog for this userid, and 3 unused bytes. The master index is generally searched only once per userid per session at the first reference to a userids private files to obtain the address of the userids catalog. Thereafter, MTS remembers where the userids catalog is to speed up subsequent references.

When a user creates his first private file, an entry is made in the master index and the user is assigned an available segment.[6] Furthermore, as is the case every time a file is created, a file descriptor is placed in the users catalog.

---

[5] The segment number (0-5) within the page is indicated in the high order 4 bits of the address (as with DSCB addresses). Segments of a user catalog need not be on the same volume.

[6] Segment 5 of each page is not assigned to new users for their first segment. This segment is reserved as an overflow segment for existing catalogs.

The FILE DESCRIPTOR contains a 1 byte flag (to distinguish it from sharing descriptors), 1 byte of owner access, 1 byte of global access,[7] 1 byte of flags indicating a) the version of the file system catalog, b) if the file should be saved by FILESAVE, c) if it is a privileged file, and d) the type of file, the 16 byte filename, the 4 byte address of the DSCB type E for the file, a 4 byte owner ID, an 8 byte STCK value of the last time the file was closed, a 2 byte creation date, a 2 byte last reference date, a 4 byte usage count, one byte of flags indicating if the file is to have program product charging applied, a 2 byte last change date, and a 12 byte program key.

In addition, if the file has been permitted (via $PERMIT) to specific userids, projects, or program keys, the file descriptor will have a 6 byte sharing descriptor linked to it.[8]

The SHARING DESCRIPTOR is composed of a 1 byte flag, a variable number of variable length SHARING DESCRIPTOR ENTRIES and, if necessary, the 6 byte link[8] to the next sharing descriptor. Each sharing descriptor entry is composed of the one byte IBM length of the userid, project number,or program key, a 1 byte flag indicating first whether the entry is a userid, project number, program key, qualified userid-program key or a qualified project number-program key and second, what type of access is allowed this userid, project number,or program key,[9] and the actual userid, project number, or program key.

The algorithm for determining access is (generally) as follows. The sharing descriptors are scanned checking for a "match" against the userid, project number, and program key in question. Since it is possible to "match" the same userid (or project number or program key) against more than one sharing descriptor entry, (by permitting access to subsets of userids, e.g., all userids whose first n characters are ...) the access of the most specific match is the one allowed.

Furthermore, userid access has higher priority than project number access and project number access has higher priority than (unqualified) program key access so that if a userid and a project number both "match", the userid access is used, regardless of the number of characters matched. (Access permitted to a program key "qualified" by a userid has higher priority than access permitted to just the userid, and access permitted to a program key "qualified" by a project number has higher priority than access permitted to just the

---

[7]Access allowed to others.

[8]This 6 byte link has the form, a 12 bit public volume number and segment number, a 20 bit relative page number, and a 2 byte offset into the segment. Thus descriptors need not be on the same volume.

[9]The types of access currently allowed are no access, read access, write expand, write change & empty, renumber & truncate, rename & destroy, permit, or any combination thereof.

project number). Finally, if no specific access applies, then the global access is used.

When a user overflows his first segment with more than 10 file and/or sharing descriptors, a new segment is allocated and the first segment is linked to it. An attempt is made to allocate the next segment on the same page as the previous segment (i.e., segment 5). In any event, a new segment will be allocated even if it becomes necessary to dynamically expand the catalog in the process.

As would be expected, whenever a user destroys a file, the corresponding file descriptor is removed from the user catalog (as well as any sharing descriptors attached to the file descriptor). Finally, when DEADUCATDES, a special procedure, is run (usually once a month) to remove expired userids from the system, the master index descriptor is removed from the master index, and all segments allocated to the expired user catalog are returned.

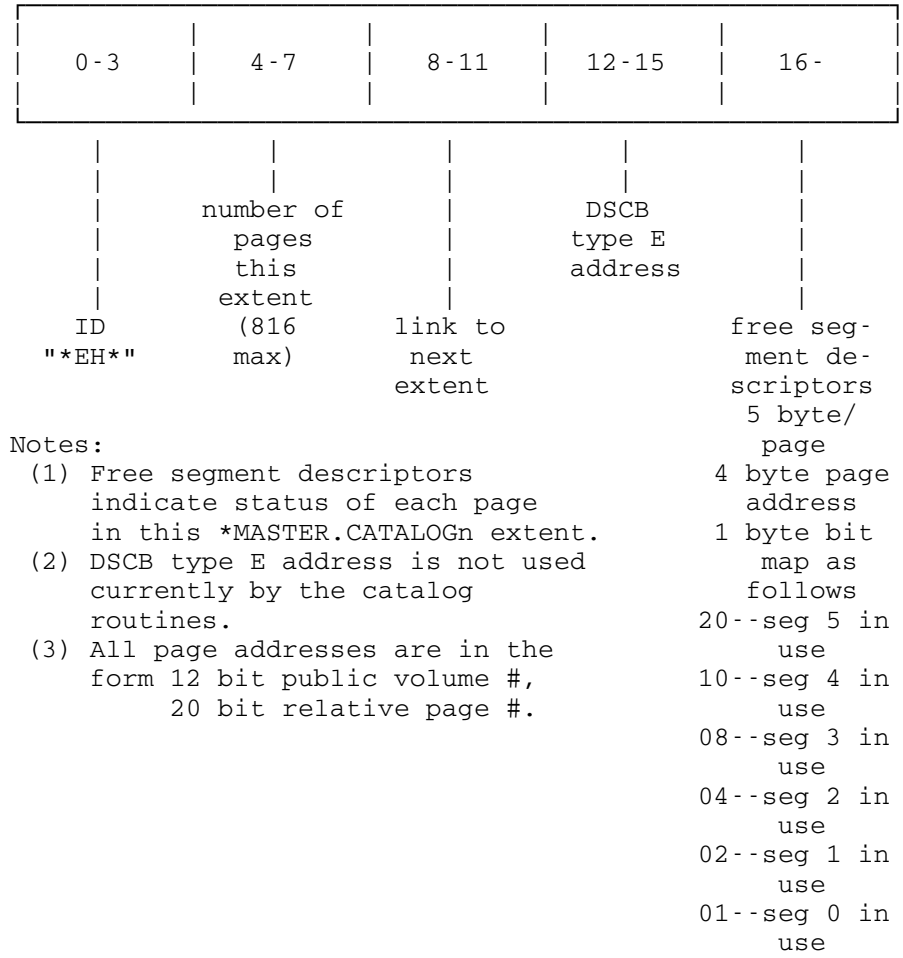
It should be noted here that the system file catalog and the scratch file catalog are identical to the user catalogs except, of course, they never expire.

One further note; the extent header of *MASTER.CATALOG0 contains in addition to the normal extent header information and free segment descriptors, the name of the last *MASTER.CATALOGn created, and the addresses of the beginnings of the master index, the system file catalog, the scratch file catalog, and the first user catalog. These pointers are read in and remembered when the operating system is initialized for reasons of efficiency. (The name of the last *MASTER.CATALOGn is needed for expansion.) In addition, proper manual setting of these pointers at the time the catalog is being built can in most cases guarantee that sufficient contiguous space will be available to the master index, system, and scratch file catalogs for expanding. This will be the case since user catalogs are allocated "down from" the first user catalog only. Again this is an efficiency move and is not necessary (though quite desirable).

Finally, there exist two resident subroutines which may be of use to system programmers interested in extracting information from the catalog about the file system. GETCINF returns file descriptor (and optionally sharing descriptor information) about any or all of the files in the users catalog. READCAT reads the catalog sequentially and returns information on a page by page basis. Since the calling sequences to these routines are rather nonstandard to say the least, the appropriate listings should be consulted.

Catalog Format

Extent Header (page 1 of each *MASTER.CATALOGn file -- also called "file header")



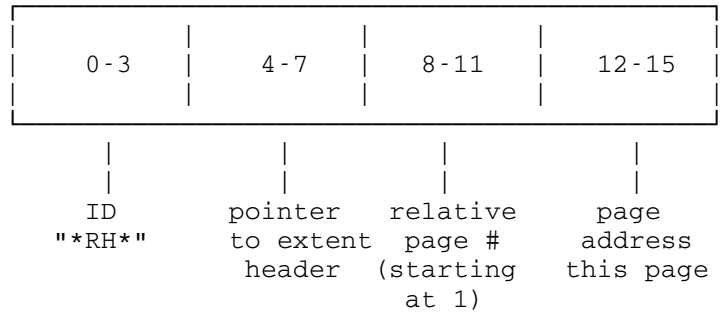
Notes:

(1) Free segment descriptors indicate status of each page in this *MASTER.CATALOGn extent.

(2) DSCB type E address is not used currently by the catalog routines.

(3) All page addresses are in the form 12 bit public volume #, 20 bit relative page #.

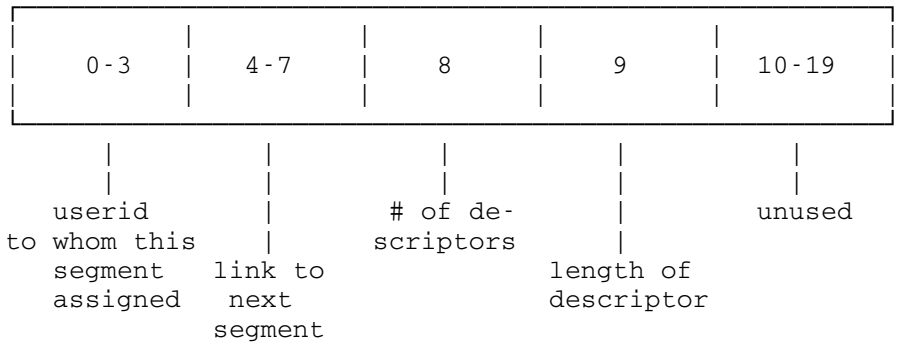
Page Header or Record Header (16 bytes)



The remainder of every page is divided into six segments, each 680 (decimal; hex 2A8) bytes in length. The segments are numbered 0-5. The starting relative address of each segment within a page is:

segment 0: X'010'
segment 1: X'2B8'
segment 2: X'560'
segment 3: X'808'
segment 4: X'AB0'
segment 5: X'D58'

Segment Header (20 bytes)

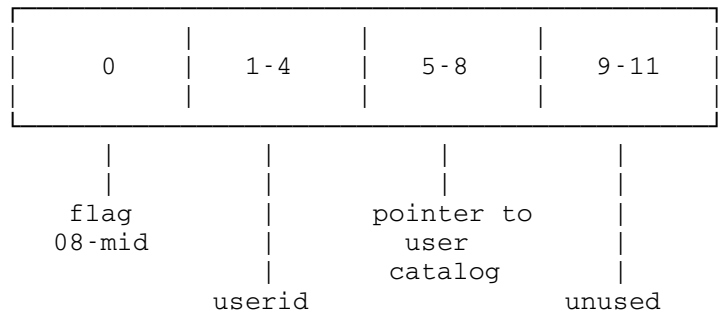


The userid above may also be one of "*MIX" (master index), or "*SYS" (public file catalog), or "<SF>" (scratch file catalog).

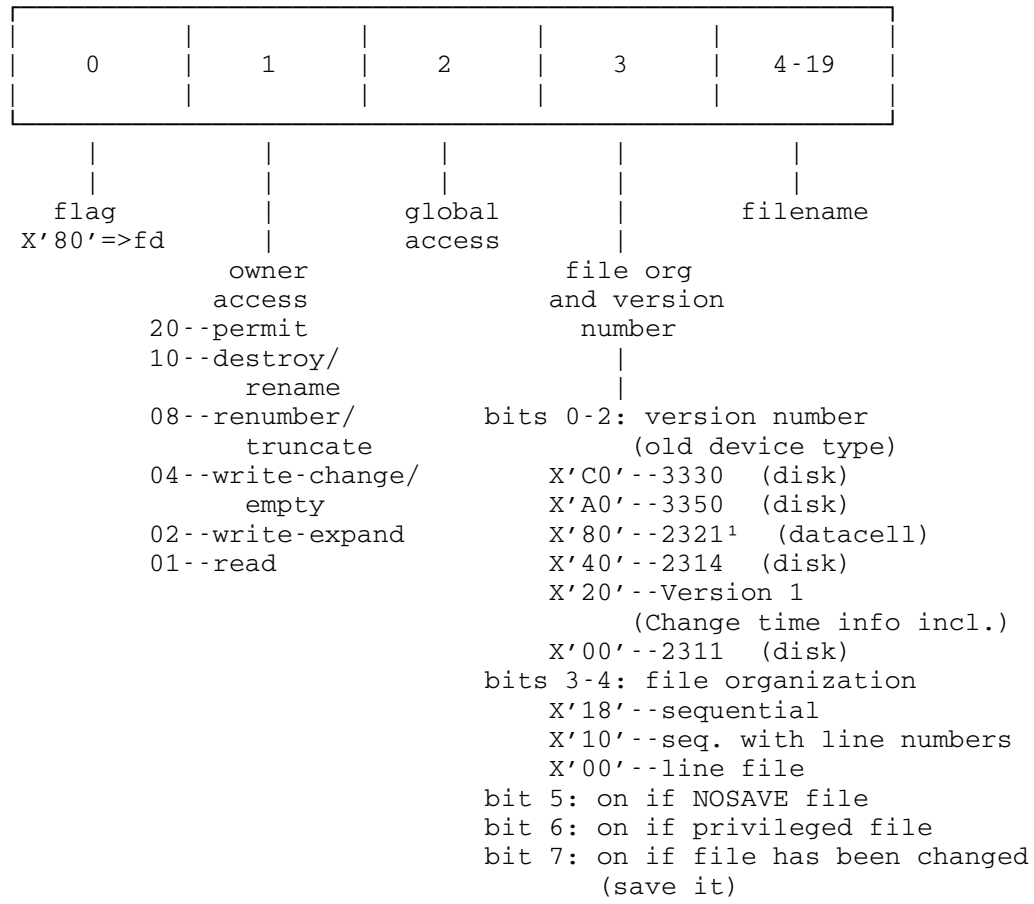
The high order bits of the public volume number (in the link to the next segment) contain the segment number (similar to DSCB addressing). For example, 20404DCF refers to the third segment on page 4DCF on MTS004.

The next segment does not have to be on the same volume.

Master Index Descriptor (12 bytes)

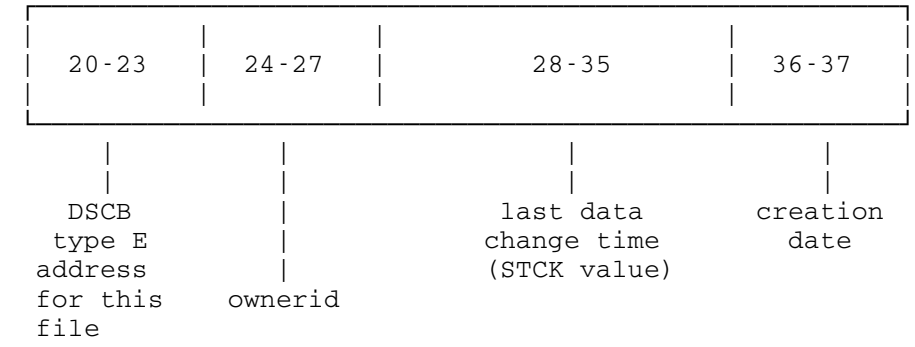


File Descriptor (66 bytes)

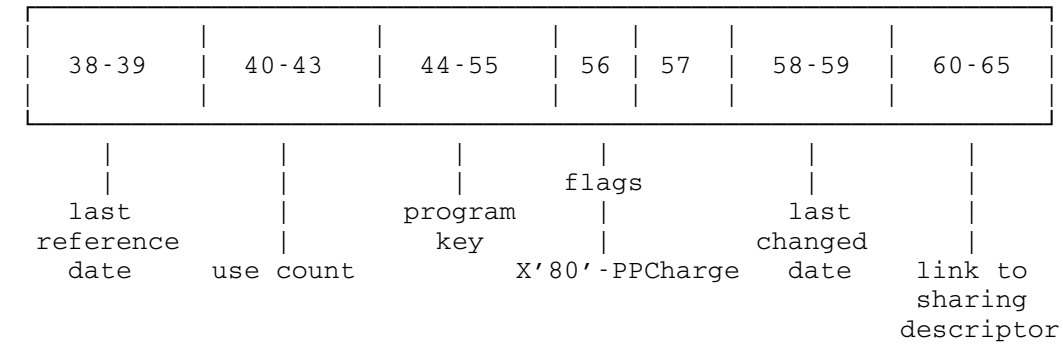


[1]Currently not supported by the file system.

File Descriptor -- continued

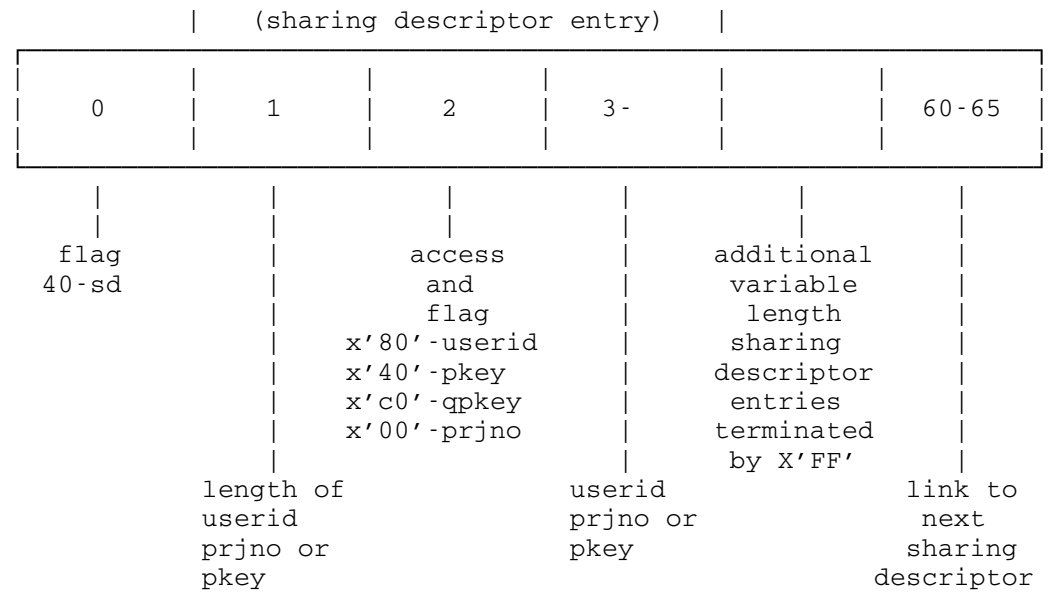


File Descriptor -- continued



Notes:
(1) Link to sharing descriptor is 6 bytes:
12 bit public volume number,
20 bit relative page number,
2 byte offset into segment
(once again the segment number is in the high order bits of the public volume number).
(2) All dates are Julian, that is, based on March 1, 1900.

Sharing Descriptor (66 bytes)



Notes:
(1) See note number 1 above concerning format of link to next sharing descriptor.
(2) For qualified pkeys, the format is:
<len>'8x'<ccid><len>'Cx'<pkey>

THE INTERNAL STRUCTURE OF LINE FILES IN MTS

A line file has two basic components, the line-hole directory and the data section. Logically, the line directory is an array of 8-byte entries, one for each line in the file; each entry contains the line number, plus a pointer (relative page number and offset) into the data section, where further information about the line is stored. This array is ordered from smallest to largest line number, which makes both sequential operations and indexed operations relatively efficient. The format of a line directory entry is:

```
LDELNUM BYTES 0-3:      line number
LDEPAG# BYTES 4-5:      relative page number (1-32767)
LDEDISP BYTES 6-7:      displacement within the page (0-4095)
```

The contents of the data section are unordered, and are allocated and freed dynamically, using a hole directory. The hole directory is an array of four byte entries; there is one entry for each page of the data section, giving its relative page number, preceded by an entry for each contiguous block of available space on that page, giving its offset and length. Each such group of entries is ordered from highest to lowest offset, to facilitate recombination of available blocks. No particular order is imposed on the groups themselves, however. The relative page number entry appears last in each group because the hole directory is scanned in reverse order. The format of a hole directory page number entry is:

```
HDEPAG# BYTES 0-1:      relative page number (1-32767)
HDEFLAG BYTES 2-3:      zero, which flags this type of entry
```

and the format of an available block entry is:

```
HDEDISP BYTES 0-1:      displacement to beginning of block (0-4095)
HDELEN  BYTES 2-3:      length of available block. (1-4095)
```

We next describe the manner in which these pieces are mapped onto the physical pages of the file. The line directory array is divided into blocks of 510 or fewer entries; each such block is stored on a separate page, and these pages are chained together on a two-way linked list, in increasing line number order. The first page in the chain is always page one of the file. The hole directory follows the line directory in the same chain; only one page may contain both hole directory and line directory entries at the point where they join.

The data section normally occupies a set of pages distinct from the line-hole directory chain. If the file is small enough to be stored in one page, however, the data section occupies part of page one.

The following is the general format of a line or hole directory page:

PHLDSO	BYTES	0-1:	offset to start of line-hole directory; this will be either X'0028' or X'0BE0' in page one, and X'0010' in all other pages
PHLDL	BYTES	2-3:	line directory length (bytes) (0-4080)
PHHDL	BYTES	4-5:	hole directory length (bytes) (0-4080)
PHSID	BYTES	6-7:	relative page number of this page (1-32767)
PHFWDP	BYTES	8-9:	forward pointer (relative page number of next page in hole-line directory chain) (0-32767 0=end of list)
PHBWDP	BYTES	10-11:	backward pointer (0-32767 0=end of list)
PHLNTP	BYTES	12-13:	line number table index (see later) 8-byte-slot number in line number table (0-8180)
	BYTES	14-39:	global file information - page one only - in other pages the line or hole directory starts at byte 16, and bytes 14-15 are unused.
	BYTES	40-3039:	data section for a one page file. In larger files the line directory starts at byte 40 of page one. There is room here for 3000 bytes of data.
	BYTES	3040-4095:	line directory starts here in a one page file. Room here for 131 lines and two hole entries.

We next describe the contents of the data section of the file. For lines shorter than 128 bytes, and many longer lines as well, the line occupies a contiguous block of storage in the data section of the file; the first two bytes of the block give the length of the line, and the remainder is the line itself. A line longer than 128 bytes may be broken into at most 16 pieces, none of which (except possibly the last) may be shorter than 128 bytes; clearly none will be larger than a page. If the line is broken up, the block pointed to by the line directory entry contains a table of pointers and lengths for the remaining blocks, followed by the first piece of the line. The structure of the line block table is as follows:

LINBKTB BYTES 0-1:	BITS 0-3 - number of pieces minus 1 (0-15) BITS 4-15 - length of first piece - does not include length of table (1-4094)
LINPAG# BYTES 2-3:	relative page number (1-32767)
LINDISP BYTES 4-5:	offset (0-4095)
LINPLEN BYTES 6-7:	length of this piece (1-4096)
BYTES 8...	up to 14 6-byte entries in the format of bytes 2-7, one for each piece, followed by the first piece of the line.

There are no alignment restrictions on blocks in the data section. The data blocks for a line may have a total length up to 32767 bytes. The data blocks for the line number table (described later) may have a total length up to 65444 (4096*16-2-15*6), leaving room for 8180 8-byte slots.

If a line or hole directory page becomes empty, normally because all the lines it points to have been deleted, it is removed from the line directory chain and added to the free page chain, which is a one-way linked list of available pages, chained through the normal forward pointer field. The pointer to the first such page, if any, is contained in the global file information in page one of the file. Pages on the free page chain can be used either as data pages or line directory pages. Once a page has been used as a data page, however, it will never be used as a line directory page. Pages beyond the number of pages in use (R1NPGS) are not chained.

To improve the efficiency of indexed operations on line files, a line number table is added to the file. The line number table is an array containing a one-way linked list of 8 byte entries, one for each line or hole directory page, with the following structure:

LTELNUM BYTES 0-3:	Line number of first line in page
LTENEXT BYTES 4-5:	Index of next entry in list (0-8180 0=end of list)
LTEPAG# BYTES 6-7:	Relative page number of corresponding page 1-32767

These entries are chained in exactly the same order as the corresponding line or hole directory pages. The pointers are entry indices, and page one always has index zero. If a line directory page contains neither lines nor holes (a condition which should only occur for page one), the line number table entry contains X'80000000'. If a page contains only holes, the entry contains X'7FFFFFFF'. Recall that each line directory page also contains the index of the corresponding line number table entry.

The line number table is stored in the data section of the file, in exactly the same format as a normal data line. The pointer to this "line" is part of the global file information in page one. Corresponding to the free page chain for line directory pages, there

is a free entry chain for the line number table, chained through bytes 4-5, as usual, and starting from the global file information.

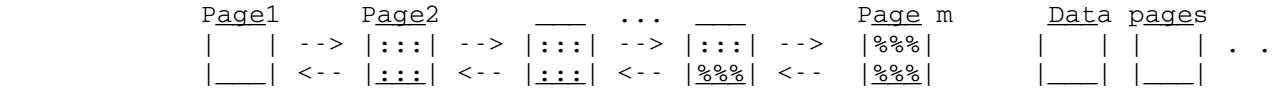
With the exception of the global file information, whose format is given below, this completes the description of the internal structure of line files.

Global file information:

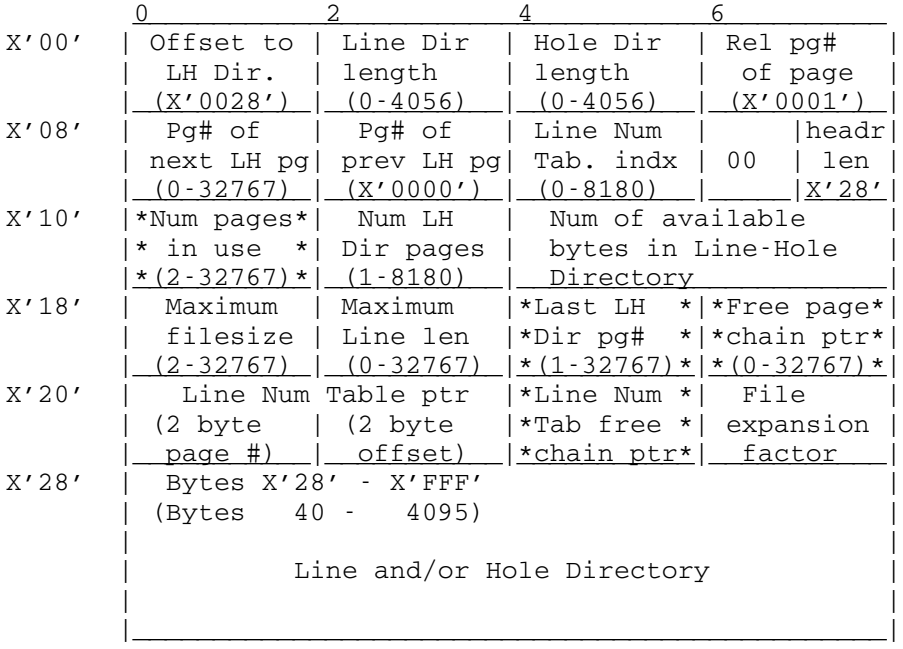
R1FTYPE	BYTE	14:	file type - always X'00'. The first nibble=0 flags that this is line file format; the second nibble=0 because it is unused.
R1HDRL	BYTE	15:	header length - always X'28'=FL1'40'=AL1(LNEFHDRL)
R1NPGS	BYTES	16-17:	number of pages in use (truncated size) (1-32767)
R1NLDR	BYTES	18-19:	number of line-hole directory pages (1-8180)
R1NAB	BYTES	20-23:	number of available bytes in line-hole directory
R1MFS	BYTES	24-25:	maximum file size - file will not be expanded beyond this size (1-32767)
R1MLL	BYTES	26-27:	maximum line length - length of the longest line written (0-32767)
R1LLDR	BYTES	28-29:	last line-hole directory page number (1-32767)
R1FPC	BYTES	30-31:	free page chain pointer (0-32767 0=none)
R1LNTP	BYTES	32-35:	line number table pointer, 2-byte page number (1-32767) & 2-byte offset (0-4095)
R1LNTFL	BYTES	36-37:	line number table free entry list (0-8180, 0=NONE)
R1FXF	BYTES	38-39:	file expansion factor

File Pages for LINE Files

File pages consist of Page One (special), Line Directory pages (:::) and Hole Directory pages (%%%) chained in a doubly-linked list, and Data pages.



1) Page One of a multi-page line file



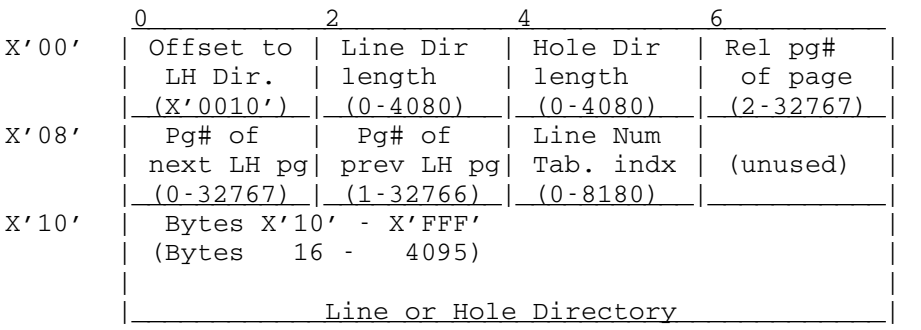
Notes:

1) The information in bytes X'0E' - X'27' is known as Global File Information. This info is updated from the FCB at the time of a CLOSE operation.

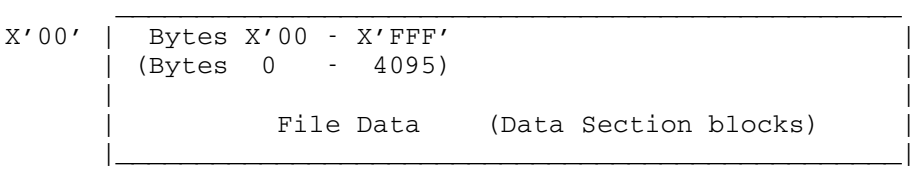
2) Byte X'0E' consists of a high-order nibble 0 which indicates this is a LINE file format and a low order nibble which is unassigned (set to 0).

3) The fields marked with asterisks are the critical fields. That is, if these are incorrect, the file can be very messed up and inconsistent.

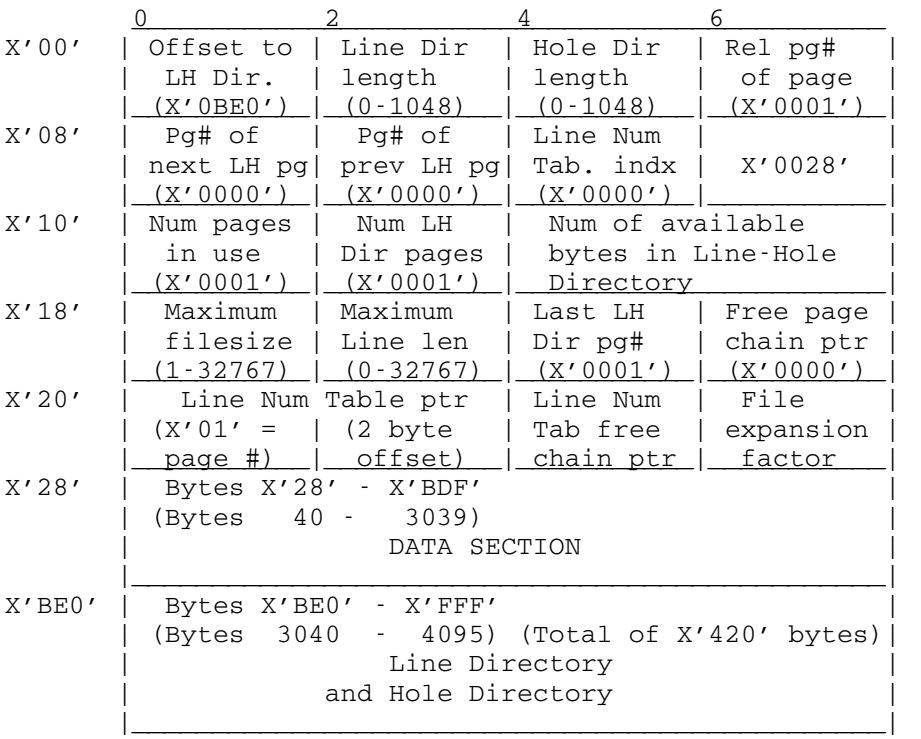
2) Line/Hole Directory; Page n of a multi-page line file (n > 1)



3) Data Section; Page n of a multi-page line file (n > 1)



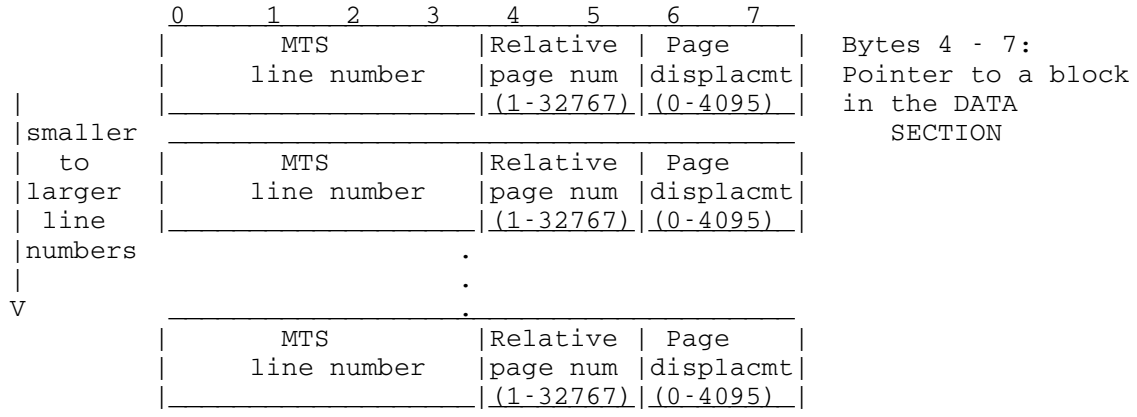
4) A One-page file (contains header, global file info, data, and Line/Hole Directories all in one page).



Structure of Line Files

Line Directory

The Line Directory consists of 8-byte records ordered from smallest to largest line number. There is one entry for each line in the file.



Hole Directory

The Hole Directory consists of one 4-byte page marker for every data page in the file. If one of those data pages contains holes, then the page marker will be preceded by one or more 4-byte hole descriptors (one for each hole in that page).

The file routines scan the Hole Directory from the end of the LH page toward the beginning. Therefore the Hole Directory entries are arranged "from right to left" rather than from left to right as the Line Directory entries are. E.g.,

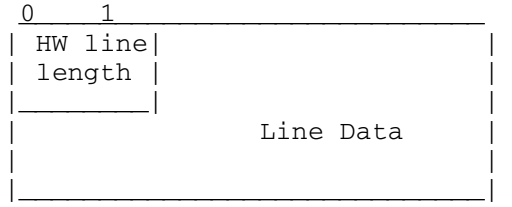
where PM1 is the page marker for the lowest numbered data page, HD1 is the hole descriptor with the smallest displacement, HD3 is the hole descriptor with the largest displacement value, etc. Note that the page described by PM2 has no holes.

Structure of Line Files

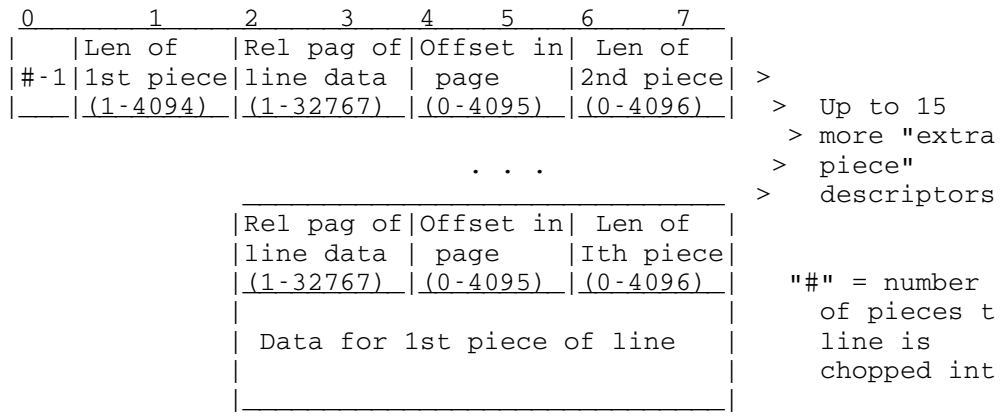
Data Section

The Data Section consists of pages of nonaligned blocks. A block always has a halfword length followed by data, line block table entries, or line number table entries. Possible blocks are:

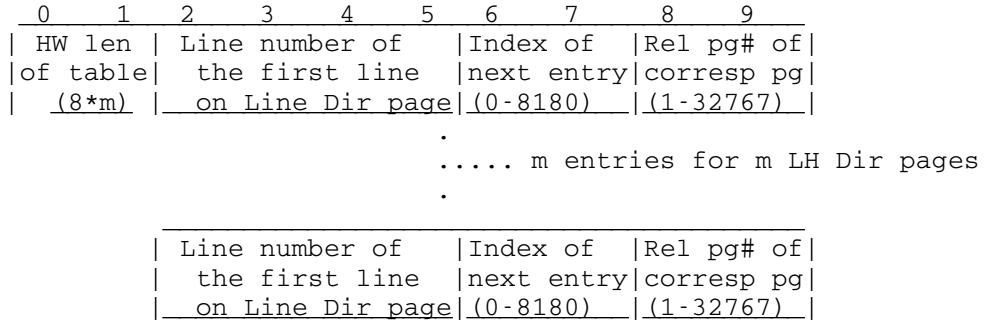
1) Contiguous storage for a line LE 128 bytes (sometimes longer)



2) Line Block Table for a line GT 128 bytes that is described in pieces.



3) Only one Line Number Table exists in the data section.



Note: Bytes 0-3 contain X'80000000' for Page One if it has neither Lines nor Holes and X'7FFFFFFF' for Hole-only Directory pages.

Internal Structure of Sequential Files in MTS

The organization of sequential files (with or without line numbers) is quite simple when compared to line files. In general, the first "n" bytes of the first physical record is used as a header by the sequential file routines in which pertinent information about the sequential file is retained.[1]

Immediately following this header information are the lines of information stored in the sequence in which they were received by the sequential file routines. Since these lines may be up to 32,767 bytes long, and since the physical records on the disk are 4096 bytes (1 page) long,it is quite possible that a line will have to be broken up and stored on more than one physical record. This is quite likely even if only "short" lines are written into a sequential file since the lines are packed end to end using up all of one physical record before going onto the next physical record. Thus, it turns out that even short lines may be broken up across physical record boundaries.

For this reason, it is convenient to refer to a segment of a line as that part of the line which resides on a physical record. Furthermore, we can refer to the first, intermediate, and last segments of a line, remembering that in fact these descriptions may all denote one segment (identical to the line) or they may denote two or more distinct segments, depending on the size of the line and how the line "fell" with respect to physical record boundaries.

The first 4 bytes in the sequential file header are the length of the header, following this is the 4 byte last pointer associated with this sequential file. This pointer is composed of a 2 byte relative record number within the file and a 2 byte offset into the corresponding physical record. This pointer is used to determine where the next line of information should be written and where the logical end of the file is.[2]

The next full word in the header following the last pointer contains the line number of the last line written, and is maintained only if this is a sequential file with line numbers. Its sole function is to insure that lines are written with increasing line numbers. The next halfword in the header is the size of the longest line in the file. This is updated (if necessary) after every write operation. The last

---

[1] Currently n=16

[2] A more detailed description of how this pointer and others are manipulated by the file routines may be found in an appendix to Volume 1, "Details on Using Sequential Files in MTS".

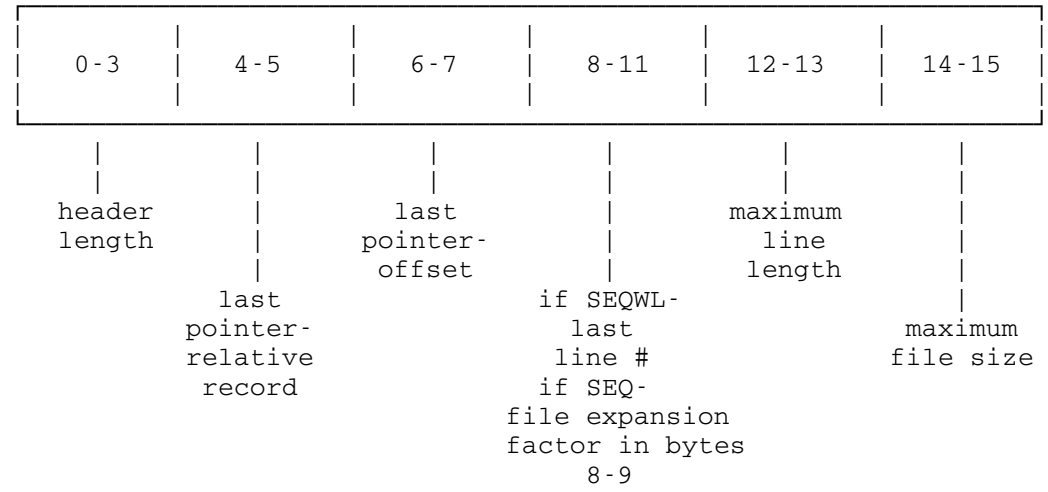
two bytes in the header are the maximum expandable size of this sequential file.

Each segment of a line in a sequential file has either 10 or 6 bytes of overhead associated with it depending on whether it is in a sequential file with or without line numbers. The 6 bytes common to both organizations is split up as 3 bytes before and after each segment. The first of the three bytes at the beginning of each segment is a flag byte indicating whether this is the first, intermediate, and/or last segment of the line, and whether this segment (i.e., the line) has a line number associated with it. The next two bytes are a count of the current segment length plus the previous segment lengths for this line. If this is a sequential file with line numbers, the next 4 bytes contain the line number associated with this segment. The three bytes at the end of the segment are similar (but not identical) to those at the front, i.e., the first two bytes are a count of the total line length minus all previous segment lengths, and the last byte is the one byte flag. The lengths kept at the front and the back of the segments are somewhat obscure but make possible the backwards reading of sequential files. Due to the judicious definition of these lengths, it is the property that: 1. For the first segment of a line, a) the leading count contains the length of the first segment, and b) the trailing count contains the total line length; 2. For the last segment of a line, a) the leading count contains the total line length, and b) the trailing count contains the length of the last segment. This is precisely the information required for forwards and backwards reading of the file.

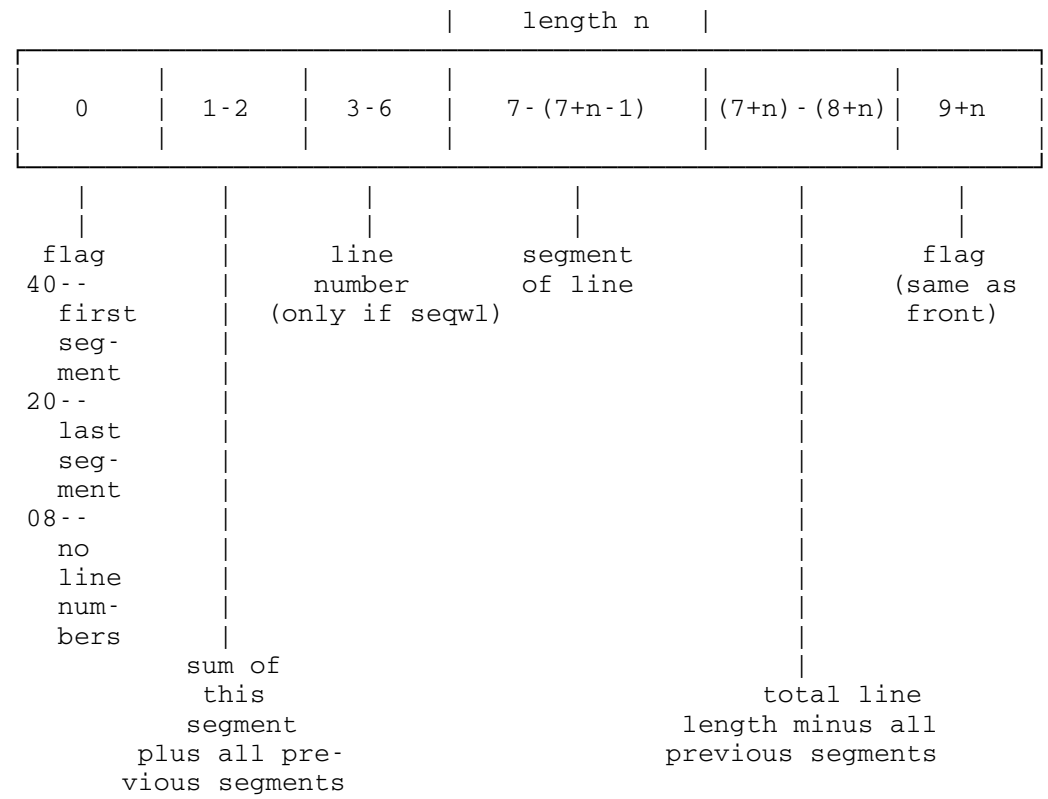
As was mentioned earlier, lines are packed sequentially onto physical records end to end, and are broken up into segments if necessary so that whenever possible all space on the physical record is used. Sometimes, however, because of the overhead associated with each segment, up to 6 or 10 bytes at the end of each physical record may be unusable. If such is the case, the physical record is filled out with the necessary number of a unique dummy byte. This, along with the length at the end of each segment, as previously mentioned, allows the backward reading capability of sequential files.

As concerns size limitations on sequential files, lines are restricted to 32,767 bytes in length. And, the total number of physical pages in a sequential file can be no greater than 32,767. Finally, as with line files, all records of the file must reside on the same volume.

HEADER -- (16 BYTES)



SEGMENT



Structure of Sequential Files

THE INTERNAL STRUCTURE OF THE SYSTEM WIDE SHARED FILE TABLE IN MTS

In a shared file environment, before any operation (reading, writing, emptying, etc.) can be performed on a file, guarantees must be made to ensure that concurrent usage of the file at any particular point in time will not endanger the integrity of the file.

To accomplish this, files are "locked" at one of three inclusive levels (read, modification, or destroy) before any specific file operation is performed. In addition, checks are made before locking is allowed to ensure that certain rules of concurrent usage will not be violated.

It should be noted that the problems of determining allowable concurrent usage of a file are separate and not related to the problem of determining allowable access to a file. It is assumed that by the time the system wide shared file table is interrogated, it has been determined that access appropriate to the locking request has been "permitted".

In order to determine who may concurrently use a file and how at any given point in time, MTS maintains a table (in shared VM) indicating at any given point in time, all the files currently open and/or locked, how they are locked, and by what task (job); as well as what tasks are currently waiting to lock the file and how they are waiting.

This table is necessary to determine (with the aid of the rules of concurrent usage) whether, at any given point in time, a particular type of opening and/or locking can be allowed.

The rules of concurrent usage are as follows:

1) Any number of tasks can have a file locked for reading at the same point in time as long as no other task has the file locked for modification or destroying.

2) Only one task can have a file locked for modification (writing, emptying, truncating, etc.) at any given point in time, and then only if no other task has the file locked for reading or destroying.

3) Only one task can have a file locked for destroying (renaming or permitting) at any given point in time, and then only if no other task has the file open, locked for reading, or locked for modification.

If it is determined, via the rules of concurrent usage, that a file cannot be locked as requested, the task is (optionally) queued to wait

on the file. (Internally this is accomplished via an SVC sleep.) Before a task is queued to wait on a file, however, checks are made to determine whether queueing a task to wait on the file will result in a deadlock situation whereby two or more tasks will wait indefinitely on their respective queues.

The simplest form of deadlock is the "single file" situation. For example, suppose both task A and task B have FILEX locked for reading, and then task A requests that FILEX be locked for modification. Since someone else (task B) also has the file locked, task A will be queued to wait on FILEX. Then suppose task B requests that FILEX be locked for modification. MTS realizes not only that someone else (task A) has the file locked, but also that queueing task B to wait on FILEX would result in both tasks A and B waiting indefinitely for the other to unlock the file. In this situation, MTS will not queue task B to wait, but will return an error indication instead.

A "single file" deadlock is fairly easy to detect, more complicated forms of deadlocks can occur when multiple files are concerned. The method MTS uses to detect "multiple file" deadlocks is as follows:

1) Define a relation B (Blocking) as follows:

   TASKA is in relation B to TASKB
   (TASKA B TASKB iff)

   TASK A has a file open and/or locked in such a way that TASK B is blocked from using that file.

   Blocking is defined as follows:
   A) A task with a file open blocks a task waiting to destroy the file.
   B) A task with a file locked to read blocks a task waiting to modify or destroy the file.
   C) A task with a file locked to modify blocks a task waiting to read, modify, or destroy the file.
   D) A task with a file locked to destroy blocks a task waiting to open, read, modify or destroy the file.

2) Build the M by M Matrix representing relation B where M is the total number of tasks either (a) with files open and/or locked blocking another task or (b) being blocked.

3) The transitive closure relation B+ of relation B is defined as follows:

   TASKA B+ TASKB iff

   There exists N tasks TASKi $1 \le i \le N$ such that

   TASKA B TASK1 B...B TASKN B TASKB

   (i.e., there exists a "chain" relating TASKA to TASKB).

Structure of Shared File Table

4) Using Warshalls algorithm, (see Gries--Compiler Construction for Digital Computers) compute the M by M Matrix which represents the transitive closure relation.

5) Now see if there exists an i such that

TASKi B+ TASKi

If so, then a deadlock situation exists.

A necessary condition for a "multiple file" deadlock is that the task being queued to wait on a file must have some other file open and/or locked and some other task must be waiting on that file. This check can easily be made to determine if it is really necessary to build the matrix.

Once a task is queued to wait on a file, it "sleeps" until the task(s) which have the file locked, unlock the file. At that point, the unlocking task determines if any task(s) sleeping on the wait queue can be "awakened". The unlocking task makes its decision using the same rules of concurrent usage described above.

The basic format of the system wide shared file table is as follows. The first 2 bytes at the beginning of the table are used by tasks to "wayt" when the table is full. 1 byte is used to "wayt" for space for a file entry, and 1 is used to "wayt" for space for an open or waiting element. The next 2 bytes are a pointer to the chain of open and/or locked file entries. Then follows 2 bytes which are a pointer to a chain of available file entries (each 24 bytes). After that 2 bytes which are a pointer to a chain of available open or waiting elements (each 6 bytes). The next two bytes are a count of the number of open and/or locked files. After this is a 2 byte count of the number of matrix computations performed and a 2 byte count of the number of deadlocks detected. These last 6 bytes are maintained for informational purposes only.

Initially the table contains only available entries. As open or waiting elements are needed, a 24 byte available entry is broken up into 4-6 byte available elements. Eventually, when the open or waiting elements are returned, the 4-6 byte available elements will be "re-grouped" into a 24-byte available entry.

The 24-byte open and/or locked file entry consists primarily of the 16-byte name and a 2-byte link to the next open and/or locked file in the chain. In addition, 1 byte is used to indicate whether the file is being modified or destroyed. 2 bytes are used as a pointer to the chain of open and/or locked elements (tasks with the file open and/or locked), and 2 bytes are used as a pointer to the chain of waiting elements (tasks waiting to lock the file).

Structure of Shared File Table

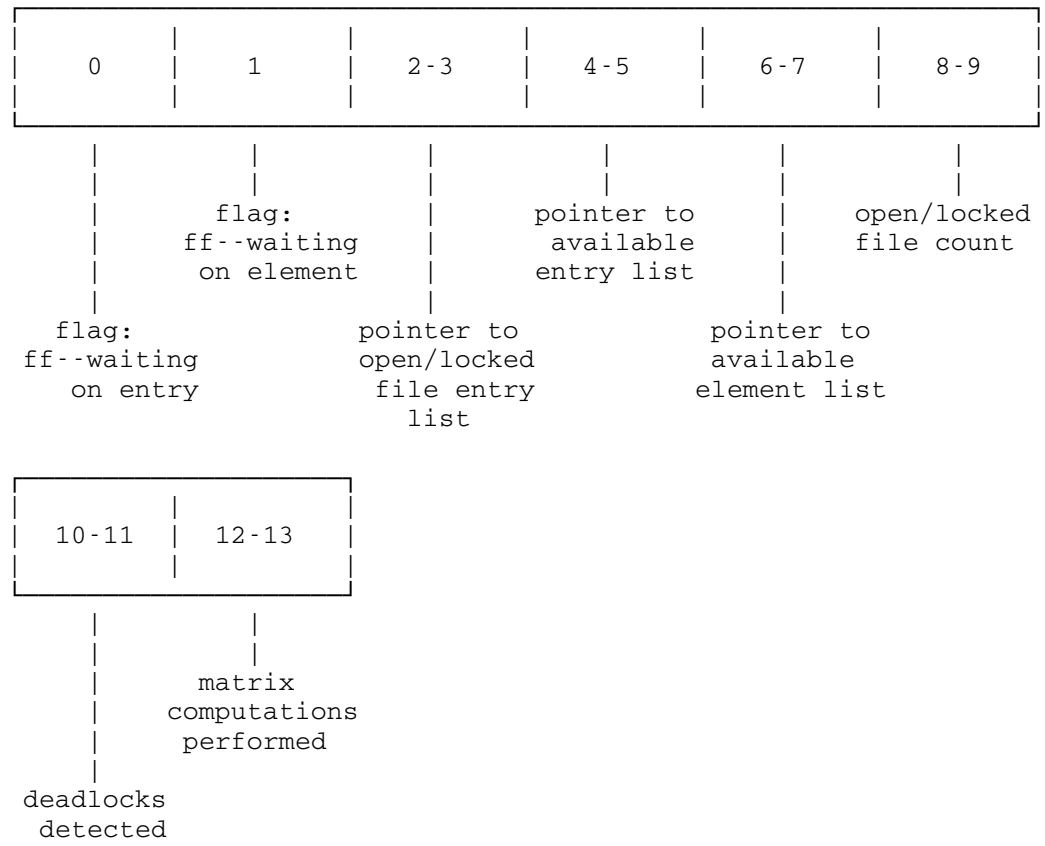
The 6-byte open/locked element consists primarily of a 2-byte task number indicating the task that has the file open and/or locked and a 1-byte flag indicating whether the task has the file open or not and if locked, how the task has the file locked. In addition, of course, a 2 byte pointer to the next open and/or locked element is necessary.

The 6-byte waiting element is identical to the 6-byte open element except that the flag byte indicates how the task is waiting to lock the file. The flag byte also indicates whether the wait has been cancelled. Finally, the flag byte contains the bit on which the waiting task "sleeps" and correspondingly is "awakened".

Structure of Shared File Table

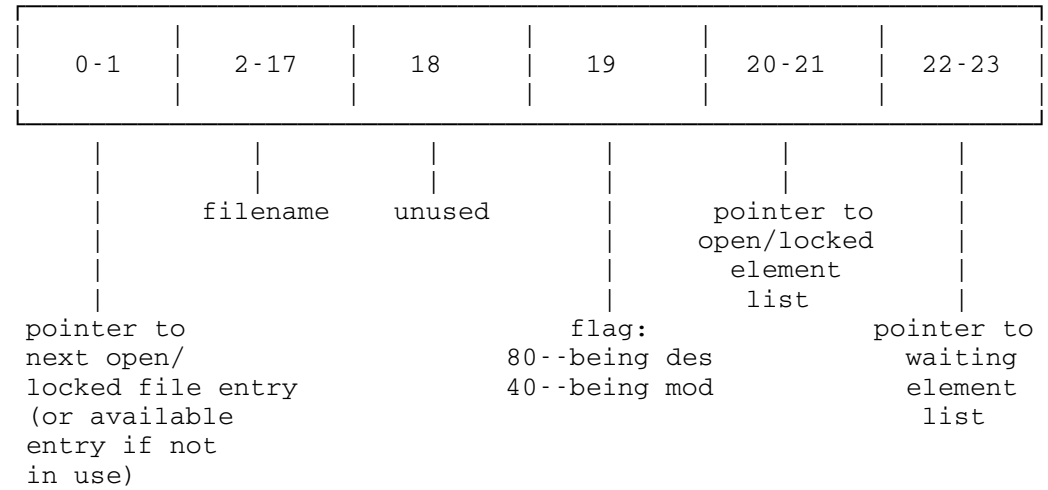
System Wide Shared File Table Format

Table Header



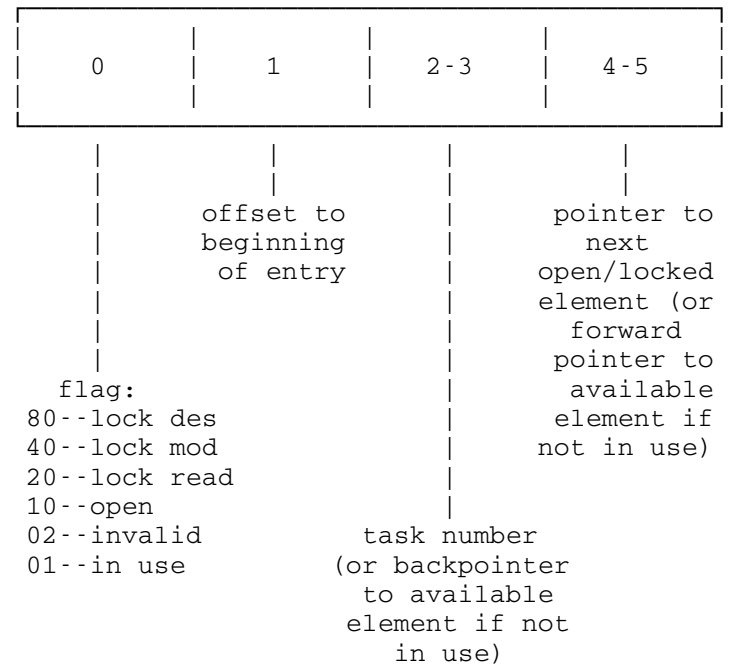
Structure of Shared File Table

File Entry (24 BYTES)

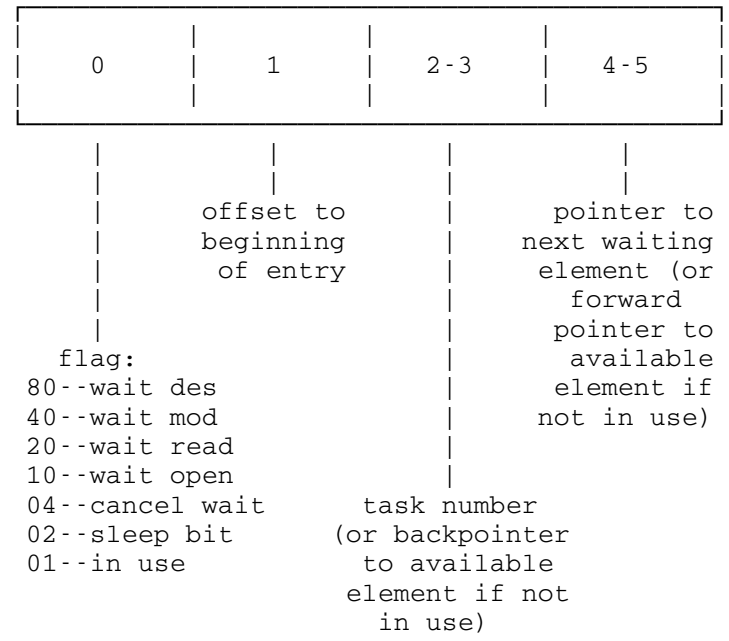


Structure of Shared File Table

OPEN/LOCKED ELEMENT (6 bytes)



WAITING ELEMENT (6 bytes)



Structure of Shared File Table

AMALCOMP

Purpose: To amalgamate and compare the lists of files during recovery from a disk disaster, and so produce lists of files for further recovery actions.

Availability: Use the AMALCOMP macro in file:cmdmaclib to run this program on the standard list files. This macro takes the following parameters:

fdir= the online filesave directory (rstr:filedir.mas), or a copy of it.

time= the time of the disaster. This should be specified in a form acceptable to the PLUS time and date routines, for example "Sep 18 10:45". If specified, this time is used to select which versions of files to restore from the filesave tapes. Specifically, no version which is more recent than the given time will be used. If the time is not specified, the most recent versions are used. This time parameter is useful if a file save has been done since the damage occurred. By specifying the time of the disaster, corrupted versions of files which have been saved on tape since the damage will not be used to "restore" the files.

I/O units used:

SERCOM - Serious errors which affect further processing and informational progress reports.

Unit 0 - The list of user ids which have suffered catalog damage.

Unit 1 - The list of files which have lost data. This is "list 1" of the disaster recovery writeup and is the output of the VTOCUtil program.

Unit 2 - The list of files which have lost sharing information. This is "list 2" and is the output of the FIXSD program.

Unit 3 - The list of all (non-scratch) files in the catalog. This is "list 3" and is the output of the Catlist program.

Unit 4 - The list of all (non-scratch) files in the VTOCs. This is "list 4" and is the output of the VTOClist program.

Unit 5 - RSTR:FILEDIR.MAS (or a copy of it). This is used to find the tape copies of files to be restored.

Unit 15 - The list of files to be destroyed because they are

missing some DSCB or data that can't be restored. This is "list 5" and is input to the CALLDR program.

Unit 16 - The list of files to be recataloged because they are missing an FD that can't be restored. This is "list 6" and is input to the RECAT program.

Unit 17 - The list of files whose data pages are to be released because they are not in the catalog and there hasn't been any catalog damage. This is "list 7" and is the input to RELDSK.

Unit 18 - The input to FASTRESTORE, used to restore various parts of files. This is "list 8".

Unit 19 - The input to *FILES, used to tell users what happened to their files. This is "list 9".

Return Codes:

0 - Everything worked, except possibly errors affecting only individual files.

4 - Serious errors occurred.

12 - Invalid PAR field.

Decision Table: In this table Y means yes, - means no, and ? means don't care.

cat bad	lost data	lost si	in cat	in vtoc	on tapes	lost users	mult vers	action
Y	Y	-	-	Y	Y	?	-	fd si data
-	Y	-	-	Y	Y	Y	-	fd si data list0
Y	Y	-	-	Y	-	?	-	note reldsk
-	Y	-	-	Y	-	Y	-	note reldsk
?	Y	-	Y	Y	Y	?	?	data
?	Y	-	Y	Y	-	?	?	destroy
Y	Y	Y	Y	Y	Y	?	?	si data
Y	Y	Y	Y	Y	-	?	?	destroy
Y	-	Y	Y	-	Y	?	?	si dscb data
Y	-	Y	Y	-	-	?	?	destroy
Y	-	Y	Y	Y	Y	?	?	si
Y	-	Y	Y	Y	-	?	?	note si
?	-	-	Y	Y	?	?	?	nothing
Y	-	-	-	-	Y	?	-	fd si dscb data
Y	-	-	-	-	-	?	-	note cat
-	-	-	-	-	Y	Y	-	fd si dscb data list0
-	-	-	-	-	Y	-	-	nothing
?	-	-	Y	-	Y	?	?	dscb data
?	-	-	Y	-	-	?	?	destroy
Y	-	-	-	Y	Y	?	-	fd si
Y	-	-	-	Y	-	?	-	recat
-	-	-	-	Y	Y	Y	-	fd si list0
-	-	-	-	Y	-	Y	-	recat list0
-	?	-	-	Y	?	-	-	reldsk
?	?	-	-	Y	?	?	Y	reldsk

Note that the correct functioning of this program depends on a subtle use of the "Mult Vers" test - namely that if there are multiple versions which appear only in the VTOCs, then the first one will NOT have the Mult Vers test true. Also if there is a version in the catalog, then Mult Vers will be on for all versions and further there will not be multiple versions in the catalog.

CATSCAN - Catalog scan and count utility

The catalog scan program CATSCAN is used to scan the catalog and report counts of various occurences in the catalog.

To run the program,

```
macrolib file:cmdmaclib
catscan [listoutput]
```

The program begins with a prompt for parameters. The parameters which it accepts and their meanings are:

FILES - Output a count of all files.

LINE - Output a count of all line files.

SEQWL - Output a count of all "sequential with line numbers" files.

SEQ - Output a count of all sequential files.

SHAREDFILES - Output a count of all files which are shared.

OTHERS - Output a count of all files which are permitted to "others".

PROTOFF - Output a count of all files which have PROT=OFF.

NOSAVE - Output a count of all files which have NOSAVE set.

CHANGED - Output a count of all files which have changed since the last file save. (CHANGED bit is set).

SPECIFIC - Output a count of all files which are permitted specifically somehow (have sharing descriptors) and also output counts for sharing descriptors for: users, projects (departments), program ids, user and program id, project and program id, private program id, *MTS.RUN, *EDIT, other public program id.

PKEY - Output a count of all files which have a program key (id).

NAMES - Output a count of files by length of file name, and for each character the number of file names which begin with it and the number of file names which contain it. Only the twelve character user file name is used for these counts.

BADCHARS - Output a count of the files that have at least one illegal character in their file names.

UNLIMO - Output a count of files that are permitted UNLIM OTHERS.

FULLO - Output a count of files that are permitted FULL OTHERS.

PERMITO - Output a count of files that are permitted PERMIT OTHERS.

PPC - Output a count of files that are marked for program product charging.

ALL - Output all of the above counts.

LIST - list on unit 0 those file names selected by setting LINE, SEQ, SEQWL, PROT, NOSAVE, CHANGED.

This program takes a long elapsed time to run, since it labouriously scans the entire catalog.

CCATL - Catalog Creation Utility

The following describes how to create the file system catalog from scratch.

```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
ccatl tptype=<type> tpdev=<device name> tpname=<volume name>
```

CCATL first prints out:

```
Catalog build program, March 87.
Enter all numbers in decimal.
```

and then prompts you to

```
Enter size of master index in pages.
```

to which you might reply

24 (which is the size used last time at UM in Nov. 1975 and on UB in Mar. 1985).

Then CCATL will print

```
First extent of the catalog must be on public volume 1
```

Then CCATL will read the label on PVN 1 and prompt you:

```
How does MTS001 sound "OK"?
```

because that is the label on PVN 1. If that’s right, you reply

```
OK .
```

Then CCATL prompt you to

```
Enter number of pages to allocate for this extent:
Remember, 1 page per extent used by extent header.
```

The standard reply is

25 (which means the master index will fit <u>exactly</u> in the first extent since the first page is used for the extent header and the other 24 is the size you specified above).

Then CCATL asks you to

Enter size of system file catalog in pages

to which a reasonable reply is

10 (which was used at UM in Nov. 1975. A value of 20 was used for the UB system in Mar. 1985).

Since the first extent of the catalog was completely pre-allocated to the master index (intentionally), another extent must be allocated for the catalog at this time. If one wanted the master index and possibly the system catalog, scratch file catalog and first part of the user catalog all on the first extent, the size in pages of the first extent should have been specified as greater than or equal to the combined sizes of the individual catalogs.

In any event, CCATL now notifies you that the

Requested size has overflowed this extent
Enter "ok" to allocate another extent
"NO" means reprompt for current catalog size .

If you enter "OK", then CCATL will ask that you

Enter public volume number for next extent of catalog

to which your reply might be

2 .

Then CCATL will ask you

How does MTS002 sound (ok)?

and you can say

OK .

Then as before, CCATL prompts:

Enter number of pages to allocate for this extent.
Remember, 1 page per extent used by extent header.

You reply as before

11 (because you want the system file catalog also to be on a single extent on a separate volume all by itself. A value of 21 was used for the UB system, Mar. 1985.)

In a similar fashion you will be asked to

Enter size of scratch file catalog in pages
    (15 was the value used for the UB system, Mar. 1985.)

and

Enter size of (first part of) user catalogs in pages
    (200 was the value used for the UB system, Mar. 1985.)

As before if the requested size of the catalog overflows the current extent, a new extent will be allocated of the proper size and on the volume requested.

When CCATL finishes, it prints out the location of the beginning of each of the catalogs (as a fullword hex disk address).

CHKVTOC

PURPOSE: To verify the correspondence between DSCB's and the PAT, to correct to the PAT where possible, and to verify and correct the label.

USE: macrolib mts:cmdmaclib
macrolib file:cmdmaclib
chkvtoc

Accepts input on GUSER and in the PAR field, with the following formats:

MTSxxx - Verification only, on volume MTSxxx. Inconsistencies are listed.

CHECKALL - Verification only, but for all public volumes. Inconsistencies are listed as the above command does.

MTSxxx FIX - Verification, plus PAT inconsistencies will be corrected, all bad DSCBs will be deallocated, and the volume label will be fixed.

MTSXXX PTYPE P1,P2,...,Pn - Verification, plus prints one line of information about each (decimal) page number in the list. If SPUNCH is assigned when this option is used, an entry is put out on it for each page given in the list, giving the file name and the file's DSCB type E location on the pack.

MTSXXX PDSCB F1,F2,...,FN - Verification, plus the DSCB type E locations of each of the files F1 thru FN are output on SPUNCH if it is assigned.

MTSXXX FINDDSCBS - Verification, plus a pattern match on all unallocated pages to find DSCB pages.

MTSxxx FINDDSCBS FIX Identical to the above option, except the PAT is fixed in the following ways: a) all discovered DSCB pages pat bytes are set properly; b) all data pages discovered as a result of their DSCB being found are flagged as in use in the PAT. After CHKVTOC does its thing, it should be rerun with the verify option to really figure out what happened, because the PAT will still not be fully consistent if any of the pages described by the discovered DSCBs were

re-allocated. When they show up next time as being doubly-allocated, then it's your ball game--.

CHONID - Program to Change File Owner

PURPOSE: To change the owner ID associated with a file.

USE:
```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
chonid
```

Input data consists of filenames (internal format, starting in column 1) followed by an ID (also internal format). Pairs of filenames and IDs are read from GUSER until an end of file is encountered.

For example,

```
  # macrolib mts:cmdmaclib
  # macrolib file:cmdmaclib
  # chonid
   Filename and new ID? *files mts.
            *FILES owner was FILE; now is MTS.

   Filename and new ID? end of file
  #EXECUTION TERMINATED
```

The above run changes the owner ID associated with the file "*FILES" to ccid MTS.

The ID MTS has unique access privileges. The ID MTS has "read" access to all files on the system, and also has access to all of a file's sharing information (this enables the FM (filemove) program to copy access information - if it is run under the ID MTS).

DASDI - Disk Pack Initialization

PURPOSE: To label, re-label, or format disk packs according to either the VAM2 or VAMX conventions. The following devices can be DASDI'ed: 2301, 2311, 2314, 3330-I, 3330-II, 7330, 3340, 3344, 3350, 3370, 3375, 3380, 3390, 6280, 9332, 9335, VM minidisks.

USE:
```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
dasdi
```

Input data consists of the following operands starting in column 1:

```
Dxxx MTSxxx {VX|V2} {pvn#|PAGING|PRIVATE} [optional pars]
```

Parameter descriptions follow. In any place a number is called for, a decimal number may be given, or X'hex-number'.

pvn --> public volume number (if it is to be a public volume)

PAGING --> if it is to be a paging volume.

PRIVATE --> if it is to be a private volume.

LO or
LABELONLY --> if the volume is to only be labelled or re-labelled (as opposed to being formatted).

IPL or
IPL=nnn --> if it is desired to leave "nnn" pages of IPL area starting at the front of the pack (as well as formatting the pack). The IPLINIT program can place a core image of the IPLREADER program in these pages. (IPLREADER is the program which decides which system to load and loads it into the bare machine.) The area is reserved by generating the necessary DSCBs to describe the IPL pages, writing them onto the disk, and marking the DSCB and IPL area pages properly in the PAT. If "nnn" is not specified, the maximum number of pages which can be described by a single full DSCB page is allocated - 968 (=38+15*62).

DASDI - Disk Pack Initialization

IPLSTART=nnn --> used in conjunction with the IPL keyword, the "nnn" value determines at what page the IPL pages should begin. This defaults to the first available page on the pack if not specified.

PAGES=nnn --> format the pack as if it had only "nnn" pages.

PATA=nnn or
PATSTART=nnn --> PAT should start at the given page address.

CLEARPAT --> construct and write PAT and IPL DSCBs - useful only with the LO option.

NOSLOW --> override the PAR=SLOW (see below) option for this DASDI operation only. This is the default

SLOW --> perform a 50 millisecond wait between formatting writes. This gives other tasks access to the disk's control unit and allows a pack to be DASDI'd on a running system.

The following parameters should only be used for special situations:

WHA --> read and write "home addresses" on the volume. This is almost never necessary on "modern" disks and is very dangerous to do since it may cause information essential to proper error recoverey to be lost. This parameter is necessary on brand new disks, however, to insure that the alternate tracks are correctly formatted. If this parameter is not given, then the VIRGIN and the ALTERNATES parameters are illegal.

WR0 --> write "record zero" on the volume. This defaults ON, is necessary on new disks, and never hurts. However, VM does not allow guest machines to write record zero (or home addresses) on minidisks (except dedicated packs and full-pack minidisks). Therefore NOWR0 is required when formatting VM minidisks. Normally, the minidisk should have been formatted under CMS using the CMS FORMAT command with the BLKSIZE 4K option. This program should then be run (under MTS) with the NOWR0, LO, and CLEARPAT options.

VIRGIN --> try to read "home addresses" but don't

complain if they can't be found. If the label is unreadable when this option is specified, DASDI will continue on.

ALTERNATES=nnnnn
ALTERNATES=(nnnnn,mmmmm)
ALTERNATES=NORMAL
ALTERNATES=KEEP
ALTERNATES=NONE (the default)

--> DASDI will also format tracks which are marked as alternate tracks in the home address, even though MTS does not use alternate tracks.

The first and second forms designate the range of the relative track numbers (starting at relative track 0) for tracks to be assigned as alternates. Note that if "ALTERNATES=KEEP" is also given, then any other track which is already an alternate will be kept as an alternate in addition to the track range explicitly specified.

You cannot explicitly specify alternates for 3340/3344/3350/3370 devices. You can either specify "ALTERNATES=KEEP" or "ALTERNATES=NONE" for these devices.

You cannot specify anything other then "ALTERNATES=NONE" unless you also specify "WHA".

"ALTERNATES=NORMAL" means that the track range for the alternates is as is found in the relevant component description.

EXAMPLE:
```
$RUN FILE:DASDI
Execution begins
MTS DASDI program (EB265). Enter input line:
Dddd llllll Vx #/PAGING/PRIVATE pars ...
D354 MTS009 VX 9
D354 CURRENTLY LABELLED AS "OLD009". PLEASE CONFIRM.
OK
PAT TO BE WRITTEN ON PAGES X'yyyyyy' THRU X'zzzzzz'
Enter next input line:
Dddd llllll Vx #/PAGING/PRIVATE pars ...
D355 UNUSED V2 PRIVATE
LABEL IS UNREADABLE. ENTER "OK" TO CONTINUE.
OK
PAT TO BE WRITTEN ON PAGES X'yyyyyy' THRU X'zzzzzz'
Enter next input line:
Dddd llllll Vx #/PAGING/PRIVATE pars ...
D340 SPOOL1 V2 PRIVATE LO
```

```
D340 CURRENTLY LABELLED AS "OLDSPL". PLEASE CONFIRM.
OK
Enter next input line:
Dddd llllll Vx #/PAGING/PRIVATE pars ...
$endfile
Execution terminated
```

The underlined portion represents responses entered by the user.

Disaster Recovery

Procedure for Recovering From Arbitrary Lost Pages (or Tracks or Volumes)

0. Some general hints:

   a. Do a %BUFFER=32.

   b. $LOG on some permanent file.

   c. Always $LIST the output files from each step so that they will appear in both the conversation buffer and in the $LOG output. This makes it a whole lot easier to back up if something screws up. (Actually, you really don't want to list either "list3" or "list4", and "listvntd1" and "listvntd2" are listed by the macros.)

   d. $Empty list? . If any steps can be skipped, then there is no risk of having lists of files from the last disaster.

   e. If the disaster has damaged files which are needed for the recovery (or the basic running of the system), then the recovery must be done from a backup system. Many of the macros take a "tables=" parameter to provide a set of alternate tables describing the desired file system. The object deck supplied via this parameter should be just the UMMPS tables, with all disks as NODMGR volumes and no fake devicelist. If you want to also use a non-standard set of file routines, then use the "frtns=" parameter in addition to the tables parameter.

   f. $set ebm=h and etm=h so times will be recorded in the log

   g. Keep a log on paper of what is going on so that the "next shift" programmers will know what has been done.

   h. There are references sprinkled throughout these instructions to files like "list1, list2", etc. These are the names which the macros will use by default. The prefix used by the macros can be set using the "set_default" macro, q.v.

   i. Turn macros on and attach FILE:CMDMACLIB.

1. Reformat and zero all damaged pages (or appropriately redefined alternate pages). One way to accomplish this is by using the VAMREC program. Remember to release the volume from the Disk Manager before running VAMREC. Another way is to re-dasdi a new volume using the DASDI program and copy the bad pack to the just-dasdi'd pack using the DISKCOPY program. Record the damaged page numbers reported by VAMREC or DISKCOPY for use with VTOCUTIL in step 15.

1.5 Run CHKVTOC and check all the volumes. This provides a basis for comparison on the state of the disk subsystem.

2. Use the VNTD program (T,C-trace the catalog-) to check to see if extent header of catalog was lost. If so, rebuild extent header from DSCB if possible (non-existent program) or restore extent header from filesave tape (currently not saved).

3. Use the FIXEH macro to run the FIX EXTENT HEADER program which reads all catalog pages in each extent to find zeroed catalog pages and rebuilds the record headers. (This program also has the capability for deallocating the zeroed segments in the extent header, but this function is not needed in the disaster recovery process.) If "pre-allocated" parts of the master index, system file catalog or scratch file catalog have been lost, this program should probably rebuild record and segment headers and relink the segments (currently it doesn't).

4. Use the VNTDOUT1 macro to run the VNTD program (V,C-verify the catalog-) to find out which catalog segments have bad pointers (to lost segments) or which catalog segments are no longer chained to some user catalog (because of a lost segment in the chain or a lost master index). This will produce the file "listvntd1", which is the input to the next step.

5. If the output from VNTD in step 4 indicates that any part of the chain of master index catalog segments has been broken, then skip ahead to step 8.

6. Use the LIST0 macro without a "par=" to run the FIX CATALOG program to chain catalog segments back together. This program uses the file "listvntd1" from VNTD (V,C) as input. It looks at the userid and link field in each affected segment to figure out how to chain the segments back together and which segment is the first in the chain. For every resultant chain of orphaned segments, FIX CATALOG attempts to find a home in the catalog for it. This must be the end of a chain of good segments starting at a master index entry since the master index is undamaged. If no

chain of segments for this userid was discovered ruptured by VNTD (in step 4), then the chain of segments must be truly orphaned due to an inopportune system crash. In this case, the segments are verified to be empty of file or sharing descriptors and if so are deallocated in the extent header. If there are descriptors of some sort in the chain, a message is produced identifying the first segment in the chain so that it can be manually re-chained after examination.

This macro generates the "list0" file of userids whose catalog was damaged. This list of userids is input to the AMALCOMP program in step 19. VNTD can be run at this point with the V,C option for verification purposes. No errors should result.

7. Skip ahead to step 11, since the master index needed no fixing.

8. This (and the following two steps) are only used if VNTD indicated in step 4 that the master index was damaged. Use the LIST0 macro with PAR=FMI to recover any corrupted master index segments. (FIX CATALOG will complain if this is not done.) This will ensure that all retrievable portions of the master index are chained back together so that step 10 does not: 1) rechain the master index after it possibly expanded as a result of the disaster making it appear to have been properly terminated when in fact it was damaged, or 2) re-create a master index entry when in fact it may exist on a section of the master index which was orphaned as a result of the disaster and thus was not found by the catalog verify program.

9. Use the VNTDOUT1 macro to again find out which catalog segments have bad pointers or which segments are no longer in some user catalog. This produces a new version of the "listvntd1" file for use by FIXCAT in the next step.

10. Use the LIST0 macro with PAR=LMI to again run FIX CATALOG, this time to fix the user catalogs. This will add userids to the "list0" file which was produced by step 8. Since PAR=LMI has been specified, FIXCAT may create an entirely new master index entry if the master index for a userid was lost. (If it has to recreate a master index entry, it calls a special entry to CRECAT, (RECRECAT) in the file system to do such.)

    The VNTD program can be run at this point with option V,C to verify that the catalog segment chaining is now fixed.

11. Use the VNTDOUT2 macro to run the VNTD program to verify the affected user catalogs to find out what sharing descriptors are no longer pointed to by file descriptors and which file

descriptors point to lost sharing descriptors. This uses the V,U,*SYS and V,U,*ALL options.

12. Use the LIST2 macro to run the FIX SHARING DESCRIPTOR program which reads the output from VNTD (file "listvntd2") and zeros sharing descriptors not pointed to by file descriptors. (These files will get their catalog information restored if the DSCB is OK or get completely restored if DSCB is bad. The AMALCOMP program will discover this fact.) This program will also zero the chain pointer in the last good sharing descriptor of any good file descriptor and add the name of the file to the file "list2". This file contains the names of files which should have sharing information restored from the filesave tapes.

    The LIST2 macro also sorts list2.

    VNTD may be run again at this point with the V,U,*ALL option for verification purposes. No errors should result.

13. If you have "only" lost one or more entire volumes then create an empty "list1" file and skip ahead to step 17. Otherwise proceed to fix the VTOCs on the affected volumes via steps 14 through 16.

14. Run the VTOCUTIL program using the VTOCUTIL macro.

    14a. If you know a PAT page has been damaged, use the FINDDSCBS and FIX option to rebuild the PAT. The FINDDSCBS and FIX option will succeed reliably if (and ONLY if) bad PAT pages are zeroed. It should ONLY be used if you know a PAT page has been damaged.

    14b. If VTOCUTIL indicates that there are problems with the DSCB chains for any file (or if a DSCB page was zeroed in step 1), use the FIX option to deallocate any DSCB's that have lost a Type E or Type F somewhere in their chain. VTOCUTIL will also update the PAT to reflect the data pages reclaimed due to deallocated DSCB's. (The AMALCOMP program will note these files as being in the catalog but missing from the volume, so they won't be lost without a trace.)

15. Use the the LIST1 macro to run the VTOCUTIL program to determine which files were affected by the damage to the records discovered in step 1. For each affected volume, enter the volume name to VTOCUTIL. VTOCUTIL should find no errors on the volume. (Step 14 should have fixed them.) Then enter the damaged pages as PAGE commands, as instructed by the macro. This produces the file "list1", a list of files and DSCB-E locations which need their data restored. If the output from VTOCUTIL indicates that

any *IPLAREA file has been damaged, then it should be replaced by using the *IPLINIT program.

Use the SORTLIST1 macro to remove any duplicates (and any *IPLAREA or *??ASTER.CATALOG?? files) and sort the "list1" file.

16. Use the VALIDATELIST1 macro to find out which files on "list1" are still consistent. Note that this must be done from the id MTS, since validate requires you to have read access to the file. You should also $SET FILEREF=OFF before doing this, to prevent the references from being recorded. Remove the consistent files from "list1" by manually editting the file. These are files which only had an unused data page damaged and are still valid. I think the VALIDATE program cannot be run from a backup system. If some consistent file is not removed from "list1", then it will be unnecessarily restored.

17. Use the LIST3AND4 macro to run the CATLIST program to produce "list3" (a list of all non-scratch files in the catalog) and to run the VTOCLIST program to produce "list4" (a list of all non-scratch files in the VTOCs). Note - if you are running on the damaged system no files should be created between the production of "list3" and "list4".

18. Determine whether the two key online filesave directories were affected by the disk problem. (These files are named "RSTR:FILEDIR.MAS" and "RSTR:TAPEDIR".) If not, proceed with step 19.

    18a. Check whether the master filesave directory, "RSTR:FILEDIR.MAS" has been damaged. If not, skip to step 18b. If the master filesave directory, "RSTR:FILEDIR.MAS" has been damaged, check to see whether the file "RSTR:FILEDIR.NEW" has also been damaged. If it hasn't, $RENAME RSTR:FILEDIR.NEW AS RSTR:FILEDIR.MAS to get the previous version. (The file save merge program leaves the previous version of the master filesave directory in RSTR:FILEDIR.NEW after it has built a new version.) If neither "RSTR:FILEDIR.MAS" or "RSTR:FILEDIR.NEW" is available, the previous version of "RSTR:FILEDIR.MAS" must be restored off of a file save tape, using *RST. (Maybe we should avoid the possibility of losing these directories by saving copies of them on particular tapes after each run of the file save merge program.) If FSTEST is used to move either directory onto the test pack, SPUNCH must be specified @I@-TRIM.

    18b. Check whether the filesave tape directory, "RSTR:TAPEDIR" has been damaged. If not, skip to step 18c. If it has, the most recent version of this file must be restored from a

file save tape using *RST. Next, it must be determined which tapes have been written by filesave since the last time the directories were saved (this is determined by scanning back over the operators logs). The tape directory entries for these tapes must be deleted for the proper running of the REGENERATE program in step 18c. (To delete the entry for a tape, just delete the line in the file corresponding to the tape's tape number, e.g. line .057 has the entry for tape number 57.)

18c. Find the file save tapes written by file save since the time which the file and/or tape directories were restored from. Then run the REGENERATE program with the options appropriate to the disaster. REGENERATE will rebuild tape directory entries for these tapes if PAR=TAPE is supplied, file directory entries for the files saved on these tapes if PAR=FILE is supplied, or both by giving PAR=BOTH.

18d. If PAR=FILE or PAR=BOTH have been specified, REGENERATE will create files of the name RSTR:FILEDIRnn which must be merged with the old master file restored in step 12a. This should be done by running the file save MERGE program with the option PAR=RECONSTRUCT. (This inhibits MERGE from declaring that files are destroyed if they are not present in the catalog. Since the catalog is being reconstructed at this point, and some files are lost but will be restored later, declaring them destroyed at this point would be wrong.) After MERGE has finished, proceed with step 13.

19. Use the AMALCOMP macro to run the AMALCOMP program. This program compares lists zero through four and the online filesave directory (RSTR:FILEDIR.MAS). It produces lists five through nine, which are the input to the subsequent steps. If you do not want to restore from the most recent tape versions, (perhaps because the disk problem occurred before the most recent file save), then you should specify a time= parameter on the amalcomp macro. It has to be in a form acceptable to the PLUS time and date routines, for example "Sep 17 10:45". For the details of what AMALCOMP does, see the decision table presented later under the heading "File information lost and final file status".

19a. Use SORTLIST8 if there are a lot of files to be restored. (See note below)

20. $Signon to RSTR. Create an empty file to use as a checkpoint file, say "checkpoint". Use the RESTORELIST8 macro with UNIT0=checkpoint to run the FAST RESTORE program to restore data and/or catalog for affected files. This reads "list8" from the AMALCOMP program and asks that the appropiate filesave tapes be mounted. For efficiency, this program uses a special entry to CREATE, (RSTRCRE) which 1) does not initialize page 1 of the file and 2) which returns the page map buffer.

Note: If you have lost a very large number of files, say several volumes worth, it may be faster to use multiple streams to do the restoration. Use the SORTLIST8 macro to sort "list8" into tape order. Then split "list8" into several pieces at tape boundaries and use the FASTRESTORE macro for each stream.

Time passes ... for any disaster worth its salt.

21. At this point you should be able to run on the production system.

22. Use the RECATLIST6 macro to run the RECATALOG program to recatalog files from scratch using "list6" from AMALCOMP. This program also fixes the file type appropriately.

23. Use the DESTROYLIST5 macro to run CALLDR to destroy files with lost DSCB or data using "list5" from AMALCOMP. CALLDR will generate an error message for files whose DSCB was lost, which should be ignored.

24. Use the RELDSKLIST7 macro to run CRELDSK to destroy files on "list7" from AMALCOMP. These files have lost catalog and data (but not DSCB) and no file save information is extant, or these files were uncataloged before the disaster.

25. Use the ACCUPDATE macro to update users disk accounting.

Additional Notes:

1) There is a program (*FILES) which takes as input "list9" from AMALCOMP and tells a user how she personally was affected by all of this.

2) Files lost without our knowledge are those created after the last filesave which lost both DSCB and catalog. (In general, we know about the userid and can tell the user to beware, unless: (1) we lost the user's master index entry; (2) we lost all of the user's catalog segments; and (3) there are none of that user's files in the file save directory.)

3) This procedure has the disadvantage that it may restore some files which were destroyed since the last online filesave. This can only happen if catalog segments associated with that user ID

has been damaged or if master index has been lost, however.

4) Losing a whole volume causes no particular problems (other than the amount of information lost). If MTS001 is lost one would have to initialize an empty master index. If other extents of the catalog are lost the extent headers must be rechained.

APPENDIX

Programs:

1. VNTD - catalog verification;
   GUSER=input commands
   SPRINT=output

2. VTOCUTIL - PAT/DSCB verification and fixing;
   SCARDS=input commands
   SPRINT=output

3. FIXEH - fix extent header
   no input or output

4. FIXCAT - fix catalog segments and master index
   SCARDS=VNTD output from V,C
   0=list of userids whose catalog was affected (list 0)

5. FIXSD - fix sharing descriptor
   SCARDS=VNTD output from V,U,*ALL or V,U,....
   SPUNCH=list of files which lost some sharing information (list 2)

6. CATLIST - list files in the catalog
   SERCOM=errors
   0=list of files in the catalog

7. VTOCLIST - list files in the VTOCs
   SERCOM=errors
   0=list of files in the VTOCs

8. AMALCOMP - generates input for FASTRESTORE.
   reads
   0=list 0
   1=list 1
   2=list 2
   3=list 3
   4=list 4
   5=RSTR:FILEDIR.MAS (or a copy of it)
   writes
   15=list 5
   16=list 6
   17=list 7
   18=list 8
   19=list 9

9. FASTRESTORE - restore data and/or recatalog file from filesave tapes.
   SCARDS=input, list 8 produced by AMALCOMP
   0=checkpoint (both read and added to - initially should be assigned to an empty file)

10. RECATALOG - recatalog files from scratch
    SCARDS=list 6 output from AMALCOMP

11. CALLDR - call DESTRYR to destroy files
    SCARDS=list 5 output from AMALCOMP

12. CRELDSK - call RELDSK to destroy uncataloged files.
    SCARDS=massaged list 7 from AMALCOMP

13. *VALIDATEFILE - Validate internal consistency of files
    GUSER=file name list, each line having a file name and options for that file (probably only ZEROCHECK for disaster recovery use)
    SERCOM=error messages

Auxiliary Programs:

1. REGENERATE - file save file and tape directory rebuilding
2. MERGE - file save file directory updating
3. *FILES - informational program to allow users to determine which of their files were affected by a disk disaster and print out a bulletin concerning the disaster.

Macros:

accupdate: runs *accrestore to update users' disk accounting. Uses as input suitable massaged data from lists 5, 6, 7, and 8.

amalcomp : runs amalcomp program to read lists zero through four and the online filesave directory and produce lists five through nine.
fdir= online filesave directory
time= time of disaster

destroylist5 : destroys the files on list 5 by running the CALLDR program. It asks for confirmation.

fastrestore : runs the fast restore program.
tables= alternate tables
frtns= alternate file routines
scards= input list of files to restore
unit0= checkpoint file

fixeh : runs the fix extent header program.
tables= alternate tables
frtns= alternate file routines
par= null to fix up the record headers, DEALLOCATE to deallocate incorrect segments. For disaster recovery, you should not specify this parameter.

list0 : runs the FIXCAT program which fixes segment chain pointers and produces "list0", the list of userids with damaged catalogs. It reads "listvntd1", produced by the vntdout1 macro.
tables= alternate tables
frtns= alternate file routines
par= nothing (if no master index damage), FMI to fix master index, LMI to fix user catalogs when there has been master index damage.

list1 : runs the VTOCUTIL program to produce "list1", the list of files which have damaged data pages.

list2 : runs the FIXSD program to produce "list2", the list of files which have damaged sharing information. It also fixes any invalid sharing descriptor pointers. It reads "listvntd2", produced by the vntdout2 macro. This also sorts "list1" and removes any duplicates.
tables= alternate tables
frtns= alternate file routines

list3and4 : runs the CATLIST program to produce "list3", the list of all permanent files in the catalog, and runs the VTOCLIST program to produce "list4", the list of all permanent files in the VTOCs. It also sorts these lists.
tables= alternate tables
frtns= alternate file routines

recatlist6 : recatalogs the files on "list6" by running the RECAT program. It asks for confirmation.

reldsklist7 : releases the disk space for files on "list7" by running the CRELDSK program. It asks for confirmation.

restorelist8 : restores the files on list "list8" by running the fast restore program.

sortfile : sorts a file.
in= input file (must be a single line file)
out= output file
keystart= start column of key (default 1)
keylen= length of key
lrecl= record length (default 255)
blksize= block size (default lrecl)
par= other sort parameters

sortlist0 : sorts "list0". It also removes duplicates.

sortlist1 : sorts "list1". This removes duplicates and *IPLAREA and anything matching "*??ASTER.CATALOG??".

sortlist8 : sorts "list8", the list of files to be restored, into tape sequence order. This is handy if you want to do multiple restore streams.

validate : runs the line file validation program.
guser= input file names
par= one-shot file name

validatelist1 : validates each file on "list1" with the ZEROCHECK option. This does not update "list1", that has to be done manually.

vntd : runs the VNTD program.
tables= alternate tables
frtns= alternate file routines

vntdout1 : produces "listvntd1" by running the VNTD program with input V,C.
tables= alternate tables
frtns= alternate file routines

vntdout2 : produces "listvntd2" by running the VNTD program with input V,U,*ALL and V,U,*SYS.
tables= alternate tables
frtns= alternate file routines

vtocutil : runs the VTOCUTIL program.

Files:

listvntd1 : output from VNTD V,C
Output from VNTD
Input to FIXCAT

listvntd2 : output from VNTD V,U,*ALL V,U,*SYS
Output from VNTD
Input to FIXSD

list0 : Userids which lost catalog segments
Output from FIXCAT
Input to AMALCOMP

list1 : Files to have data restored - data lost, DSCB ok
Output from VTOCUTIL
Input to AMALCOMP

list2 : Files to have sharing information restored
Output from FIXSD
Input to AMALCOMP

list3 : Files in the catalog
Output from CATLIST
Input to AMALCOMP

list4 : Files in the VTOCs
Output from VTOCLIST
Input to AMALCOMP

list5 : Files whose data was lost and must be destroyed by calling DESTRYR
Output from AMALCOMP
Input to CALLDR

list6 : Files whose FD was lost and must be recataloged
Output from AMALCOMP
Input to RECAT

list7 : Uncataloged files which must be destroyed by calling RELDSK.
Output from AMALCOMP
Input to CRELDSK

list8 : Files to restore (data and/or DSCB and/or FD and/or SD). This list also contains location of files on filesave tapes.
Output from AMALCOMP
Input to FASTRESTORE

list9 : Files and userids affected
Output from AMALCOMP
Input to *FILES

Decision Table for AMALCOMP Program:

In this table Y means yes, - means no, and ? means don't care.

cat bad	lost data	lost si	in cat	in vtoc	on tape	lost users	mult vers	action
Y	Y	-	-	Y	Y	?	-	fd si data
-	Y	-	-	Y	Y	Y	-	fd si data list0
Y	Y	-	-	Y	-	?	-	note reldsk
-	Y	-	-	Y	-	Y	-	note reldsk
?	Y	-	Y	Y	Y	?	?	data
?	Y	-	Y	Y	-	?	?	destroy
Y	Y	Y	Y	Y	Y	?	?	si data
Y	Y	Y	Y	Y	-	?	?	destroy
Y	-	Y	Y	-	Y	?	?	si dscb data
Y	-	Y	Y	-	-	?	?	destroy
Y	-	Y	Y	Y	Y	?	?	si
Y	-	Y	Y	Y	-	?	?	note si
?	-	-	Y	Y	?	?	?	nothing
Y	-	-	-	-	Y	?	-	fd si dscb data
Y	-	-	-	-	-	?	-	note cat
-	-	-	-	-	Y	Y	-	fd si dscb data list0
-	-	-	-	-	Y	-	-	nothing
?	-	-	Y	-	Y	?	?	dscb data
?	-	-	Y	-	-	?	?	destroy
Y	-	-	-	Y	Y	?	-	fd si
Y	-	-	-	Y	-	?	-	recat
-	-	-	-	Y	Y	Y	-	fd si list0
-	-	-	-	Y	-	Y	-	recat list0
-	?	-	-	Y	?	-	-	reldsk
?	?	-	-	Y	?	?	Y	reldsk

Note that the correct functioning of this program depends on a subtle use of the "Mult Vers" test - namely that if there are multiple versions which appear only in the VTOCs, then the first one will NOT have the Mult Vers test true. Also if there is a version in the catalog, then Mult Vers will be on for all versions and further there will not be multiple versions in the catalog.

DISKCOPY - Copy disk packs

PURPOSE: To copy disk packs from pack to pack or to or from tapes. All disks which can be DASDI'd can be DISKCOPY'd with the following exceptions: 2311s and 2314 would need appropriate Unit Check routines to be developed first; DISKCOPY will not write track 0 (the label) for FBA devices.

USE:
```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
diskcopy
```

All input is read from GUSER and all prompts and error messages are written to SERCOM. Every line read by diskcopy which begins with a "$" is interpreted as an MTS command (passed to CMDNOE). Parameters are separated by one or more blanks. All input requests are preceded by prompts indicating the expected keywords or parameters. The top level of input processing is for copy requests. At the top level prompts are made for identifying the input device type, the output device type and options to apply to the copy operation. An end of file at this level causes a (normal) return to MTS.

A lower level command loop exists for acquiring a particular device. For disks, the device name and corresponding volume label are requested. For tapes, one or more (up to four) fdnames are expected, in the order of their use. If during the course of the copy the list of tapes is exhausted this command loop will be reentered (an end of file at this point, in the middle of a copy will kill the copy and the program). If a tape is rejected, any following tapes in the list are ignored and the tape request loop reentered.

Options for the copy are:

SLOW Causes a 50 millisecond real time wait before most disk operations. If SLOW is not specified on a disk-disk copy, the affected control units will be tied up almost continuously.

IPL Causes track zero IPL records to be copied. This option is forced on for copies to tapes.

SWAP Causes volume labels to be changed. For disk-disk copies the volume labels (and volume serial numbers) are interchanged and the file system is notified of this occurrence. Before the actual interchange is

done, DISKCOPY will ask for verification. If there has been some troble with the copy, it can be aborted safely at this point. For tape to disk copies, the volume label and serial number on the tape replace the disk volume label and serial number. This option has no effect on disk to tape copies -- the tape always receives an exact copy of the disk label record.

AUTORETRY
Causes fail page reads from a disk to be retried 5 times before admitting failure.

EXAMPLE:

```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
diskcopy

Enter "FROM" device type (DISK/TAPE):
disk
Enter device name and volume label (Dxxx MTSyyy):
D008 MTS493
Enter "TO" device type (DISK/TAPE):
TAPE
Enter tape device or pseudo-device name(s):
*T1* *T2* *T3*
Enter options (SLOW, SWAP, IPL, AUTORETRY):
slow
Volume copied: 29453 data pages copied, 2 relocations
Enter "FROM" device type (DISK/TAPE):
$ENDFILE
EXECUTION TERMINATED
```

The underlined portions represent user input.

DSK - Disk Table Utility

PURPOSE: To manipulate the table of disk volumes. Disks may be dynamically added and removed, as well as allowing the drive address to be forgotten.

USE: Enter MTS *DSK on the operator's console, or run SYS:DSK.

The commands are:

ADD MTSxxx
MOUNT MTSxxx
: to add "MTSxxx" to the system. This should be used to add a new MTS volume to a running system when it becomes necessary to expand the available disk space.

CATALOG MTSxxx
NOCATALOG MTSxxx
: these commands affect whether the file system catalog is permitted to expand onto the designated volume.

DELETE MTSxxx
DISMOUNT
REMOVE MTSxxx
: to make the volume "MTSxxx" unavailable to MTS. Doing this on a running system will cause all jobs which reference the pack to receive "Hardware error or software inconsistency" errors. This should only be used on a running system in cases of extreme panic.

DMGR MTSxxx
NODMGR MTSxxx
: these commands control whether the Disk Manager is to be used to manage the specified volume. These commands should NOT be used unless the Disk Manager is installed in your version of MTS. Also, if you are using the version of the PDP which pages through the Disk Manager, you should not remove the Disk Manager from any volume which contains a paging extent.

EXPLICIT MTSxxx
NOEXPLICIT MTSxxx
: these commands affect whether files will be created on the designated volume. If a volume is designated EXPLICIT, nothing will be created on the volume unless explicitly specified via the VOLUME= parameter on the MTS $CREATE command. NOEXPLICIT removes this designation.

FORGET MTSxxx
: to forget the device address of an MTS volume. This causes the file routines to re-initialize the disk table entry for the specified volume on the next reference to it, which is useful if a program such as VAMREC has been used to change information in the volume's PAT which the file routines need to know about (e.g., the pack's relocation entries).

LIST
DISPLAY
: To list all currently defined volumes in the table. Useful for determining which volumes are on which drives. This will also summarize the total amount of space and the total amount of free space in the system.

LIST MTSxxx
DISPLAY MTSxxx
: to list the DSKTAB information for the volume "MTSxxx".

POLR MTSxxx
NOPOLR MTSxxx
: These commands control whether the volume should be considered one of last resort for the creation of files. If it is, no files will be created on it unless insufficient space is available on the other MTS volumes. NOPOLR resets this designation.

SCRATCH MTSxxx
NOSCRATCH MTSxxx
: These commands control whether temporary files can be created on this pack in spite of the EXPLICIT flag. A SCRATCH EXPLICIT disk will be treated as NOEXPLICIT when creating temporary files.

SPACE
AVAILABLE
: to summarize the total amount of space and the total amount of free space in the system.

STOP
END
DONE to end the run. An end-of-file may also be used.

A single command may be specified in the PAR field of the $RUN command.

For those commands that accept a volume name, a list of volume names may also be given, eg.

LIST MTS001 MTS002

EXAMPLES: POLR MTS009
ADD MTS099
NODMGR MTS008

FM - File Move Utility

PURPOSE: To move files from the current file system to a file system on a test pack. The program calls the regular MTS READ subroutine to read files, and the file routine entry points to create, write, and permit files.

USE:
```
macrolib mts:cmdmaclib
macrolib file:cmdmaclib
setdefault tpdev <test pack device name>
setdefault tptype <test pack device type>
setdefault tpname <test pack volume name>
filemove
```

FILEMOVE checks that alternate file system routines are loaded by comparing the V-con "INITCAT" with CNFGINFO information.

Input to FILEMOVE is read from SCARDS, and consists of lines of the form:

<source filename> [<target filename>] [<options>]

<source filename>
: is the name of the file to be moved from the current system. If it is ***DESTROY*** then <target filename> is required and FILEMOVE will destroy that file on the test pack. If a private file is to be destroyed in this manner its ccid prefix must be specified.

<target filename>
: is the name which the file will have when moved to the test system. If the name has a ccid, that id is used as the owner id of the file created in the test system. If the second name designates a public file, a user id may also be prefixed, which designates that user id as the owner of the target file. <target filename> may also be specified as "userid:" in which case the user id becomes the owner of the file in the test system. If <target filename> is omitted it is implied to be identical to <source filename>.

<options> is one or more of:

NODATA*
DATA=NO
: requests that FILEMOVE create a file with attributes of <source filename> (size, type, owner, pkey,

access, ...) but that no data should be moved into it. (The size of this file is reduced to 25 pages if it is larger, unless the SIZE= keyword is specified, qv.)

LSTCHG=<date>
requests that FILEMOVE check to see if the <source filename> has been changed since the specified date and print a warning if so. <date> can be any date/time recognized by the Plus time and date routines.

VOLUME=xxxxxx
requests that FILEMOVE create <target filename> on the specified volume.

SIZE=n requests that <target filename> be created at the specified size. This option is ignored unless ***NODATA*** is also specified.

OWNER=ccid
requests that <target filename> be created with the specified owner. This is exactly the same as specifying it with <target filename> it just looks a little less weird than ccid:*name.

If CREATE fails because the file already exists in the test system, FILEMOVE asks if it is ok to destroy the file by reading from GUSER. A response of OK lets it go ahead. A response of ALLOK tells FILEMOVE not to prompt again, but just go ahead and destroy any files it feels like. FILEMOVE may be run with PAR=ALLOK to tell it to never prompt before destroying files.

All error messages are written on SERCOM.

For example,

```
# setdefault tpdev d104
# setdefault tptype 3380
# setdefault tpname mts600
# filemove
  hasp.tst seg2:hasp vol=MTS601
  OK to destroy "SEG2:HASP"?
  ok
  SEG2:HASP destroyed.
  TSTP:HASP.TST copied to SEG2:HASP.
  end of file
+ User program return
```

causes the file HASP.TST to be moved to the file SEG2:HASP on volume MTS601 in a test system.

Additional examples of FILEMOVE input:

```
file1
file1 VOL=xxxxxx
file1 ***NODATA***
file1 data=no
file1 VOL=xxxxxx ***NODATA***
ccid:file1
*file1
file1 file2
file1 file2 VOL=xxxxxx LSTCHG=07/22/81
file1 file2 ***NODATA*** VOL=xxxxxx
file1 ccid:file2
file1 ccid:*file2
file1 ccid:
file1 owner=ccid
***DESTROY*** file2
***DESTROY*** ccid:file2
***DESTROY*** ccid:*file2
tstp:pag001 pdp.:*pag001 ***nodata*** size=5000p
```

FSTEST - Testing the File Routines

This program provides a simple-minded command language for generating calls to most of the standard file system routines. It may be run with the segment two file routines, or with a private copy. To run with the regular file system, use:

```
macrolib file:cmdmaclib
fstest [debug]
```

To run with private file routines, use:

```
macrolib file:cmdmaclib
fstest frtns=<file routines> [debug]
```

If the optional "debug" parameter is given, FSTEST is loaded via a $DEBUG command, rather than a $RUN command. You may want to load fake DSACC routines also.

Logical I/O units referenced:

```
GUSER:  Command input
SERCOM: Timing and error comments
SPRINT: DISPLAY, GETFINF, FCB, and BCBS output.
```

If private file routines are used, the first command issued must be INITCAT. FSTEST is often used to move files between the current catalog and a test pack. The following example shows how to move the file MTS.:TPFILE from the test pack labelled MTS511 on D011, a 3330 device, to the file CURFILE on the current system.

```
# macrolib file:cmdmaclib
# fstest frtns=file:filertns debug
+ set spunch=curfile@I
+ mod dsktab+12 CL6'MTS511'
+ mod tables CL8'3330D011'
+ break telloper
+ run
- INITCAT
- USERID MTS.
- OPEN MTS.TPFILE
- READI
- STOP
+ stop
```

There are a lot of commands available. They can be abbreviated by any unique initial substring. The shortest abbreviation is underlined in the following list. The parameters to the various commands may be:

1. Hexadecimal string, e.g. AB01
2. Unsigned decimal number
3. Positive line numbers, internal form, e.g. 1000 is line 1.000
4. Character string, e.g. ON
5. Filename, internal form, e.g. <SF>0086T for -T
6. Page number. In the DISPLAY, MODIFY, CLEAR, DUMP, and REWRITE commands, the page parameter may be any of the following:

   a. A decimal number.
   b. "X" followed by a hexadecimal number.
   c. "F" followed by a relative page number (as in a. or b.) in the current open file.
   d. "*", which denotes the same page specified in the last such command. The page is not re-read if * is specified.

There is an internal file control block, with an initial maximum buffer count of five, which may be displayed with the FCB and BCBS commands. Additional BCBS up to the maximum are allocated by the OPEN, WRITE, and COPY commands in more or less the same way MTS does it. The maximum buffer count may be changed by the MAXBUFS command.

All commands which call file system routines will also print the supervisor state and problem state CPU times (in that order), in milliseconds per call. Any non-zero return codes from file routines will be printed as "RC= n".

Commands:

BCBS

Displays the internal buffer control blocks.

CLEAR page offset count [value]

Modifies the specified page at the specified offset with count bytes containing the two-digit hex number value, whose default is zero. Other parameters are as in DISPLAY, except that count is in bytes, not words, and is required.

CLOSE

The current open file is "closed" (i.e. CLOSER will be called). Note: Different paths through the file routines will be taken depending on whether the file has been changed (written) or not.

COPY external-filename

Calls GETFD and READ in the regular system and writes the lines

to the open file. Reading is done @-trim@-ic, and line numbers are preserved. The entire file is copied, unconditionally. This command is useful for copying files to a test pack.

CREATE internal_filename [size] [maxsize] [SEQ]

Size and maxsize are decimal numbers. The defaults are a size of one page, and a maxsize of 255 pages. A line file will be created unless SEQ is specified.

DDUMP [HDR] [DATA] [ON fdname] [page [count]]

Dumps count line directory pages starting at relative page no. page in the current open file. This command is similar to the FDUMP command, except it follows the chain through the line directory page headers rather than sequential page numbers in the file. If the initially designated page isn't a line directory page, the dump will terminate. Use of the HDR option will cause the LH directory page header information for each page in the chain requested to be printed in symbolic format. Use of the DATA option will cause the LH directory page to be dumped in hex format. DATA defaults ON unless HDR is specified. Both HDR and DATA may be specified in which case the data is dumped in both formats. The DDUMP command is provided to dump the contents of the line directory of a file.

DESTROY internal_filename

DISPLAY page {offset [count] | HDR}

Prints count fullwords in hex, at offset into the specified page. Offset is in hex, and count is decimal, and the default count is 1. Use of the HDR option will print out the page header information if the page specified is a Line/Hole Directory page or if it is Page One.

DUMP [ON fdname] [page [offset [count]]]

This command behaves like display, with the following exceptions: 1) the output format is that of STDDMP; 2) an fdname may be specified, and, if omitted, *PRINT* is used; 3) other parameters may be omitted - the default values for page, offset, and count are *, 0, and 1024, respectively.

EMPTY

Empties the open file.

$ENDFILE

End-of-file yields execution terminated.

FCB

Displays the internal file control block.

FDUMP [HDR] [DATA] [ON fdname] [page [count]]

Dumps count pages starting at relative page no. page in the current open file. Use of the HDR option will cause the header information for each line hole directory page to be printed in symbolic format. Use of the DATA option will cause all pages to be dumped in hex format. DATA defaults ON unless HDR is specified. Both HDR and DATA may be specified in which case the data is dumped in both formats. The default fdname is *PRINT*, and the default for page is 1, and for count is 1 if page is specified, and all used pages in the file otherwise.

GETFINF [count] [CHEAP]

Calls GETFINF for the current open file, and displays the result (in hex of course). Count is the number of bytes to be returned and displayed. 20 or less is a short call. The default is 42. CHEAP indicates only the cheap file information is to be returned; all "expensive" information is made zero.

HELP

Prints a list of available commands (but no explanation)

INITCAT

Calls this file system entry. Useful only if running with a private copy of file routines, and in that case should be issued before any other file system calls.

MAXBUFS count

Changes the maximum buffer count in the internal file control block to count and allocates or frees buffers if necessary. The default maximum buffer count is 5.

MODIFY page offset data

Modifies the specified page at the specified offset with the specified hex data, which may be arbitrarily long, and may contain embedded blanks or commas. Page and offset are the same as in the DISPLAY command. If the DISPLAY or MODIFY commands specify a file page, neither the FCB nor the BCBS will be changed, but the MODIFY command also changes the in-core copy if there is one.

MTS

Returns to MTS command mode. FSTEST made be $Restarted.

NOTE

Calls NOTE and prints out the current values of the read, write, and last pointers, and the last line number in the currently open file. A bad return code is issued if the file is not sequential. Note that the last line # is meaningful only for sequential-with-line-number files.

OPEN internal_filename

The specified file is opened so that commands which require an open file will work. If a file is already open it is closed first.

PKEY progkey

Changes the program key used in file system calls to progkey. The default program key is *EXEC.

POINT [read ptr [write ptr [last ptr [last line #]]]]

All arguments are hex values which modify the values of the given arguments in the currently open file. A zero value given for any of the arguments causes the corresponding pointer to be reset to the front of the file. A value of FFFFFFFF causes the corresponding pointer value to remain unchanged. An error return results if the open file is not sequential.

PROJNO pno

Changes the project number used in file system calls to pno. The default project number is the one FSTEST is running under.

READI [flag [line [count [truncation length] [scrwd]]]]

Reads count lines from a line file, starting with line. Flag is the one-byte value passed to READI to determine the nature of the read. The default is x'08' (last op bkwd, this op fwd, not indexed, not skip, no truncation). The "this op" direction bit is copied to the "last op" bit for operations after the first. If indexed is specified, only one op is done, regardless of count. If skip is specified, the write to SPUNCH is also skipped. Defaults are line=-infinity, count=+infinity. Lines are written on SPUNCH with the line number parameter as returned from the read, so that @I can be specified if desired. If the output truncation flag is set, the truncation length parameter should be specified, which defines the maximum amount of bytes which will be transferred from the file buffer to the output area. The real length of the last line read is given if this truncation flag bit is asserted. If scrwd is specified, it is a hex value which updates the scratch word parameter, which otherwise is left unchanged. The scratch word is implicitly zeroed by the OPEN and EMPTY commands.

The flag values are combinations of
x'01' => indexed
x'02' => skip
x'04' => next line file op given
x'08' => sets last op type
x'10' => truncate

READS [flag [count [truncation length]]]

Reads lines from a sequential file. Reads count lines, starting with the read pointer (as set by POINT. The read pointer is zeroed implicitly following an OPEN or EMPTY command.) If flag asserts that truncation of output is to be done, the truncation length parameter should be specified. Default flag is zero (this op forwards, no skip, no truncaton.) The read pointer is updated according to the operation. Lines read from the file are copied to SPUNCH in a similar fashion to READI.

RENAME internal_filename1 internal_filename2

RENUMBER [first [last [beginning [incr]]]]

Renumbers the open file. Defaults are first=-infinity, last =+infinity, beginning=1000, incr=1000.

REWRITE [page]

Writes the current contents of the internal buffer into the specified page. The default for page is *.

STOP

Terminates execution.

TRUNCATE

Truncates the open file.

USERID id

Changes the userid used in the file system calls to id. The default userid is the one FSTEST is running under.

VOLUME n

Specifies a public volume number to be used in subsequent display and modify commands. If no VOLUME command is given, public volume 1 is used.

WRITE line [count [length [incr]]]

Writes count lines, length bytes long, starting at line,

incrementing by incr. The write is done to the open file, and WRITES or WRITEI is called, depending on file type. Length should not exceed 32767. The defaults are count=1, length=10, incr=1000. The line written is length initial characters from the repeated string "0123456789".

PM - Obtain a Pack Map

PURPOSE: To obtain a pack map consisting of:

- -volume label dump
- -PAT dump
- -relocation entries listing
- -file ordered map
- -page ordered map

Approximately 150 pages of output are produced.

USE: For use with the current system,

```
macrolib file:cmdmaclib
packmap
```

This program reads input from GUSER, prints error messages on SERCOM, prints output on SPRINT, and accepts a PAR= field which must be either "HEX" or "DEC".

Input currently consists of public volume names only. An end-of-file terminates execution. The PAR= field defaults to "HEX" and it determines whether output is in hex or decimal. Items which are always hex or decimal are marked (HEX) or (DEC) in the listing.

The relocation entries are listed giving the old relative page number followed by the new (relocated) relative page number. The actual old disk address and the new disk address follow in parentheses. Disk addresses consist of cylinder/head/record.

File Ordered Listing

The file ordered listing is an alphabetical listing of all files on the pack. Each entry consists of a filename, followed by the relative DSCB type-E page location, followed by the relative DSCB type-E disk location in parentheses, followed by the data page range associated with the DSCB expressed in relative page numbers, followed by the data page range expressed as disk addresses in parentheses, followed by the first relative DSCB type-F page location if there are more than 38 data pages in the file, and so on.

For example,

```
*ACCOUNTING1 6060009c(2/E/1) 11D-142(5/0/1-5/C/3)

  7060009C(2/E/1) 143-158(5/C/5-6/0/5)
```

...is interpreted to mean that the DSCB type-E for ACCOUNTING1 is located at relative page number 9C on public volume number 6. In fact it is the seventh DSCB in that page -- there are 16 DSCB slots in every DSCB page. The actual disk address of page 9C is cylinder 2, head E, record 1. Note that the three pages on each track are actually record numbers 1, 3, and 5. The data page range for ACCOUNTING1 defined by this DSCB type-E is relative page numbers 11D through 142 (or cylinder 5, head 0, record 1 through cylinder 5, head C, record 3). The first (and only in this case) DSCB type-F for ACCOUNTING1 is also located at relative page number 9C on public volume number 1, and so on.

Page Ordered Listing

Each entry for the page ordered listing takes one of two forms. For data pages, the entry consists of the data page range expressed in relative page numbers, followed by the data page range expressed as disk addresses in parentheses, followed by the filename. For DSCB pages, the entry consists of the page range expressed in relative page numbers, followed by the page range expressed as disk addresses in parentheses, followed by the string "DSCB".

TABLMOD - Shared File Table Utility

PURPOSE: UMMPS job program to display and modify the in-core open file table. If signons or signoffs become impossible, one should do an "LSTAT FILE filename" to see if someone has one of the accounting files or *STATISTICS locked.

USE: At the operators' console:

```
tablmod
```

From an MTS task:

```
macrolib file:cmdmaclib
tablmod
```

Commands are read until an end-of-file is entered. The following commands are available:

VERIFY verifies (entry and element) allocations are consistent by chasing through various chains in shared file table. Also prints # open files, # matrix computations, # deadlocks detected.

TRACE prints the entire shared file table. Each entry consists of a filename, job number, open status ("OPEN" or "NOTO", possibly "INVLD"), lock status ("LOCKR", "LOCKM", or "LOCKD"), and wait status ("WAITO", "WAITR", "WAITM", or "WAITD").

LSTAT prints the information associated with a single file ("LSTAT FILE filename") or a single job ("LSTAT JOB nn") in the same format as the DUMP command.

LOCATE prints the halfword offset into the in-core table for the entry associated with a particular file ("LOCATE FILE filename").

CLEAN takes the specified job ("CLEAN JOB nn") off all open or locked and waiting lists.

CLEANF same as CLEAN except for one file only ("CLEANF JOB nn FILE filename").

DEQ takes the specified job ("DEQ JOB nn") off all waiting lists.

DEQF same as "DEQ" except for one file only ("DEQF JOB nn FILE filename").

TABLMOD - Shared File Table Utility

HOGL lists the top three jobs using the InCore File table.

Validate - Validate Files

PURPOSE: To validate line and sequential files. This program is used to determine whether a file has been damaged as a result of a hardware or software failure. VALIDATEFILE is normally used to determine whether a file has been damaged by a software failure. It also has features which are used during disk disaster recovery, however, and it is this particular usage which is described here.

USE: $RUN *VALIDATEFILE {PAR=options}

Input is read from GUSER, error output is produced on SERCOM, and bulk output from the DUMP or FLAG options (see below) is produced on SPRINT. The PAR field keyword options may be separated by blanks or commas. The keywords are:

filename - This is the name of the file to be validated. If present, this must be the first parameter. If a file name is given, no input is read from GUSER, and when validation of this file is complete, the program terminates.

DUMP - The contents of the file's line directory and hole directory is dumped in a formatted fashion. Each line of the dump describes an item in the file--a data line, the line number table line, or a hole--which consumes space in the data portion of the file. The lines contain: 1) "HOLE" if the item is a hole, or "LNT" if the item is a part of file's line number table, or the line number of the data line which this item is a part of; 2) the location of the item in the line directory pages of the file, in the form of a relative page number and offset within the page; 3) the segment number of the item in case it is part of a data line too long to entirely fit within one file page and therefore is segmented into smaller chunks; 4) the amount of space in the data page which the item describes; 5) the location of the designated space, in the form of a relative page number and offset within the page; 6) the offset within the page where the next item should reserve space for; 7) comments which describe what (if anything) is improper about this item.

FLAG - This option causes only the items which are improper to be dumped, in the format described above.

FIX - This option is only valid for the user ID FILE. It causes VALIDATEFILE to call MTS when it encounters an invalid directory page header. Occasionally it

is desirable to manually repair this header in VM to continue with the validation of the file. SDS or the MTS $MODIFY command may be used to do this, then the program can be restarted to continue validation.

ZEROCHECK - This option causes VALIDATEFILE to read in every data page occupied by a data line and check the page for being zeroed. The disaster recovery procedures begin by zeroing pages which are damaged, and these pages may belong to files. The pages may, however, belong to a file but may not currently be in use by the file and so the file would not require restoration. VALIDATEFILE provides this feature to isolate this case.

Any of the above options may be given as the right hand side to the OPTIONS= keyword. If more than one is desired, the right hand side may be a parenthesized list of options. If the OPTIONS= keyword appears in the PAR field without a file name, this defines the default values for those options for the duration of the VALIDATEFILE run.

VALIDATEFILE establishes the default options from the PAR field. If a file was specified there, the file is validated and the program stops. If a file name was not given in the PAR field, a file name and options are prompted for on GUSER. A file name and options (as described above) may be given, and the file is validated according to the specified options. An end-of-file stops the program.

RETURN CODES:

0 => File was checked and is consistent
4 => File was checked and is inconsistent
8 => File could not be checked (no access, nonexistent, ...)

EXAMPLES:

```
$RUN *VALIDATEFILE GUSER=filelist -
 PAR=OPTIONS=ZEROCHECK

$RUN *VALIDATEFILE PAR=*ACCOUNTING FLAG

$RUN *VALIDATEFILE SPRINT=dump PAR=OPTIONS=DUMP
badfile1
badfile2 FIX
end-of-file
```

VAMREC - Error Recovery Program

VAMREC is a utility program which allows the user to attempt error recovery after disk errors. It is similar in function to DISKMOD except that it is easier to use (perhaps a bad thing?) and it expects the disk format to be VAMX. The program reads commands from GUSER, puts prompts and error messages on SERCOM, and uses SPRINT for some of its output.

The program uses the following prompts:

1. VOLUME:
2. VOLUME NOT FOUND, RESPECIFY OR GIVE DEVICE NAME:
3. ERROR ON LABEL READ, GIVE PAT PAGE NUMBER:
4. ACTION:
5. WRITE OPERATION. CONFIRM:
6. CMD:

When prompts 4 or 6 are issued the response of "?" or HELP will print out a list of allowed command verbs.

For 1 or 2 a six character volume name or a four character device name is required. Either may be followed by the option "NOPAT", in which case no attempt is made to read or validate the PAT (useful for packs from foreign systems). An end-of-file terminates execution.

Specification of a drive by device address rather than volume label bypasses the label check on the pack.

A decimal (or hexadecimal number in quotes) page number is the expected response to 3. An end of file results in prompt 1.

Prompt 4 is caused by the occurence of an error while reading the PAT. The legal responses are:

STOP --- execution terminated
CHECK --- check the current PAT page for illegal characters
MTS --- return to MTS
FINISH --- terminate reading of the PAT and use what has been read up to this point as the PAT contents.
IGNORE --- pretend there was no error and continue to read PAT pages
COMMAND --- don't read any more PAT pages, give CMD: prompt.
$mtscommand --- the MTS command is executed.

An end of file results in prompt 1.

Prompt 5 is given when a command needs to do a write operation for its completion. Positive responses are: OK, YES, and !. check all your previous work before you give a positive response!!!

VAMREC will harass you if it thinks you may be doing something wrong. If messages like "CAN'T;..." or "WARNING:..." appear, make sure you are confident of what you are doing before proceeding.

Command mode is indicated by the prompt CMD:. Optional portions of the command prototypes are given in braces ({ }). Just about any place where a decimal number can be given, a hex number may be provided instead by enclosing the hex value in primes, e.g. 'FF' is decimal 255. The commands are:

ADD

This command makes the volume available to users. It results in a call to VOLREL with an "add" code.

ASSUME ON
ASSUME OFF

If ON is given, no checking is done to see if a page in a READ or WRITE command really corresponds to a valid page on the device byte for the page or the track address). If OFF is give I the checking is done (as is initially the case).

CALL routinename text

The CALL command invokes a subroutine previously loaded by the LOAD command. The subroutine is called with an OS S-type calling sequence, with the following parameters:
(1) the address of the in-core buffer continaing the last record read from disk
(2) the fullword length of the record
(3) the address of the text following the routine name on the CALL command
(4) the fullword length of the text.

COUNT BLOCK {NOT} values
COUNT LABEL {NOT} values
COUNT PAGE {NOT} values
COUNT PAT {NOT} values
COUNT RECORD {NOT} values

Counts occurrences of the specified "values" in the specified area. One or more of the following values may be specified, separated by blanks: DSCB, PAT, END, DATA, FREE, ERROR, 'x', or "c" where x is a one-byte hex value and c is one character. Use of the NOT operand will find the set complement of the values specified.
Example: COUNT PAT NOT FREE will provide a page usage profile.

DASDI - see FORMAT

DISPLAY LABEL
DISPLAY {SHORT} x y

DISPLAY {SHORT} PAT x y
DISPLAY RELOCATION

The SHORT operand, if given, causes the display output to be formatted with at most 16 hexadecimal bytes displayed per line. Otherwise, 32 hex bytes maximum will be displayed.
The LABEL parameter results in the volume label displayed in EBCDIC and in hexadecimal.
"X" is the displacement into the designated area (the buffer containing the current record's contents, or the PAT as a contiguous chunk) of the beginning of the data to be displayed.
"y" is the number of bytes that is to be displayed and will default to 4 if omitted. The numbers are assumed to be decimal unless there are quotes around them. The display is in hexadecimal.
The RELOCATION parameter causes all the relocation entries on the volume to be printed out in a formatted fashion.

ERRORCHECK {TRACK}
ERRORCHECK PAT
ERRORCHECK HEAD
ERRORCHECK ALL (this does 100% checking)
ERRORCHECK COMPARE
ERRORCHECK RANDOM

TRACK (or just ERRORCHECK) results in a seek to every track and a search for and read of the first full page on every track. In case of a unit check, the sense information plus the page number and seek address are printed on SERCOM.
PAT results in a hexadecimal dump of all relocation entries. The entries are checked for consistency among themselves also (e.g., a bad page should not have itself as the relocating page, etc.).
HEAD causes a read of every page accessed with the specified head number to be read. This is useful for checking out a surface on a disk if it is suspect. If an error results, the sense information and page number are displayed.
COMPARE results in a seek to and read of the page corresponding to the "bad" address of each relocation entry. A line is printed for each such page.
ALL results in reading each and every page on the pack (as long as the pat byte for the corresponding page does not indicate it to be an error page). This action is followed by the action one would get with the COMPARE variant of ERRORCHECK.
RANDOM results in reading each and every page on the pack (as long as the PAT byte for the corresponding page does not indicate it to be an error page). Pages are read randomly. When all pages have been read (they are read only once per pass), the message PASS FINISHED is printed and the next pass begins.
RPS will be used for seeking if it can be.

FIND BLOCK {NOT} values
FIND LABEL {NOT} values
FIND PAGE {NOT} values
FIND PAT {NOT} values
FIND RECORD {NOT} values

Scans the specified area for the values requested. One or more of the following values may be specified, separated by blanks: DSCB, PAT, END, DATA, FREE, ERROR, 'x', or "c" where x is a one-byte hex value and c is one character. Use of the NOT operand will find the set complement of the values specified.
Example: FIND PAT ERROR PAT END finds the overhead bytes in the PAT.

FORGET

This command instructs the system to forget where the given volume is located so that the next non-VAMREC reference to it will cause the volume's label and PAT to be re-read. It is useful if any relocation entries have been generated by VAMREC to force the system to become cognizant of them.

FORMAT CYLINDER cc
FORMAT GROUP x
FORMAT TRACK cc hh
DASDI CYLINDER cc
DASDI GROUP x
DASDI TRACK cc hh
FORMAT RETRY cc hh

The FORMAT ASDI commands issue formatting write commands (write count-key-and-data) on a given track, a group of tracks (as defined by the device type--groups are record collections which occupy integral numbers of tracks), or a full cylinder. The difference between a DASDI operation and a FORMAT one is that DASDI will rewrite the home address on the track and record zero, whereas FORMAT will not. To be formattable, the affected records must be marked as either free, allocated, or error pages in the PAT. If marked free, a warning will be issued.
The FORMAT RETRY option specifies a track on which a special record will be written that will be read by the READ RETRY command (2305, 3330 only). The PAT bytes for all pages on such a track must be X'C1'.

FREE

This command cancels the effect of a KEEP and will free the device/volume.

INITIALIZE

This command changes all data and DSCB entries in the PAT to available entries. It zeroes the relocation entry count and flag. It then rewrites all the PAT pages. A data entry in the PAT is defined as a byte which does not have bits 0 and 1 on simultaneously (i.e., B'11XXXXXX') nor does it have bits 1 thru 7 on simultaneously (i.e., B'X1111111'). This excludes any PAT-page bytes and any kind of error pages.

KEEP

This command causes the volume evice given in response to the VOLUME query to be acquired, its PAT to be read, and a bit is set so that it will not be freed as is normally done between commands Therefore, at the end of the execution of this command, VAMREC's knowledge and control of the disk volume is complete and up-to-date.

KEYS ON
KEYS OFF

This command affects the operation of record-oriented READ or WRITE commands. If KEYS ON has been given, the channel commands issued read and write the key field of the record. The default, KEYS OFF, cause only the data portion of the records to be operated upon. The key field is not touched by commands which are page-oriented.

LABEL
RELABEL

The label is rewritten with the contents of the label buffer.

LOAD routine fdname

The LOAD command loads a subroutine from the FDname specified. This subroutine may subsequently invoked by using the CALL command (described above) to manipulate a disk record in memory, or perform some other function.

MODIFY LABEL x c
MODIFY x c
MODIFY PAT x c

"x" is the displacement in the specified buffer (neither LABEL nor PAT specified indicates the current page). "x" is assumed to be decimal unless enclosed by quotes. "c" is the character string to be placed at "x". It must be enclosed by quote-marks (for a hex string) or double-quotes (for an EBCDIC string --- successive double-quotes within the EBCDIC string denote a single double-quote). The length of the string (if hex) must be even.

MTS
A return to MTS.

PAGE c h r

The relative page number of record "r" on track (head) "h" of cylinder "c" is printed, if the record is a page.

PRINT {SHORT}
PRINT {SHORT} DSCB
PRINT {SHORT} PAT

The SHORT parameter is optional and behaves as with the DISPLAY command.
With no parameters this command results in a hexadecimal and EBCDIC dump of the current page on SPRINT. The parameter DSCB results in a dump of all DSCB pages on the pack (many pages of output). The PAT parameter results in a dump of the PAT pages.

READ page#
READ BLOCK block#
READ BUFFEREDLOG
READ HOMEADDRESS (or HA) {c h}
READ PAGE page#
READ RECORD c h r
READ RETRY c h

This command reads the record specified by the parameters. "page#" is a page number and "block#" a block number for a fixed-block device (quotes for hexadecimal), "c" is the cylinder number, "h" is the head number, and "r" is the record number.
When RECORD is specified, the seek address must specify a legal page number, but the check for a legal page may be overridden by the ASSUME command (see above). The amount of data read defaults to 4096 bytes, but may be changed by the IOLEN command (see above, also).
When RETRY is specified, record R1 is "read" by a SPACE-COUNT command to force the control unit into command retry (correctable error in the key field). The record so read is the one written by the "FORMAT RETRY c h" command. The PAT bytes corresponding to all pages on such a track must be X'C1'.
The BUFFEREDLOG operand allows the buffered log (2305, 3330, 3350) to be read and displayed (note that this resets the log).
The BLOCK operand is only valid for 3370 or other fixed-block devices.

RECORD n
The cylinder, head, record address for relative page "n" is printed.

RELOCATE
RELOCATE PAT

If no parameter is specified, the current page (the page last processed by a READ command) is relocated. If the PAT byte for this page is B'11XXXXXX' indicating it is bad, or if the PAT byte for this page is B'00000000' indicating that the page is available, the PAT byte is merely set to B'11000000' (X'C0') and the PAT is rewritten. Otherwise, the PAT byte is flagged as bad (X'C0'), a free page is found in the PAT, it is flagged with the same PAT byte as the old page, a relocation entry is added to the PAT, the contents of the record buffer are written to the new page address, and the PAT is rewritten. If the parameter PAT is specified, the PAT is relocated. The old PAT pages are flagged as not available (i.e., X'C0') and a new block of free space is located. The first PAT page address is set in the label. The label is rewritten and then the PAT pages are rewritten. No additional relocation entries are generated.
Remember to flush the Disk Manager Cache before doing this.

REMOVE
The command which makes the volume unavailable to other users. It results in a call to VOLREL with a "remove" code.

STOP
The normal way to terminate execution.

VERIFY
VERIFY PAT
VERIFY DSCB

The PAT parameter or no parameter causes all bytes in the PAT buffer to be checked for illegal characters (this check is also done whenever the PAT is rewritten.) If an illegal character is found, its location in the PAT is displayed along with a query as to whether to continue or not. OK, YES, or ! will cause continuation of the verification, anything else will not. The DSCB parameter causes all DSCB pages to be read and all the checksums of the DSCB's to be verified. If incorrect, the expected checksum value will be printed also.

WRITE
WRITE {PAGE} x
WRITE PAT
Write LABEL

If no parameter is given, this command causes the contents of the record buffer to be written back to the same address as the previous READ command. If the buffer has been implicitly changed, or the page address has, an error message or warning will be issued.

If the PAT parameter is given, the PAT pages are re-written from the PAT buffer.
If the PAGE operand is given, the contents of the record buffer are written at the given page. Thus to move pages, one would do a "READ x" followed by a "WRITE PAGE y".
The LABEL parameter causes the label to be rewritten to the disk from the contents of the label buffer.

ZAP x c n
ZAP PAT x c n

The PAT parameter results in the PAT buffer being modified; else the current page is modified. The "x" is the displacement in the buffer, the "c" is the one byte fill character, and the "n" is the number of times the character is to be placed in the buffer.

Some examples of the commands.

```
DISPLAY '100' 10
DISPLAY 256 'A'
DISPLAY LABEL
DISPLAY PAT '1000' '30'
MODIFY PAT '2000' '7F7F7F7F'
MODIFY LABEL 4 "TMTS02"
ZAP PAT 50 '41' 100
FORMAT GROUP 2
READ BUFFEREDLOG
KEEP
FORMAT RETRY '20' 0
READ RETRY '20' 0
```

Note: HELP or "?" gives a list of valid commands. A command starting with a "$" is passed on to MTS. All commands can be abbreviated to the minimum number of characters required to distinguish them. The order of recognition is as printed by the HELP command.

Before any relocation is done, the KEEP command should be used. The FORGET--FREE command combination then results in MTS discovering the new state of the pack at the next non-VAMREC access of the pack.

VAMREC uses the resident unit check routines, except when reading the funny record with READ RETRY. All write operations are read checked.

VNTD - Catalog Utility

PURPOSE: To verify, trace, and/or dump the file system catalog. This program will verify, trace, and/or dump the entire catalog, a particular user catalog, or a particular user file.

USE:
```
macrolib file:cmdmaclib
vntd
```

The program accepts input on SCARDS if no PAR= field is given. An end-of-file terminates the program. SPRINT and SERCOM are used for output.

Parameters must be separated by blanks or commas and must be given in the following order:

(1.) any, all, or none of the following:
"V" - verify
"T" - trace
"D" - dump
"L" - use STDDMP format for dump output

(2.) at most, one of the following:
"C" - the entire catalog (the default)
"U" - a particular user catalog
"F" - a particular user file

if "U" is given, then a legal MTS userid must be the next parameter. if "F" is given, then a legal internal filename must be the next parameter. if "...,U,*ALL" is entered then all user catalogs are processed. if "...,F,*ALL,ID" is entered then all files corresponding to the specified user ID are processed.

EXAMPLES:
```
$Run FILE:VNTD+Copy:SYSDEFS PROT=OFF
          PAR=T,C
$Run file:vntd+copy:sysdefs prot=off
          par=v,t,d,l,u,w045
$Run file:vntd+copy:sysdefs prot=off
          par=d,f,w045fyle
```

OUTPUT: VERIFY

Verifying the entire catalog will validate segment allocation as well as error checking record and segment headers. Presently the catalog can only be verified when

segments are not being allocated or deallocated, ie: when no one else is using the system. Verifying the entire catalog will take >30 minutes.

Verifying a user catalog will error check record and segment headers as well as file descriptors. In addition, it will check that file descriptors point to sharing descriptors in the same catalog and that all sharing descriptors are accounted for.

Verifying a file checks file descriptors & sharing information for reasonableness.

TRACE

Tracing the catalog will print out the file header locations and the number of pages in each file.

Tracing a user catalog will print out the segment locations for each segment assigned to the user catalog.

Tracing a file will print out file and sharing descriptor locations.

DUMP

Dumping the catalog will dump (via SDUMPor STDDMP - as selected by the "L" parameter) each file header.

Dumping a user catalog will dump each segment assigned to the user catalog.

Dumping a file will dump the file descriptors and sharing descriptors associated with the file in the catalog.

VTOCUTIL - VTOC Utility

Purpose: A general utility for examination, verification and fixing of the label/PAT/DSCB structure of a disk volume.

Availability: Use the VTOCUTIL macro in file:cmdmaclib. It takes as parameters:

tables= alternate UMMPS tables - defaults to system tables

frtns= alternate file routines (without tables) - defaults to system tables if tables= wasn't specified, otherwise to file:filertns, the current file routines.

scards= command input - defaults to *source*

If any extra parameters are given, then VTOCUTIL is run in a "one-shot" mode, with the extra parameters passed as the only command.

Commands are:

MTS - goes to MTS command mode

MCmd <command> or $<command> - execute MTS command

STop - stops

Help - for help

TRACE {ON | OFF | FULL} - trace the command parsing

Files <filenames> (<foptions>) - print info about files

Pages <pagenumbers> (<poptions>) - print info about pages

BIGger (than) <int> (pages) (ON <fdname>) - print names of files which are bigger than <int> pages

DISContinuities - Print information about discontiguous pages within files

SUMMARY {ALL | <volname>} - Print usage summary for specified volume or all volumes

ALL <options> - process all volumes

<volname> <options> - process named volume

where <options> are:

(NO)FINDdscbs - to pattern match for DSCB pages (default NO)

(NO)PATTERNmatch - synonym for FINDdscbs

(NO)CHECK - to check VTOC consistency (default CHECK)

(NO)FIX - to fix some of the errors found (default NOFIX)

<filenames> can be:

one or more names separated by blanks, in id:name form or in "internal" form.

LIST=<fdname> where <fdname> specifies a file containing one or more filenames, one per line.

<foptions> are:

ON <fdname> to write the output on <fdname>.

WITH Pages to print the page numbers of the pages in the files.

<pagenumbers> are separated by blanks and can be in hex by enclosing in primes (eg '0010'). They can also be in cylinder/head/record form by separating with slashes (eg 16/5/3 or '10'/'5'/'3' or '10/5/3'). Also ranges are accepted by separating with a dash (eg 16/5/3-3000).

<poptions> are:

ON <fdname> to write the filename and its DSCB-E on <fdname>.

EXTERNAL to have the DSCB-E written (via ON <fdname>) in hexadecimal characters, instead of internal form.

The PAGES, FILES, and BIGGER commands apply to the volume or volumes last specified in a <volname> or ALL command. super

Index