Published: 03/04/67
Supersedes: BG.2, 05/27/66

Identification

The System Segment Tables
R.C. Daley, D.M. Ritchie

Purpose

The system segment tables are used by segment control and page control in maintaining active segments. Entries are established in the tables by segment control when segments become active and loaded as a result of a user reference. The tables are used by page control in manipulating pages for loaded segments.

Introduction

The system segment tables consist of three distinct subtables: the active segment table (AST), the descriptor segment table (DST), and the process segment table (PST). These tables are per-system tables shared by all processes running under the same version of Multics and are manipulated only by the segment control and page control modules.

The active segment table (AST) contains an entry for each non-descriptor segment which is currently active. These entries are created in the AST by segment control whenever an inactive segment is referenced by a running process; at most one entry appears for one segment, no matter how many processes refer to it. Once the corresponding AST entry has been created the segment is said to be active. Each AST entry indicates whether the corresponding segment is loaded (page table in core) or unloaded (page table not in core). Normally, a segment is loaded by segment control whenever the segment becomes active. The AST entry for a loaded segment is used by page control in determining what action to take on a missing-page fault.

Page control may, due to inactivity, cause a loaded segment to become unloaded. When the AST becomes full, segment control may decide to remove an AST entry for a currently active but unloaded segment to make room for a new AST entry.

The descriptor segment table (DST) contains an entry for each descriptor segment which is currently loaded. These

entries are created by segment control whenever an unloaded descriptor segment is needed by a running process. Once a new DST entry is created, the corresponding descriptor segment is provided with a page table and the segment is loaded.

When, due to inactivity, the last page of a descriptor segment is removed, page control deletes the corresponding DST entry, thus rendering the segment unloaded. Note that the concepts of active and inactive do not apply to descriptor segments.

The process segment table (PST) contains an entry for each process which is currently active (see BG.3.03 and BJ.2.01). When a process becomes active, segment control creates an entry for that process in the PST. This entry is deleted by segment control when the corresponding process becomes inactive. During the time that the process is active, the PST entry for that process contains the unique identifiers of segments which require special consideration. For example, the Known Segment Table (KST; see BG.1) of a process must be active if the process is active. Thus the PST entry for a process contains the unique ID of the KST of that process. The main function of the PST is to allow quick access to all segments within a particular process which require special treatment. Each entry in the PST is reached by a relative pointer in the Process Data Segment for that process.

Locks

Because the SST is a system-wide data base, accessible to all processes simultaneously, it must be lockable; that is, it must be possible to prevent one process from modifying an entry that another process is reading. The mechanism for ensuring that this is possible is discussed in detail in section BG.15. A summary here of the most important points may be worth while.

A lock consists of a word associated with a data set. This word is zero if the set is unlocked; in this case any process may read the data set with assurance that a stable copy is being read. If the lock is nonzero, then it contains a process number in whose behalf the data set is locked. This process may use the data as it wished, but no other process may use the data in any way until the lock is removed.

There are actually two types of locks, block locks and loop locks. For the former type, processes wishing to use the data set are blocked if the set is unavailable whether because it is already locked or another process is reading the data. They are awakened automatically when the lock is removed or when all readers of the data have disappeared so that the lock can be set.

Because waiting on block locks may deprive a process of a processor for a relatively long time, loop locks are used in cases where this is undesirable, for example during a page fault. Processes attempting to set a loop-type lock enter a tight loop testing the lock word; as soon as the lock becomes zero, it is set in favor of the new process. While a loop lock is set in behalf of a process, the processor is masked against interrupts that might take the processor away from the process for a long time - for example, timer runout. Since a processor waiting on a loop lock is serving no useful purpose, it is important that loop locks be set only for a short time. For each of the locks mentioned below, information is given as to which type of lock it is.

Contents of the AST

The AST contains a header and AST entries. The header is at the beginning of the AST and appears only once.

For each non-descriptor segment which is currently active within the system there is an AST entry. An AST entry contains the following information. (The description of the header follows that of the AST entry description since it refers to some of the AST entry items.) All objects called "pointers" in AST entries, AST process trailers and AFT trailers (described below) are relative pointers within the segment containing the AST. By convention, such pointers are null if they are zero.

1. AST entry interlock (block)
2. Unstable page control values lock (loop)
3. Entry hold count
4. Page table hold count
5. Wired-down segment count
6. Unique identifier of segment
7. Maximum segment length
8. Global transparent-usage switch
9. Large page switch
10. Number of inferior AST entries

11. Pointer to AST entry for parent directory segment
12. Branch index in parent directory
13. Date and time branch was last modified
14. Hyperpage size
15. Page batching size
16. Pointer to page table (absolute core address)
17. Removal list switch
18. Forward removal list pointer
19. Backward removal list pointer
20. Process trailer pointer
21. Active meter table pointer
22. AFT pointer for original file
23. AFT pointer for move file
24. Number of outstanding I/O requests
25. Number of hyperpages in core
26. Current segment length
27. Segment loaded switch
28. Date and time last used
29. Date and time last modified
30. Activity indicator
31. Time activity indicator was last modified

1. AST entry interlock (block)

This interlock is used to restrict access to items 1 - 21 of an AST entry to only one process. In particular, this segment cannot be deactivated or unloaded while this lock is set. However, page faults can be serviced.

2. Unstable page control values lock (loop)

This lock is set when page control is modifying one of items 22 - 31 of an AST entry, for example during a page fault. This lock is independent of item 1.

3. Entry hold count

This is a count of processes which need this segment in an active state in order to remain active; an example of such a segment is the KST for some process. If this item is non-zero, the segment will not be deactivated even if the other criteria are met. (See item 17.)

4. Page table hold count

In a manner similar to the last item this item is a count of processes which require the page table for this segment to remain in core in order to remain loaded processes. The segment will not be unloaded even if the number of pages in core drops to zero if this item is non-zero.

5. Wired down segment count

In a manner similar to the last two items this item is a count of processes which require all pages of this segment currently in core to remain in core. Examples would be the Process Data Segment for each process and special segments that the process has requested. No pages of this segment will be removed from core until this count is zero. (An example of a special segment might be a module of the hard-core supervisor for which the process needs a version different from the usual one.)

6. Unique identifier for this segment

This item uniquely identifies the segment among all that exist or have existed in any MULTICS system. When segment control wants to determine if a given segment is active, it looks for an AST entry with a matching unique identifier. To speed this searching, a hash-coded table-lookup technique is used. The unique identifier is the item which is hashed.

7. Maximum segment length (units: 1024 words)

The item is used by page control; for example, when a page table is being constructed, it is used in the selection of page size.

8. Global transparent-usage switch

This switch is the logical AND of all transparent-usage switches in the KST entries for the segment described by this AST entry. If any one of the processes using the segment have the transparent-usage switch OFF, this switch will be OFF. The setting of this switch is interrogated by page control whenever an I/O request is made. If the switch is OFF, the time-last-used or the time-last-modified will be updated.

9. Large-page switch

If the segment is currently paged in 1024-word pages, this switch is ON. Note that if the segment is larger than 16K words, the switch must be ON, since this is the largest segment of small pages possible.

10. Number of AST entries directly inferior to this (directory) segment.

When an entry is added to the AST, there already exists

an AST entry for the segment which is immediately superior to the new segment. This count is incremented by one in the AST entry for the immediately superior directory segment. Whenever a directly inferior AST entry is removed, this count is decremented by one. An AST entry is a candidate for removal whenever this count is zero, the entry-hold count and the page table hold count are zero, and the page table for the segment has been removed from core.

11. Pointer to AST entry for directory segment containing the branch for the segment

At various times--for example when informing Directory Control of updated information to be placed in the branch of a segment being deactivated--Segment Control must be able to locate the branch for the segment even though the process doing the deactivating may not know the directory segment in which it appears. This item locates the AST entry for the directory containing the branch corresponding to this segment. Also, this item may be used to update the inferior segment count for the superior directory when this entry is being deactivated.

12. Slot number in directory of branch of the segment

This item and the last form a complete pointer to the branch defining the segment.

13. Date and time branch was last modified.

This item is copy of the information contained in the directory branch of the segment. It is used to make sure that the access rights and protection list of the segment are properly recorded in the KSTs of all processes using the segment. If the time entry for the segment in the KST of a process is the same as this item, the access bits in the KST are correct; if the times differ, the access bits and protection list must be recomputed because the branch information may have been changed.

14. Hyper-page size

A hyperpage is a contiguous collection of pages which is treated as a unit by the file system--for example, a hyperpage is always read and written as a block. (See section BG.5.) This item is the number of pages in a

hyperpage; the size of the hyperpage in words is the product of this item and the current page size.

15. Page batching size

This item makes provision for a strategy of bringing in from secondary storage a batch of several consecutive hyperpages when a fault occurs on a page in one of them. The process waiting for the missing page is unblocked as soon as the page it faulted on arrives, and need not wait for the entire batch. This item will be unity in the initial implementation.

16. Pointer to page table (upper 18 bits of a 0 mod 64 absolute core address)

This item points to the page table for the segment. It is provided by page control when the segment is loaded and is used by page control to access the page table during page faults. This pointer has meaning only when the segment-loaded switch (item 27) is ON.

17. Removal-list switch

When certain criteria are met, an entry in the AST may be considered for removal. These criteria are: the segment is unloaded (item 27 is OFF), the AST entry-hold count (item 3) is zero, the number of active inferior segments (for a directory segment) is zero (item 10), and finally, the AST entry interlock (item 1) is not set. Candidates for removal have this item ON, and are linked in a forward and backward threaded list; they are not actually excised until space is needed in the AST.

18. Forward removal-list pointer

19. Backward removal-list pointer

The candidates for removal (marked by having the removal-list switch, item 17, ON) are kept in a doubly threaded list. Their hash-table pointers (see below) are not removed until they are actually deleted. Thus if it is desired to reactivate an entry on this list, the information therein need not be regenerated and the threads allow the removal list to be repaired easily after the newly reactivated entry has been taken off the list.

20. Pointer to process trailers

This item, if non-zero, points to the first of a threaded list of process trailers for this segment, which may exist whenever the segment-loaded switch (item 27) is ON. These trailers exist for the purpose of finding the segment descriptor words for this segment (SDWs) in each process in which the segment is active. Thus, when a segment is unloaded, page control finds all the SDWs for the segment and marks them with a segment fault. Similarly, when the access rights to a segment change all the SDWs are filled with segment faults. When a fault occurs the new access rights are computed for the process causing the fault. The process trailer for a segment is removed when that process terminates the segment; and when a process is deactivated, the process trailers for that process are removed from all AST entries of segments known to that process. On the other hand, all process trailers for a segment are deleted when the segment is unloaded. Each process trailer contains the following information.

20.1 Segment number

This is the number by which this process knows this segment; knowing this number and the pointer to the PST entry, the segment descriptor words for this segment in this process can be found.

20.2 Pointer to PST entry

From this item the process to which this trailer refers can be recovered, so that the SDWs for the segment may be accessed.

20.3 Forward AST trailer pointer

20.4 Backward AST trailer pointer

All the trailers for each AST entry are threaded in two directions to allow these trailers to be removed easily when a process terminates the segment; for when this occurs, a single trailer on the affected AST entry must be removed. The double threads allow the list of trailers for each AST entry to be repaired easily.

20.5 Forward PST trailer pointer

20.6 Backward PST trailer pointer

The PST (see below) for each process contains the end of a thread through all process trailers for active segments known to that process, so that all these trailers can be removed when the process is deactivated. The list is threaded in two directions so it can be patched together when a single segment becomes inactive. Thus each AST process trailer has two pairs of threads running through it; one pair links all trailers for a single AST entry, the other links all trailers for a single process.

20.7 Pointer to AST entry

When the process trailers on all segments for a process are removed, it is necessary to be able to find the AST entry for each such segment. That is, this item is used when the trailer is accessed via the PST entry rather than via the AST entry.

21. Active Meter Table Pointer

This item points to an entry for this segment in the Active Meter Table, a wired-down data base maintained by the accounting modules. The (relative) pointer is supplied by accounting when the segment is activated and is used when informing accounting of changes in the status of a segment: activation, deactivation, changes in length, and movement to a new device.

22. Pointer to trailer for (original) file

Besides process trailers for each process using a segment, the AST entry for a segment has at least one active file trailer (AFT) containing information on the location in secondary storage of the file corresponding to the segment. This information is kept for the benefit of the DIM responsible for the file. Each such trailer contains

22.1 AFT interlock

This lock is used by the DIM, for example to prevent interference between processes attempting to change the file length (item 22.2). This lock is neither a loop lock or a block lock; if it is set the DIM finds something else to do rather than waiting.

22.2 File length (units: 64 words)

This item identifies the actual length of the file as recorded on the DIM's device.

22.3 Device Interface Module Identifier

This item contains a number identifying the DIM responsible for the file; requests for I/O of pages in this segment are directed to this DIM.

22.4 Secondary storage address of file map

The file map of a file in secondary storage is a table, stored on the same device as the file, and similar in concept to a page table for a segment in core; it contains the sector address of each 64-word sector of a file. Thus, the sectors of a file, while logically in contiguous locations on the device, may physically be scattered about in widely separated areas. This item points to the secondary storage address of this file map.

22.5 `History information is being saved' count

The DIM may maintain some sectors of the filemap of a particularly high-activity file in core storage so as to eliminate multiple references to the device for sectors of a file. This strategy is comparable to the use of associative registers in appending. This item is a count of the number of filemap sectors for this file being maintained in core in the DIM history table (see BG.10).

22.6 DIM history table pointer

Whenever item 22.5 is non-zero, this item points to a filemap sector in the DIM history table. This sector in turn may be linked to other sectors of the filemap to a depth recorded in item 22.5. This item and the last are maintained for and by the DIM.

22.7 File-being-lengthened switch

This switch is set by the DIM while it is changing the length of the file on the device.

23. Pointer to trailer for move file

One AFT trailer per segment is the normal case. However, the multi-level system may decide to move a file from one device to another. While this move is going on, there are two AFT trailers for the segment, one for the original file, pointed to by item 22, and one for the move, or new, file, pointed to by this item. Each page table word (PTW) for a segment being moved contains a bit indicating

whether that page has been moved to the new device. When a page is to be read in, the device it comes from is determined by this bit. If a page is to be removed from core, and it has not been moved, it is always written onto the new device and its PTW is marked so the move is remembered. No segment can be unloaded until all its pages have been moved so if an attempt is made to unload a segment with two AFT's such that all its PTW's have not been marked `moved', the attempt will fail and an unmoved page will be read in. This page will ultimately be removed from core due to inactivity and written on the new device, then another attempt will be made to unload the segment, until finally the file has been completely moved.

If item 23 is non-zero, it points to the AFT trailer for move file, that is, the new file.

24. Number of outstanding I/O requests

This count is incremented by page control whenever it ussues an I/O request for a page, and decremented after the request has been completed. No segment can be unloaded until this item is zero..

25. Number of hyperpages in core

Until the number of hyperpages in core becomes zero, the segment cannot be unloaded. In item 27 below all the criteria for unloading are given.

26. Current segment length

This item guides page control in creating a page table for the segment. Page table words for pages beyond the the current length are marked so that an empty block of core will be assigned when they are accessed. Those within the current length are marked so that the proper page will be read in.

27. Segment-loaded switch

This item is ON if the page table is in core. The page table cannot be released until: the number of outstanding I/O requests (item 24) is zero; the number of hyperpages in core (item 25) is zero; the page table hold count (item 4) is zero, and any moving of the file has been completed

(there is only one AFT or all page table words have been marked "moved"). Finally, the AST entry must be unlocked (item 1). This item is maintained by page control.

28. Date and time segment last used (updated when a page is read)

29. Date and time segment last modified (updated when a page is written and the "page modified" switch is set in the PTW)

These two items are maintained by page control, and will stay with the segment when it is deactivated. They are only set if the global transparent-usage switch (item 8) is OFF.

30. Activity indicator

This item is incremented by a constant each time some process issues an I/O request for a page of this segment and it is decremented periodically in time. It thus provides a measure of segment activity and is used by multilevel to help determine if the segment should be moved to another device. (See Section BH.1.)

31. Time activity indicator was modified

This item is used to decay the previous item with time.

Contents of the PST

For each process which is currently active within the system there is a PST entry. Each PST entry may be reached by a pointer in the Process Data Segment (PDS) of the corresponding process (see section BJ.1.03). The PST gathers together conveniently information that would otherwise have to be searched out from the AST and other tables. The "pointers" are relative pointers within the segments in which the pointer-to information lies.

1. PST entry interlock (loop)
2. Unique ID and segment number of KST
3. Unique ID and segment number of PDS
4. Unique ID and segment number of hardcore stack
5. Unique ID and segment number of Process Definitions Segment
6. Pointer to thread of all AST trailers for process
7. Pointer to DST entry for the basic descriptor segment (hard core ring) of this process

8. Pointer to list of special segments for this process
9. Number of special segments.

The items in the PST are interpreted below.

1. PST entry interlock (loop)

The interlock is used to restrict access to the PST entry to only one process at a time. Any process attempting to use a PST entry which is interlocked to some other process must wait until the interlock is removed.

2. Unique ID and segment number of KST

3. Unique ID and segment number of PDS

4. Unique ID and segment number of hardcore stack

5. Unique ID and segment number of Process Definitions Segment

These items are in effect an extension of the HST (Hardcore Segment Table - see BG.1). The segments mentioned, although part of the hardcore supervisor, are different for each process and thus their unique ID's cannot be stored in the HST, which is a system-wide table. Instead the HST entries for these segments contain a pointer to one of these items.

6. Pointer to thread of all AST process trailers for this process

When a process is deactivated (see BG.3.03) this item allows tracking down all the segments associated with it and removing their trailers from the AST entries for the segments.

7. Pointer to DST entry for the basic descriptor segment (hard-core ring) of this process

This item contains the index for the DST entry for the hard-core ring. Other DST entries associated with this process can be accessed by following the next-entry pointers in the DST entries.

8. Pointer to list of special segment pointers

This item points to additional information about the process: a list of pointers to the AST entries of special segments for this process. If there are no special segments, this item is zero. An example of a special segment might be a module of the hard-core supervisor other than the usual one.

9. Number of special segments

Placing this item in the PST entry, rather than in the special segment structure allows easier allocation of space for the special segment list.

Contents of the DST

For each descriptor segment which is currently loaded there is a DST entry. A DST entry contains the following information. Unless stated otherwise, "pointers" are relative pointers within the SST segment.

1. DST entry interlock (block)
2. Unstable page-control-values lock (loop)
3. Ring number of descriptor segment
4. Pointer to page table for segment (absolute core address)
5. Pointer to next DST entry for process (if any)
6. Pointer to previous DST entry for process (if any)
7. Pointer to PST entry
8. Page table hold count
9. Maximum segment length
10. Large page switch
11. Hyperpage size
12. Number of hyperpages in core

The items in the DST are interpreted below.

1. DST entry interlock (loop)

The interlock is used to restrict access to items 1 - 11 of a DST entry to only one process at a time. While this lock is set, the segment cannot be unloaded.

2. Unstable page-control-values lock (loop)

When page control is modifying item 12, this lock is set to prevent interference. This lock is used during a page fault; it is independent of item 1.

3. Ring number corresponding to this entry

Since there is a separate descriptor segment for each ring within a process this item is used to identify which ring is associated with the descriptor segment defined by the DST entry.

4. Pointer to page table for this descriptor segment (upper 18 bits of 0 mod 64 absolute core address).

This item points to the page table for the descriptor segment. It is provided by page control when the segment is loaded and is used by page control; for example, to locate a page table word being filled with a pointer to a page.

5. Pointer to next DST entry for the process

This item points to the DST entry of a loaded descriptor segment for another ring; thus the DST entries for a process are threaded together. If this entry is the last entry in the thread, this pointer is zero.

6. Pointer to previous DST entry for the process

This item points to the previous DST entry in the threaded list for this process. If this entry is the first entry (i.e., is pointed to from the PST entry), this pointer is zero.

7. Pointer to PST entry for this process

This item points to the PST entry for the process. It is used quickly to find the PST entry when deleting DST entries.

8. Page table hold count

This item records the number of processes which require this descriptor segment to be loaded at all times, that is, that need its page table wired into core. Unless this number is zero, the page table cannot be removed (i.e., the segment cannot be unloaded).

9. Maximum segment length

10. Large page switch

11. Hyperpage size

These three items have the same interpretation as do the corresponding items for non-descriptor segments. However, in the initial implementation they will be constant.

12. Number of descriptor segment pages for this ring which are currently in core.

This count is used by page control to determine if the DST entry can be removed. When page control has removed the last page of a descriptor segment from core, it releases the DST entry and calls core control to unassign the descriptor segment page table.

The SST Header

The SST includes a fixed-length leader containing information on the table as a whole. The SST header has the following information.

1. Page table lock (loop)

This item serves as a lock on all the page tables. It is used by page control after a page fault until it can find the proper page to lock individually to prevent loss of the page table.

2. Free storage area lock (loop)

This item, used by the SST allocation routine, locks the unused portion of the SST.

3. AST hash table lock (block)
4. Size of AST hash table
5. Pointer to AST hash table

The AST may be searched by unique identifier. To speed this process a hash table is used. The same algorithm is used as described for the KST in BG.1.

6. AST removal list lock (loop)

This lock is provided so that while this list is being rethreaded after a change no other process can access it in an inconsistent state.

7. Pointer to removal list

This item points to the first item on the removal list.

PL/I Implementation of the SST

SST Header

The PL/I declaration for the SST Header is

```
dcl 1 sst ctl (sstp),
    2 ptl bit  (36),          /* page table lock - loop*/
    2 fsl bit (36),           /* free storage area lock - loop*/
    2 ast,
      3 htlock bit (36),      /* active segment table hash table
                                 lock - block*/
      3 rllock bit (36),      /* ast removal list lock - loop*/
      3 hbc fixed bin (17),   /* size of hash table */
      3 htp bit (18),         /* pointer to hash table */
      3 rlp bit (18),         /* pointer to removal list */
    2 sstvar area ((M));      /* SST allocation area */
```

M is a constant which must be decided upon before compilation.

The AST

The AST uses a hash table so that it can be searched efficiently by unique identifier. The declaration for the hash table is

```
dcl 1 astht (sstp sst.ast.hbc) ctl (asthtp),
      2 vs bit (1),           /* vacant switch */
      2 ds bit (1),           /* deleted switch */
      2 entryp bit (18);      /* pointer to AST entry */
```

vs is ON if the entry is vacant. ds is ON if the entry has been deleted. entryp points to an entry in the AST. The hashing and searching algorithms are the same as those used with the KST (BG.1).

The declaration for AST entries is

```
dcl 1 aste ctl (astep),

      2 lock bit (36),      /* ast entry lock */

      2 upcvl bit (36),     /* unstable page control values
                               lock */

      2 ehc bit (17),       /* entry hold count */

      2 pthc bit (17),      /* page table hold count

      2 wdsc bit (17),      /* wired-down segment count */

      2 id bit (70),        /* unique identifier */

      2 msl bit (9),        /* maximum seg length in units of
                               1024 words */

      2 gtus bit (1),       /* global transparent usage */

      2 lps bit (1),        /* large page switch */

      2 infcnt bit (17),    /* number of inferior AST entries */

      2 astparent bit (18), /* pointer to ast entry for
                               parent */

      2 xbranch bit (17),   /* branch index in parent
                               directory */

      2 dtbm bit (52),      /* date and time branch last
                               modified */

      2 hps bit (8),        /* hyper-page size (number of
                               hardware pages) */

      2 pbs bit (8),        /* page batching size */
```

```
    2 ptp bit (18),      /* pointer to page table */
    2 fastp bit (18),    /* pointer to trailers or forward
                            removal list ptr */
    2 bastp bit (18),    /* backward removal list pointer */
    2 amtp bit (18),     /* active meter table pointer */
    2 rls bit (1),       /* removal list switch */
    2 exfp bit (18),     /* pointer to trailer for original
                            file */
    2 movp bit (18),     /* pointer to trailer for move file */
    2 iocnt bit (8),     /* number of outstanding I/O */
    2 hpn bit (9),       /* number of hyperpages in core */
    2 csl bit (13),      /* current segment length in units of
                            64 words */
    2 sls bit (1),       /* segment loaded switch */
    2 dtu bit (52),      /* date last used */
    2 dtm bit (52),      /* date last modified */
    2 actind bit (36),   /* activity indicator */
    2 actime bit (52);   /* time actind has been modified */
```

Notice that the pointer to process trailers and the forward removal list pointer have been combined since they are never meaningful simultaneously.

The declaration for AST process trailers is

```
  dcl 1 astr ctl (astrp), /* segment or process trailer */
    2 segno bit (18),     /* segment number */
    2 pstep bit (18),     /* pointer to pste */
    2 astep bit (18),     /* pointer to aste */
```

```
    2 fastp bit (18),     /* forward ast trailer pointer */
    2 bastp bit (18),     /* backward ast trailer pointer */
    2 fpstp bit (18),     /* forward pst trailer pointer */
    2 bpstp bit (18);     /* backward pst trailer pointer */
```

The declaration for active file trailers is

```
  dcl 1 aft ctl (aftep),  /* active file trailer */
    2 llock bit (36),     /* lock on file length changes */
    2 filength bit (17)   /* file length */
    2 dimid bit (4),      /* device interface module identifier */
    2 filep bit (116),    /* secondary storage address of file
                             map */
    2 dimhct fixed bin (17),    /* `history information is being
                                   saved' count */
    2 dimtp bit (18),     /* pointer to dim history information */
    2 file_length_lock bit (1);       /* file being lengthened
                                         switch */
```

PST

The declaration for PST entries is

```
  dcl 1 pste ctl (pstep),
    2 llock bit (36),     /* entry lock */
    2 hcsegs (4),         /* list of hardcore per-process
                             segments */
     3 uid bit (70),      /* unique ID */
     3 segno bit (18),    /* segment number */
    2 fstep bit (18),     /* pointer to AST trailers */
    2 dstep bit (18),     /* dst entry for hardcore ring */
    2 sslist bit (18),    /* pointer to special segment list */
    2 sslength bit (18); /* number of special segments */
```

Item sslist if non-zero is a pointer to a list of special segments for the process. The format for this list is

```
dcl pstss (pstep pste.sslength) bit (18);
```

DST

The declaration for a DST entry is

```
dcl 1 dste ctl (dstep),
    2 lock bit (36),        /* dst entry lock */
    2 upcvl bit (36),       /* unstable page control values lock */
    2 ringno bit (18),      /* ring number */
    2 ptp bit (18),         /* pointer to page table */
    2 fdstep bit (18),      /* pointer to next dst entry for the
                               process */
    2 bdstep bit (18),      /* pointer to previous dst entry for
                               the process */
    2 pstep bit (18),       /* pointer to pst entry */
    2 pthc bit (17),        /* page table hold count */
    2 msl bit (9),          /* maximum segment length */
    2 lps bit (1),          /* large page switch */
    2 hps bit (8),          /* hyperpage size */
    2 hpn bit (9);          /* number of hyper-pages in core */
```