To: Distribution

From: John J. Bongiovanni

Date: 02/16/82

Subject: Reliable File Storage: Physical Volume Management. Page Control, and Segment Control

1. ABSTRACT

Reliable File Storage is a significant user-compatible enhancement to the Multics Storage System. Under this enhancement. the state of the Storage System on disk would be consistent at all times. An effect of this is the ability to boot Multics following even catastrophic system failure (crash without Emergency Shutdown, or ESD) without special recovery of all physical volumes which were mounted at the time of the crash. This document describes the supervisor changes necessary to implement Reliable File Storage. These changes are focused in Physical Volume Management, Page Control, and Segment Control. Each of these areas is discussed separately, following a statement of objectives and an outline of design strategy. Where relevant. changes to supervisor data bases are described in detail (include files). At the end of the MTB, all supervisor changes required are summarized by module.

With Reliable File Storage, it will not be necessary to salvage physical volumes which were mounted at the time of a crash without ESD (with the possible exception of the Root Physical Volume (RPV)). At worst. some free records of physical volumes which were mounted at the crash will be lost temporarily to the system. These records can be recovered by a later volume salvage. With the exception of this loss of some free disk space. there is no impact on file system integrity or system performance as a result of using the volumes prior to salvaging. The salvaging of mounted physical volumes. in use by the Storage System, while the system is running, will be discussed in a future MTB.

Send comments on this MTB by one of the following means:

By Multics Mail, on MIT or System M:
Bongiovanni.Multics

By Telephone:
HVN 261-9314 or (617)-492-9314

2. MOTIVATION - CURRENT FILE SYSTEM

When Multics crashes without an ESD, the state of the Storage System may be in an inconsistent state. in the following senses:

The volume map of a physical volume describes records within the paging region which are free (i.e., not allocated to any segment). This map is arbitrarily out of date. as it reflects the state of the volume when the volume was last mounted.

A Volume Table of Contents Entry (VTOCE) may refer to an address which does not belong to the owning segment. Such an address may belong to another segment. be free (hence contain residual data from another segment). or be outside the paging region of the pack. This situation results from physical media failure which overwrites the VTOCE with arbitrary data. Although not usually the result of ESD failure, this situation can occur when the system crashes because of hardware failure.

The linked-list of free VTOCEs may be damaged. due to a crash while an update was in progress or to physical media failure.

Some of these failure modes are potentially catastrophic, and others are merely security hazards. To prevent these problems following a crash without ESD, a physical volume salvage is required for each volume which was mounted at the time of the crash. This salvage accomplishes the following:

Validates each VTOCE for reasonable field contents ("syntactically") and reconstructs the free VTOCE list.

Validates the file map of each VTOCE for valid addresses and reused addresses (i.e., two VTOCEs claiming the same record address in the paging region).

Rebuilds the volume map as the difference between the entire paging region and the set of all addresses claimed by one and only one VTOCE.

A physical volume salvage involves walking the VTOC sequentially. and it takes from 3 to 5 minutes to accomplish. Volume salvaging is invoked automatically when a volume is mounted which had not been demounted properly. Typically, all volumes which were mounted at the time of the crash are mounted automatically (and hence salvaged) when the Initializer process leaves ring-1. The effect is that all volumes are salvaged. sequentially. before Answering System initialization. At a large site, this lengthens the down time seen by users considerably after a crash without ESD.

3. OBJECTIVES

Reliable File Storage will satisfy the following objectives:

Maintain the state of the Storage System on each physical volume in a consistent state at all times. so that the volume can be mounted and used by the Storage System without exceptional recovery action following any type of crash.

Maintain sufficient redundancy in all Storage System data stored on disk to detect likely media failures.

Allow implementation of the Multics release which contains Reliable File Storage without the need for a hierarchy reload or volume reload.

Allow the use of physical volumes initialized or used on the Multics release which contains Reliable File Storage to be used on previous Multics releases. This aids site exposure testing of the new release. since sites would be able to switch between releases without a hierarchy or volume reload.

Allow complete recovery of any physical volume from any suitably privileged process while the volume is mounted and in use by the Storage System. That is. allow a physical volume salvage while the target volume is being used for paging.

The design described in this document satisfies all except the last objective.

4. DESIGN STRATEGY

The following is a general description of the design.

Currently, the Volume Map is copied into a region in an fsmap_seg when the volume is mounted. Record addresses are withdrawn from and deposited to this region (the image of the Volume Map). When the volume is demounted. the image in the fsmap_seg is copied back to the Volume Map. This mechanism is replaced by an intermediate buffer, or stock, of free record addresses per volume. These addresses are withdrawn from the Volume Map, but never used until the updated Volume Map (indicating that they are in use) is written to disk successfully. Normally, record addresses are withdrawn from and deposited into this stock. Associated with each stock is a low threshold and a high threshold. When the number of free records in the stock falls below the low threshold, more record addresses are withdrawn from the Volume Map into the stock. When the number of free records in the stock exceeds the high threshold, some record addresses are deposited from the stock into the Volume Map. When the volume is demounted, any free record addresses in the stock are deposited into the Volume Map. If the system crashes without ESD, all record addresses marked as free in the Volume Map are indeed free. Some free record addresses may be lost after a crash without ESD, however. Addresses which are lost include those which were in the record stock at the time of the crash. and nulled addresses of segments active at the time of the crash.

The Volume Map is divided into sections. each of which contains redundant information which allows media damage to be detected. This redundant information includes a checksum of the bit map in the section. The size of a section is chosen to minimize the overhead associated with checksum computation.

The file map in the VTOCE contains a checksum which allows media damage to be detected.

Currently, all free VTOCEs on a volume are linked in a list. the index of first entry of which is kept in the PVTE. This is replaced by a bit map of free VTOCEs on the volume (a VTOC Map).

A small stock of free VTOCEs is also maintained for performance (to reduce page faults against the VTOC Map). There is no need to synchronize the VTOCE stock and the VTOC Map, as there is for the record stock and the Volume Map.

5. PHYSICAL VOLUME MANAGEMENT

5.1. Disk Pack Layout

The revised disk pack layout is defined in disk_pack.incl.pl1 (Note: all include files referenced in this MTB are attached). The salient changes are as follows:

Structurally, the pack layout is compatible with the MR 9 pack layout. This means that the constant record addresses of the sections of a disk pack are the same (e.g., the VTOC begins at record 8, the Volume Map consists of records 1 through 3, etc.).

The format of the Volume Map (defined in volmap.incl.pl1) has changed incompatibly. The new format is pictured in Attachment 1, and it has the following characteristics:

Each page of the Volume Map is divided into sections which are of equal size. This size is a multiple of the physical device addressable unit (64 words). Initially, each Volume Map section will be 128 words. The Volume Map is described in the Volume Label (begin record number, number of records, and size of a section).

Each Volume Map section contains redundant information for protection against media and transmission failures (specifically, the Physical Volume unique identifier, or PVID; and a checksum). With a section size of 128 words, the checksum can be computed with a small amount of overhead.

Each map word describes 32 addresses. This allows a fast record-within-section to bit-within-map conversion.

In a map word, a bit set ON means that the corresponding record address is free. This affords additional protection, as common hardware and software errors set bits or words to zero.

The format of the VTOC header has changed compatibly (reference vtoc_header.incl.pl1). The dumper bit map has the same format as in MR 9. The header has been retained for compatibility, but it is no longer used.

It formerly contained a description of the VTOC (now in the volume label) and a VTOCE index of the first in a threaded list of free VTOCEs. Free VTOCEs are now described by the VTOC Map.

The VTOC map occupies what was an unused record. It has the same format as the Volume Map, except that it describes VTOCE indices instead of record addresses. In the initial implementation, the maximum number of VTOCEs per pack is reduced from 36720 to 31744 (the constraint is the number of VTOCEs which can be described in one map page). It is extremely unlikely that this reduction will affect existing sites.

The Volume Label has changed compatibly (reference fs_vol_label.incl.pl1), and it has the following characteristics:

A disk pack is now completely self-describing. The constants used to find sections of the pack (e.g., the VTOC origin) have been replaced by fields in the label. In the initial release. these fields will contain the MR 9 constants. This self-description will allow the layout of the pack to be changed easily in future releases, if necessary.

A copy of the Volume Label is kept in a previously unused record in the Label Region. This redundancy will allow the pack to be recovered in the event of damage to the label. Such redundancy is more important now that the pack is self-describing, based on fields in the Volume Label.

A Volume Map version has been included as a field which was zero previously. This version will be used to trigger conversion of the Volume Map and generation of the VTOC map when an MR 9 pack is mounted.

The field time unmounted has been moved. In its previous location, a value will be placed which will trigger a volume salvage if a pack with the new layout is mounted on a system running MR 9. This relatively minor change to the Volume Label format allows packs with the new layout to be used in MR 9.

The field vol_trouble_count is a count of the number of times damage to any of the control structures on the pack has been detected since the last physical volume salvage. Control structures include the Volume Map. the VTOC Map, and the VTOCE file maps. This field is used in a heuristic at volume acceptance which triggers automatic volume salvage.

5.2. Compatibility Considerations

From the previous section, it should be obvious that a pack can be moved between an MR 9 system and a later system with the following costs on each cross-system mount:

When an MR 9 pack is mounted on a later system, a new Volume Map and VTOC Map must be generated. In order for VTOCE checksums to be correct, they must be computed for each in-use VTOCE. This requires a sequential scan of the entire VTOC. The VTOC Map is built during this scan, as free VTOCEs are detected easily. The cost is approximately that of a volume salvage.

When a pack with the new layout is mounted on an MR 9 system, a full volume salvage is required. This will rebuild the Volume Map in the proper format from the VTOCE file maps, and it will rebuild the threaded list of free VTOCEs by syntactic detection of unused VTOCEs. A volume salvage requires a sequential scan of the entire VTOC.

5.3. System Data Bases

The following system data bases are changed as indicated:

The File System Device Control Table (fsdct) is eliminated. The few remaining useful fields in the fsdct are moved to the header of the Physical Volume Table (PVT).

The PVT header has been expanded, as has each PVTE. Reference pvt.incl.pl1.

The segments which contained volume bit maps (fsmap_seg's) have been eliminated.

A new segment, stock_seg, contains a stock of free addresses for each mounted physical volume (reference stock_seg.incl.pl1). It also contains a stock of free VTOCEs for each mounted physical volume. This is an unpaged segment, with sufficient space to hold stocks for all disk drives defined in the configuration. Each record_stock is 64 words long and contains enough room for 116 record addresses. Each VTOCE stock is 8 words long and contains enough room for 16 free VTOCE indices.

A volmap_seg is active and entry-held for each mounted physical volume. This segment is not in the File System; it describes the Volume Map and VTOC Map of the volume. Initially, it is a 4K segment, with the first 3 pages describing the Volume Map and the 4th page describing the VTOC map. The offset and length of the Volume Map and of the VTOC Map within this segment are in the PVTE, along with a Segment Descriptor Word (SDW) to access the segment.

volmap_abs seg is an abs-seg used to access a volmap_seg.

5.4. System Initialization

The system data bases referenced above are initialized as follows:

A PVTE, a record_stock entry, and a vtoce stock entry is allocated for each disk drive defined in the configuration.

The record_stock and vtoce stock entries are initialized as empty.

The PVTE is initialized to point to the record_stock and VTOCE stock associated with the drive. The SDW describing the volmap seg is initialized to invalid (segment fault).

Other changes are necessary so that Page Control can withdraw record addresses from the Hardcore Partitions during initialization.

The first page of each Hardcore Partition is initialized as a fake Volume Map. This Volume Map describes the Partition. and it is initialized to indicate that the Partition is free. except for the first record.

A volmap seg is activated for each volume which is defined as containing a Hardcore Partition. This segment describes the fake Volume Map.

There is no need for a fake VTOC map.

When the RPV is accepted. all volmap_seg's are destroyed and record_stocks are cleared to empty. Relevant PVTE fields of affected drives are re-initialized. From this point (RPV acceptance), there is no longer need to withdraw addresses from the Hardcore Partition.

5.5. Volume Acceptance

When a Storage System volume is mounted. it is accepted as follows:

If the Volume Map version is 0, the volume is salvaged. This results in the generation of a Volume Map (in the new format) and a VTOC Map. Additionally, a checksum is computed for each VTOCE. By proper sequencing, this is done in a manner which is safe across crashes. That is, if a crash occurs during the salvage, the pack is in a consistent state.

If the volume belongs to the Root Logical Volume (RLV), a heuristic is used to determine whether the volume needs to be salvaged. The volume will be salvaged if there are fewer than 100 free records and the vol_trouble count in the Volume Label is non-zero. Note that a crash without ESD adds one to the vol trouble count. If the volume does not belong to the RLV, warning messages will be printed on the console if the number of free records is exceptionally low, or if the vol_trouble_count is exceptionally high.

A volmap_seg is activated and entry-held. An SDW to this segment is placed in the PVTE.

PVTE fields are initialized from information in the Volume Label. The Volume Map is scanned, and an array of flags is built in the PVTE indicating which Volume Map sections contain free record addresses.

No record addresses or free VTOCE indices are withdrawn from the respective maps on disk. This will be done on demand (i.e., when the first page is created on the device or the first VTOCE is created).

5.6. Volume Demounting

When a Storage System volume is demounted. the following occurs:

All VTOCE indices are cleared from the vtoce_stock entry and updated to the VTOC Map.

All record addresses are cleared from the record_stock entry and updated to the Volume Map.

The volmap_seg SDW in the PVTE is invalidated.

The volmap_seg is destroyed.

6. PAGE CONTROL

6.1. Overview

The mechanism to implement this design is focused in Page Control, in the routine which manages the depositing and withdrawing of record addresses. When a new page is created, a record address for that page is withdrawn from the stock of free addresses. A record address may be deposited (returned to the pool of free record addresses) for a number of reasons. Deletion of a segment is a simple example. During deletion. all pages belonging to the segment are deposited. A more common but more complex example is the depositing of nulled pages during segment deactivation. Nulled pages are pages which contain all zeros (logically), but which have record addresses assigned. They are pages which have been created recently and never written to disk, or pages which have recently been cleared to zeros. Nulled pages are never reflected in the file map in the VTOCE, and they are deposited during deactivation.

The mechanism for withdrawing and depositing record addresses is simple, conceptually.

When a withdrawal is requested. attempt to withdraw a record from the stock for that volume. If none are left. initiate the withdrawal of record addresses from the Volume Map into the stock, and await the completion of this activity. No addresses which are withdrawn from the Volume Map can be used until the updated Volume Map has been written to disk. So the completion of this activity corresponds to the completion of the write I/O to the Volume Map.

When a deposit is requested, attempt to deposit into the stock for that volume. In the current system, an address is deposited only after the VTOCE which previously owned the address has been written to disk. So such addresses can be reused immediately. If there is not enough room in the stock for all addresses to be deposited, deposit the remainder directly to the Volume Map.

When the number of free addresses in the stock falls below a threshold, initiate the withdrawal of record addresses from the Volume Map into the stock. No address withdrawn from the Volume Map can be used until the updated Volume Map has been written successfully to disk.

When the number of free addresses in the stock grows to higher than a threshold, initiate the depositing of record addresses from the stock into the Volume Map.

The thresholds referenced above are constants which will be determined from performance measurements during the development. The low threshold is likely to be around 50. This allows a minimum of 100 milliseconds to withdraw more addresses from the Volume Map before the stock empties (which is sufficient time for both I/Os involved). Large systems generate page faults at the rate of approximately one every 2 milliseconds. and the page fault rate is an upper bound on the address withdrawal rate.

The complexity arises from the low level of the system in which these operations must be accomplished, the interrupt-like flavor of some of them, concurrency constraints. and the place occupied by withdrawal and depositing in Multics. These are discussed in the next section.

6.2. Constraints

The following considerations constrain the implementation:

> Currently, withdrawal is called by Page Control with the global Page Table Lock held. Depositing, however. is called by Segment Control. without any canonical locks. and typically in an unwired environment.
>
> Withdrawing from and depositing to the record stock can be done with lockless protocols. as each can be implemented as an atomic operation against one cell in the stock. This is not possible for withdrawing from and depositing to the Volume Map, due to checksums.
>
> A Volume Map page (accessed via the volmap_seg) must not be modified between the time a write I/O is requested for it until the I/O is complete. This is necessary to guarantee the consistency of the page on disk.

6.3. Volume Map Update Strategy

The Volume Map is updated under two different circumstances:

> On demand, when a withdrawal is requested and the record stock contains no free addresses. or when a deposit is requested and the record stock is full. In this case. the requesting process must wait for completion of the Volume Map update (including writing the updated Volume Map to disk). This is called a demand update.
>
> When the number of addresses in the record stock falls outside of the thresholds for the stock. In this case, an update of the Volume Map is initiated. but there is no need for the process which notices the condition to wait for the completion of the update. This is called an asynchronous update.

Correct synchronization of operations against the Volume Map is implemented by a per-volume Volume Map lock and a finite-state model of asynchronous Volume Map updates. Both the Volume Map lock and the current asynchronous update state are maintained in the PVTE for the device. The following conventions apply:

> The asynchronous update state, or state, of the Map can

be one of the following:

Idle (I) - no asynchronous activity in progress

Read-in-Progress (R) - the stock is outside of threshold and requires a Volume Map update. A read of a Volume Map page has been requested but has not been noticed as having completed.

Write-in-Progress (W) - A Volume Map page has been modified. A write of the page has been requested but has not been noticed as having completed.

The state may be changed from Idle only under the protection of both the per-volume Volume Map lock and the Global Page Table lock. The state may be changed from Read-in-Progress or Write-in-Progress only under the protection of the Global Page Table lock.

An asynchronous Volume Map update is initiated under the protection of the per-volume Volume Map lock. After acquiring the lock, a read is requested of an appropriate page. and the state changed to R. The completion of the update is done by Page Control, which polls periodically for pending Volume Map activity.

The Volume Map may be updated from call-side under the protection of the per-volume Volume Map lock. The state must be I. After modifying Volume Map pages, each page modified must be written to disk successfully before releasing the lock.

6.4. Locking Hierarchy

The per-volume Volume Map lock occupies a place in the locking hierarchy between the AST lock and the Global Page Table lock. As a consequence. a process may take page faults with the per-volume Volume Map locked. A process may not take a page fault which requires record address withdrawal with a per-volume Volume Map locked. A process which holds the Global Page Table lock may acquire a per-volume Volume Map lock, but it may not wait for it.

6.5. Locking Services

The following services implement the protocols outlined above.

lock_wired_nowait

> This routine attempts to lock the per-volume Volume Map lock for a specified volume. It is called with the global Page Table Lock held. It returns with the lock held only if the lock can be acquired immediately and the state is I. Otherwise, it returns with an indication of failure to acquire the lock.

lock_wired_wait

> This routine attempts to lock the per-volume Volume Map lock for a specified volume. It is called with the global Page Table Lock held. It returns with the lock held only if the lock can be acquired immediately and the state is I. Otherwise, it returns with an appropriate wait event (for the lock or for pending I/O against the Volume Map).

lock_unwired

> This routine attempts to lock the per-volume Volume Map lock for a specified volume. When it returns, the lock is held and the state is I. It will wait, if necessary.

unlock

> This routine unlocks the per-volume Volume Map lock for a specified volume, notifying any processes which are waiting for the lock.

grab_volmap_page_unwired

> This routine reads a Volume Map page into memory and wires it for modification from an unwired environment. It is called with the lock held and the state I. If the number of records in the stock exceeds the high threshold, any excess records are deposited into the Volume Map.

write volmap_page unwired

> This routine writes a Volume Map page to disk, unwires it, and waits for completion of the I/O. It is called with the lock held and the state I.

6.6. Deposit/Withdrawal Services

The following services allow depositing and withdrawal or records:

withdraw_record_wired

> This routine attempts to withdraw a single record address from the record stock for a specified device in a lockless manner. If it cannot. it returns either an error (out-of-physical-volume) or a wait event (for completion of asynchronous Volume Map update). It is called with the global Page Table lock held.

deposit record_unwired

> This routine attempts to deposit a single record address into the record stock for a specified device in a lockless manner. If it cannot. it deposits the address directly to the Volume Map, waiting if necessary.

deposit_list_unwired

> This routine attempts to deposit a list of record addresses into the record stock for a specified device in a lockless manner. If it cannot deposit the enitre list into the stock, it deposits as many as it can into the stock. and the remainder into the Volume Map. It waits if necessary.

6.7. Error Handling

If a checksum of a section of the Volume Map is found to be in error when it is read from disk, the section is assumed to be allocated in its entirety. The affected section of the Volume Map is changed to indicate this, and a message is printed on the console to this effect.

7. SEGMENT CONTROL

7.1. Overview

Two areas of Segment Control are of interest: allocation and freeing of VTOCEs, and VTOCE checksums.

7.2. Allocation and Freeing of VTOCEs

VTOCEs are allocated and freed under the protection of the per-volume VTOC Map lock.

When a VTOCE is allocated. it is allocated from the vtoce stock (if the stock contains any free VTOCE indices). Otherwise, it is allocated from the VTOC Map. and the vtoce stock is replenished from the VTOC Map at the same time.

When a VTOCE is freed. it is freed into the vtoce stock (if the stock contains empty slots). Otherwise, it is freed into the VTOC Map.

7.3. VTOCE Checksum

The revised VTOCE format is depicted in vtoce.incl.pl1. An unused field (vtoce.infqcnt) is used for the checksum. This checksum is a checksum of that portion of the file map which is in use. That is, it is a checksum of vtoce.csl file map entries (current segment length). The checksum is validated each time the segment is activated, and it is recomputed when the file map is updated.

7.4. Error Handling

When a checksum error occurs. the entire file map is assumed to be invalid. The segment is marked as damaged, and it cannot be accessed until the next physical volume salvage or until it is truncated explicitly. At the next physical volume salvage, any record address in the file map of a damaged segment which is not claimed by another VTOCE or the Volume Map (as a free record address) is left in the file map. and the segment is made accessible. If a segment damaged in this way is truncated explicitly. no record addresses are deposited.

8. LIMITATIONS AND PARAMETERS

The following limitations exist under the new disk pack layout described in this document:

> The number of Volume Map pages is 3. which provides for record addresses up to 95231. This can be increased by increasing the number of Volume Map pages. at the cost of a larger ASTE for the volmap_seg (when the device is mounted).
>
> The number of VTOC Map pages is 1. which provides for VTOCE indices up to 31743 (that is. a pack may have a maximum of 31743 VTOCEs. or distinct segments. on it). This can be increased by increasing the number of VTOC Map pages. at the cost of a larger ASTE for the volmap_seg (when the device is mounted).
>
> To put the numbers above in some perspective. the largest device supported by Multics currently has 67200 records. Further. Multics cannot support devices with more than 131072 records without restructuring disk control or page control.
>
> With this design. permanent wired storage increases as follows:
>
>> The PVT header is 20 words (previously 8).
>>
>> Each PVTE is 22 words (previously 12). One is required for each disk drive defined in the configuration.
>>
>> Each record stock is 64 words. One is

required for each disk drive defined in the configuration.

Each VTOCE stock is 8 words. One is required for each disk drive defined in the configuration.

In addition. a 4K Active Segment Table Entry (ASTE) is active and entry-held for each mounted Storage System volume.

9. METERING

The metering data collected is described in the include file stock_seg.incl.pl1, under the structure rsmeters. Most of this data will be useful in tuning the design. and most likely will not be of use to sites. Additional metering will be developed during implementation. as appropriate.

10. PHASING

This design will be implemented in three phases. as follows:

Page Control and Physical Volume Management changes to use record stocks for disk packs with the current layout. This involves changing the PVTE format, at least recompiling all programs which reference the PVTE. This will allow longer exposure of the most complex portions of the implementation. It will also allow more time to meter and tune the implementation.

Segment Control changes to maintain and use VTOCE checksums. This phase requires volume salvages to implement.

Implementation of the new disk pack layout. This phase requires volume conversions to implement.

11. SUMMARY OF CHANGES

All changes required in this design are indicated below by module.

accept fs_disk
: Eliminate use of fsdct. If an MR 9 volume is mounted. call convert_volume_map, convert_vtoc_map, and walk_vtoc_compute checksum to convert the volume to proper format. Activate and entry-hold the volmap_seg. When called for the RPV. call make sdw$reset_rpv to terminate allocation against the Hardcore Partition.

activate
: If the VTOCE checksum is invalid, truncate the vtoce and damage the segment.

adopt_seg
: Recompile for new include files.

boot (BOS)
: Pick up time unmounted correctly under both new and old label format.

copy_fdump
: Instead of using the fsdct to find the Dump Partition. find the PART DUMP card in the Config deck and read the label of that volume.

create_vtoce
: Call checksum to compute vtoce.checksum for new (empty) VTOCE. Eliminate use of the obsolete field vtoce.infqcnt.

dbm_man
: Recompile for new include files.

dctl
: Recompile for new include files.

delete_volume_log
: Eliminate use of fs vol_label.incl.pl1.

demount_pv
: Destroy the volmap seg. Null out new fields in PVTE.

device_control
: Recompile for new include files.

disk_control
: Recompile for new include files.

disk_emergency
    Recompile for new include files.

disk_init
    Recompile for new include files.

disk_left_
    Recompile for new include files.

disk_rebuild
    Rework for new disk pack format.

disk_rebuild_caller
    Rework for new disk pack format. Detect MR 9 label and print error message.

display_ast_
    Recompile for new include files.

display_label
    Display new label fields.

display_pvolog
    Eliminate use of fs vol_label.incl.pl1.

display_volume_log
    Eliminate use of fs vol_label.incl.pl1.

dmpr_finish_
    Eliminate use of fs vol_label.incl.pl1.

dmpr_log_
    Eliminate use of fs vol_label.incl.pl1, vtoce.incl.pl1.

dmpr_output_
    Eliminate use of fs vol_label.incl.pl1, vtoce.incl.pl1.

dump_volume_
    Eliminate use of fs vol_label.incl.pl1

dump_vtoce
    Recompile for new include files.

find_partition_
    Recompile for new include files.

free_store
    This module is replaced by Page Control stock management.

fs_unload_disk_interrupt

Recompile for new include file.

fsdct
Deleted.

fsout_vol
Update all free entries in the stocks to the VTOC Map and the Volume Map (first the VTOCE stock, then the record stock). Eliminate use of fsdct.

get_io_segs
Get wired space for record_stock_seg.

get_pvtx
Recompile for new include files.

hc_dmpr_primitives
Build bit map of in-use VTOCEs from VTOC Map. Checksum the file map in the VTOCE.

hp_delete_vtoce
Recompile for new include files.

init_disk_pack_
Initialize new fields in the Volume Label.

init_empty_root
Initialize new fields in the Volume Label.

init_lvt
Eliminate use of fsdct.

init_pvt
Eliminate use of fsdct, fsmap_seg's. Call init_record_stock for each device. Reserve the first page of each Hardcore Partition as a fake volume map. and call empty_volume_map to initialize each one. Activate and entry-hold a volmap segment for each Hardcore Partition. call make_sdw$init_hcp_thread to set up allocation of space on Hardcore Parititon.

init_root_dir
Eliminate use of the fsdct.

init_sys_var
Eliminate use of fsdct.

init_vol_header_
Rework for new disk pack format.

initializer
Eliminate use of fsdct. Establish a bad_dir handler

to salvage the offending directory (eliminating the need for a depth 2 directory salvage after a crash without ESD).

list_vols
Recompile for new include files.

load_vol_map
Delete.

logical_volume_manager
Recompile for new include files.

make_sdw
Maintain thread pointer for Hardcore Parition allocation in static storage, instead of in the fsdct.

mdx
Recompile for new include files.

merge_volume_log
Eliminate use of fs vol_map.incl.pl1.

on_line_salvager
Eliminate reference to fsdct.

page_error
Rework error messages.

page_fault
Use new address withdrawal mechanism, including recursive page faults on wait conditions.

partition_io
Recompile for new include files.

pc
Rework for new deposit/withdrawal mechanism.

priv_delete_vtoce
Return a cleared VTOCE to the free VTOCE pool.

purge_volume_log
Eliminate use of fs vol_label.incl.pl1.

pvname_to_pvtx_
Recompile for new include files.

pvt
Make a CDS.

rcp_disk_

Recompile for new include files.

rcp_init_disk_sharing
Recompile for new include files.

read_disk
Recompile for new include files.

record_to_vtocx
Recompile for new include files.

recover_volume_log
Eliminate use of fs vol_map.incl.pl1. vtoce.incl.pl1.

reload_volume_
Eliminate use of vtoc_header.incl.pl1.

reloader
Rework for new disk pack format.

restor (BOS)
Change for new Volume Map.

retrieve_from_volume_
Eliminate use of fs vol_label.incl.pl1.

retv_copy
Eliminate use of obsolete vtoce.infqcnt.

retv_vol control_
Eliminate use of fs vol_label.incl.pl1.

ring_0_peek
Recompile for new include files.

rldr_check_pvol_
Change for new disk pack format.

rldr_input_
Change for new Volume Label (pick up time unmounted correctly).

rldr_output_
Change for new disk pack format. Compute VTOCE checksum.

rldr_volume_map_
Change for new Volume Map format.

rldr_vtoc_header_
Change for VTOC Map.

salv_dir_checker_
    Eliminate use of fsdct.

salv_directory
    Eliminate use of fsdct.

salvage_pv
    Rework for new disk pack format. Validate checksum for each VTOCE examined.

salvager
    Eliminate use of fsdct. Eliminate salvage to depth 2.

save (BOS)
    Change for new Volume Map format.

seg_fault
    Eliminate use of fsdct.

segment_mover
    Eliminate use of the obsolete field vtoce.infqcnt.

set_disk_table_loc
    Eliminate use of fsdct.

set_sons_lvid
    Eliminate use of fsdct.

set_volume_log
    Eliminate use of fs vol_map.incl.pl1.

shutdown
    Eliminate use of fsdct.

sstn (BOS)
    Pick up VTOC origin from Volume Label.

status_
    Eliminate use of fsdct.

sweep_pv
    Recompile for new include files.

syserr_log_init
    Pick up location of LOG Parition from config deck instead of fsdct.

truncate_vtoce
    Compute VTOCE checksum.

update_vtoce
    Compute file map checksum.

vacate_pv
Recompile for new include files.

verify_dump_volume
Eliminate use of fs vol_label.incl.pl1. vtoce.incl.pl1.

verify_label_
Recompile for new include files.

vm_vio
Rework for new disk pack format.

volume_cross check

vtoc_attributes
Eliminate use of the obsolete field vtoce.infqcnt.
Compute VTOCE checksum.

vtoc_man
Use new VTOCE allocation/freeing scheme.

vtocx_to_record
Recompile for new include files.

wired_shutdown
Eliminate use of fsdct.

ATTACHMENT 1

FORMAT OF VOLUME MAP SECTION

0	Physical Volume Identifier	
1	Checksum for this Section	
2	Address of First Record	Number Words in Map
3	Number Records in Map	Number Free Records
4 - N	Section Map (Bit Map with 32 Records per Word	

ATTACHMENT 2

INCLUDE FILES REFERENCED IN THIS MTB

```
/*      BEGIN INCLUDE FILE...disk_pack.incl.pl1        Last Modified January 1982 for new volume map   */

/*
   All disk packs have the standard layout described below:

   Record  0                : contains the label, as declared in fs_vol_label.incl.pl1.
   Record  1 to 3           : contains the volume map, as declared in vol_map.incl.pl1
   Record  4 to 5           : contains the dumper bit map, as declared in dumper_bit_map.incl.pl1
   Record  6                : contains the vtoc map, as declared in vtoc_map.incl.pl1
   Record  7                : contains a copy of the label, as declared in fs_vol_label.incl.pl1
   Records 8 to n-1         : contain the array of vtoc entries; ( n is specified in the label)
   each record contains 5 192-word vtoc entries. The last 64 words are unused.
   Records n to N-1         : contain the pages of the Multics segments. ( N is specified in the label)

   Sundry partitions may exist within the region n to N-1, withdrawn or not as befits the meaning
   of the particular partition.


   A conceptual declaration for a disk pack could be:

   dcl 1 disk_pack,
   2 label_record           (0 : 0)            bit(36 * 1024),
   2 volume_map_record      (1 : 3)            bit(36 * 1024),
   2 dumper_bit_map_record  (4 : 5)            bit(36 * 1024),
   2 vtoc_map_record        (6 : 6)            bit(36 * 1024),
   2 label_record_copy      (7 : 7)            bit(36 * 1024),
   2 vtoc_array_records     (8 : n-1),
   3 vtoc_entry ( 5 )                          bit(36 * 192),
   3 unused                                    bit(36 * 64),
   2 Multics_pages_records  (n : N-1)          bit(36 * 1024);


*/

dcl (LABEL_ADDR             init (0),                   /* Address of Volume Label */
     VOLMAP_ADDR            init (1),                   /* Address of first Volume Map record */
     VTOC_MAP_ADDR          init (6),                   /* Address of first VTOC Map Record */
     VTOC_ORIGIN            init (8),                   /* Address of first record of VTOC */
     SECTORS_PER_VTOCE      init (3),
     VTOCES_PER_RECORD      init (5),
     DEFAULT_HCPART_SIZE    init (1000),                /* Size of Hardcore Partition */
     MAX_VTOCE_PER_PACK     init (31774))               /* Limited by size of VTOC Map */
                            fixed bin (17) int static options (constant);

/*      END INCLUDE FILE...disk_pack.incl.pl1   */
```

```
/* BEGIN INCLUDE FILE ... fs_vol_label.incl.pl1 .. last modified January 1982 for new volume map format  */

/* This is the label at fixed location of each physical volume. Length 1 page */

dcl  labelp ptr;

dcl 1 label based (labelp) aligned,

/* First comes data not used by Multics.. for compatibility with GCOS */

    2 gcos (5*64) fixed bin,

/* Now we have the Multics label */

    2 Multics char (32) init ("Multics Storage System Volume"), /* Identifier */
    2 version fixed bin,                                     /* Version 1 */
    2 mfg_serial char (32),                                  /* Manufacturer's serial number */
    2 pv_name char (32),                                     /* Physical volume name. */
    2 lv_name char (32),                                     /* Name of logical volume for pack */
    2 pvid bit (36),                                         /* Unique ID of this pack */
    2 lvid bit (36),                                         /* unique ID of its logical vol */
    2 root_pvid bit (36),                                    /* unique ID of the pack containing the root. everybody must agree. */
    2 time_registered fixed bin (71),                        /* time imported to system */
    2 n_pv_in_lv fixed bin,                                  /* # phys volumes in logical */
    2 vol_size fixed bin,                                    /* total size of volume, in records */
    2 vtoc_size fixed bin,                                   /* number of recs in fixed area + vtoc */
    2 not_used bit (1) unal,                                 /* used to be multiple_class */
    2 private bit (1) unal,                                  /* TRUE if was registered as private */
    2 flagpad bit (34) unal,
    2 max_access_class bit (72),                             /* Maximum access class for stuff on volume */
    2 min_access_class bit (72),                             /* Minimum access class for stuff on volume */
    2 password bit (72),                                     /* not yet used */
    2 pad1 (16) fixed bin,
    2 time_mounted fixed bin (71),                           /* time mounted */
    2 time_map_updated fixed bin (71),                       /* time vmap known good */
    2 old_time_unmounted fixed bin,                          /* set to cause salvage pre-MR10 */
    2 volmap_version fixed bin,                              /* version of volume map (currently 1) */
    2 time_salvaged fixed bin (71),                          /* time salvaged */
    2 time_of_boot fixed bin (71),                           /* time of last bootload */
    2 time_unmounted fixed bin (71),                         /* time unmounted cleanly */
    2 pad1a (2) fixed bin,
    2 vol_trouble_count fixed bin,                           /* Number times structure damaged detected since last salvage */
    2 err_hist_size fixed bin,                               /* size of pack error history */
    2 time_last_dmp (3) fixed bin (71),                      /* time last completed dump pass started */
    2 time_last_reloaded fixed bin (71),                     /* what it says */
    2 pad2 (40) fixed bin,
    2 root,
      3 here bit (1),                                        /* TRUE if the root is on this pack */
      3 root_vtocx fixed bin (35),                           /* VTOC index of root, if it is here */
      3 shutdown_state fixed bin,                           /* Status of hierarchy */
      3 pad7 bit (1) aligned,
      3 disk_table_vtocx fixed bin,                          /* VTOC index of disk table on RPV */
```

```
      3 disk_table_uid bit (36) aligned,                 /* UID of disk table */
      3 esd_state fixed bin,                             /* State of esd */
    2 volmap_record fixed bin,                           /* Begin record of volume map */
    2 size_of_volmap fixed bin,                          /* Number of records in volume map */
    2 vtoc_map_record fixed bin,                         /* Begin record of VTOC map */
    2 size_of_vtoc_map fixed bin,                        /* Number of records in VTOC map */
    2 volmap_unit_size fixed bin,                        /* Number of words per volume map section */
    2 vtoc_origin_record fixed bin,                      /* Begin record of VTOC */
    2 dumper_bit_map_record fixed bin,                   /* Begin record of dumper bit-map */
    2 pad3 (54) fixed bin,
    2 nparts fixed bin,                                  /* Number of special partitions on pack */
    2 parts (47),
      3 part char (4),                                   /* Name of partition */
      3 frec fixed bin,                                  /* First record */
      3 nrec fixed bin,                                  /* Number of records */
      3 pad5 fixed bin,
    2 pad4 (5*64) fixed bin;

dcl  Multics_ID_String char (32) init ("Multics Storage System Volume") static;

/* END INCLUDE FILE fs_vol_label.incl.pl1 */
```

```
"BEGIN INCLUDE FILE pvt.incl.alm


      "Created 02/23/82  1151.7 est Tue by convert_include_file,
      "   Version of 12/01/81  1540.3 est Tue.

      "Made from >user_dir_dir>Multics>Bongiovanni>cctm>pvt.incl.pl1,
      "   modified 02/23/82  1151.6 est Tue

"
"     Structure pvt
"


      equ     pvt.n_entries,0
      equ     pvt.max_n_entries,1
      equ     pvt.n_in_use,2
      equ     pvt.rwun_pvtx,3
      equ     pvt.shutdown_state,4
      equ     pvt.esd_state,5
      equ     pvt.prev_shutdown_state,6
      equ     pvt.prev_esd_state,7
      equ     pvt.root_lvid,8
      equ     pvt.root_pvtx,9
      equ     pvt.root_vtocx,10
      equ     pvt.disk_table_vtocx,11
      equ     pvt.disk_table_uid,12

      equ     pvt.rpvs_requested_word,13
      bool    pvt.rpvs_requested,400000     " DU

      equ     pvt.rlv_needs_salv_word,14
      bool    pvt.rlv_needs_salv,400000     " DU

      equ     pvt.volmap_lock_wait_constant,15
      equ     pvt.volmap_idle_wait_constant,16
      equ     pvt.vtoc_map_lock_wait_constant,17
      equ     pvt.n_volmap_locks_held,18
      equ     pvt.n_vtoc_map_locks_held,19
      equ     pvt.last_volmap_time,20       " DOUBLE
      equ     pvt.last_vtoc_map_time,22     " DOUBLE
      equ     pvt.total_volmap_lock_time,24 " DOUBLE
      equ     pvt.total_vtoc_map_lock_time,26 " DOUBLE
      equ     pvt.n_volmap_locks,28
      equ     pvt.v_vtoc_map_locks,29
      equ     pvt.array,30                  " LEVEL 2


"
"     Structure pvte
"
```

```
equ      pvte_size,21

equ      pvte.pvid,0
equ      pvte.lvid,1

equ      pvte.dmpr_in_use_word,2
bool     pvte.dmpr_in_use,400000         " DU
equ      pvte.skip_queue_count_word,2
equ      pvte.skip_queue_count_shift,9
bool     pvte.skip_queue_count_mask,777777
equ      pvte.brother_pvtx_word,2
equ      pvte.brother_pvtx_shift,0
bool     pvte.brother_pvtx_mask,000777

equ      pvte.devname,3

equ      pvte.device_type_word,4
equ      pvte.device_type_shift,27
bool     pvte.device_type_mask,000777
equ      pvte.logical_area_number_word,4
equ      pvte.logical_area_number_shift,18
bool     pvte.logical_area_number_mask,000777
equ      pvte.used_word,4
bool     pvte.used,400000                " DL
equ      pvte.storage_system_word,4
bool     pvte.storage_system,200000      " DL
equ      pvte.permanent_word,4
bool     pvte.permanent,100000           " DL
equ      pvte.testing_word,4
bool     pvte.testing,040000             " DL
equ      pvte.being_mounted_word,4
bool     pvte.being_mounted,020000       " DL
equ      pvte.being_demounted_word,4
bool     pvte.being_demounted,010000     " DL
equ      pvte.check_read_incomplete_word,4
bool     pvte.check_read_incomplete,004000 " DL
equ      pvte.device_inoperative_word,4
bool     pvte.device_inoperative,002000 " DL
equ      pvte.rpv_word,4
bool     pvte.rpv,001000                 " DL
equ      pvte.salv_required_word,4
bool     pvte.salv_required,000200       " DL
equ      pvte.being_demounted2_word,4
bool     pvte.being_demounted2,000100    " DL
equ      pvte.vol_trouble_word,4
bool     pvte.vol_trouble,000040         " DL
equ      pvte.vacating_word,4
bool     pvte.vacating,000020            " DL
equ      pvte.hc_part_used_word,4
bool     pvte.hc_part_used,000010        " DL
equ      pvte.volmap_lock_notify_word,4
bool     pvte.volmap_lock_notify,000004 " DL
equ      pvte.volmap_idle_notify_word,4
bool     pvte.volmap_idle_notify,000002 " DL
equ      pvte.vtoc_map_lock_notify_word,4
bool     pvte.vtoc_map_lock_notify,000001 " DL

equ      pvte.n_free_vtoce,5             " UPPER
equ      pvte.vtoc_size,5                " LOWER
```

```
	equ	pvte.dbmrp,6			" UPPER

	equ	pvte.nleft,7			" UPPER
	equ	pvte.totrec,7			" LOWER

	equ	pvte.dim_info,8

	equ	pvte.curn_dmpr_vtocx,9		" UPPER

	equ	pvte.n_vtoce,10			" LOWER

	equ	pvte.volmap_seg_sdw,12		" DOUBLE
	equ	pvte.volmap_astep,14

	equ	pvte.volmap_offset,15		" UPPER
	equ	pvte.vtoc_map_offset,15		" LOWER

	equ	pvte.volmap_lock,16
	equ	pvte.vtoc_map_lock,17
	equ	pvte.volmap_stock_ptr,18
	equ	pvte.vtoce_stock_ptr,19

	equ	pvte.volmap_async_state,20	" UPPER
	equ	pvte.volmap_async_page,20	" LOWER

	equ	VOLMAP_ASYNC_IDLE,0		" MANIFEST
	equ	VOLMAP_ASYNC_READ,1		" MANIFEST
	equ	VOLMAP_ASYNC_WRITE,2		" MANIFEST

"END INCLUDE FILE pvt.incl.alm
```

```
/*      BEGIN INCLUDE FILE ... pvt.incl.pl1 ... last modified January 1982 */

/* The physical volume table (PVT) is a wired-down table.
   It has one entry for each spindle present, be it for
   Storage System or "I/O" use.
*/

     dcl     pvt$                     ext,
             pvtp                     ptr,
             pvtep                    ptr;

     dcl     1 pvt                    based (pvtp) aligned,

               2 n_entries            fixed bin (17),          /* number of PVT entries */
               2 max_n_entries        fixed bin (17),          /* max number of PVT entries */
               2 n_in_use             fixed bin (17),          /* number of PVT entries in use */
               2 rwun_pvtx            fixed bin,               /* rewind_unloading pvtx */
               2 shutdown_state       fixed bin,               /* state of previous shutdown */
               2 esd_state            fixed bin,               /* state of ESD, >0 iff in ESD */
               2 prev_shutdown_state fixed bin,                /* shutdown state of previous bootload */
               2 prev_esd_state       fixed bin,               /* ESD state of previous bootload */

               2 root_lvid            bit (36) aligned,        /* Logical volume ID of Root Logical Volume (RLV) */
               2 root_pvtx            fixed bin,               /* Index to PVTE for Root Physical Volume (RPV) */
               2 root_vtocx           fixed bin,               /* VTOCE index for root (>) */
               2 disk_table_vtocx     fixed bin,               /* VTOCE index for disk table on RPV */
               2 disk_table_uid       bit (36) aligned,        /* File System UID for disk_table */
               2 rpvs_requested       bit (1) aligned,         /* RPVS keyword given on BOOT */
               2 rlv_needs_salv       bit (1) aligned,         /* RLV required (not requested) salvage */
               2 volmap_lock_wait_constant bit (36) aligned,/* For constructing wait event: OR pvte_rel into lower */
               2 volmap_idle_wait_constant bit (36) aligned,/* For constructing wait event: OR pvte_rel into lower */
               2 vtoc_map_lock_wait_constant bit (36) aligned, /* For constructing wait event: OR pvte_rel into lower */
               2 n_volmap_locks_held fixed bin (17),           /* Current number of volmap locks held */
               2 n_vtoc_map_locks_held fixed bin (17),         /* Current number of VTOC Map locks held */
               2 last_volmap_time fixed bin (71),              /* Time a volmap was last locked/unlocked */
               2 last_vtoc_map_time fixed bin (71),            /* Time a VTOC Map was last locked/unlocked */
               2 total_volmap_lock_time fixed bin (71),        /* Total time volmap's were locked (integral) */
               2 total_vtoc_map_lock_time fixed bin (71),      /* Total time VTOC Maps were locked (integral) */
               2 n_volmap_locks fixed bin (35),                /* Number times a volmap was locked */
               2 v_vtoc_map_locks fixed bin (35),              /* Number times a vtoc_map was locked */

               2 array                (0 refer (pvt.n_entries)) like pvte;

     dcl     1 pvte                   based (pvtep) aligned,

               2 pvid                 bit (36),                /* physical volume ID */

               2 lvid                 bit (36),                /* logical volume ID */

               2 dmpr_in_use          (3) bit (1) unaligned,   /* physical volume dumper interlock */
               2 pad3 bit (6) unaligned,
```

```
          2 skip_queue_count   fixed bin (18) unsigned unaligned, /* number of times this pv skipped for per-proc allocation du
e to saturation */
          2 brother_pvtx fixed bin (8) unaligned,      /* next pvte in lv chain */


          2 devname            char (4),               /* device name */

          (2 device_type       fixed bin (8),          /* device type */
          2 logical_area_number fixed bin (8),         /* disk drive number */
          2 used               bit (1),                /* TRUE if this entry is used */
          2 storage_system     bit (1),                /* TRUE for storage system (vs io disk) */
          2 permanent          bit (1),                /* TRUE if cannot be demounted */
          2 testing            bit (1),                /* Protocol bit for read_disk$test */
          2 being_mounted      bit (1),                /* TRUE if the physical volume is being mounted */
          2 being_demounted    bit (1),                /* TRUE if the pysical volume is being demounted */
          2 check_read_incomplete bit (1),             /* page control should check read incomplete */
          2 device_inoperative bit (1),                /* TRUE if disk_control decides dev busted */
          2 rpv                bit (1),                /* TRUE if this is the root physical volume */
          2 pad1               bit (1),
          2 salv_required      bit (1),                /* TRUE if accepting this vol required salvaging */
          2 being_demounted2   bit (1),                /* No more vtoc I/O during demount */
          2 vol_trouble        bit (1),                /* Salvage on next accept */
          2 vacating           bit (1),                /* don't put new segs on this vol */
          2 hc_part_used       bit (1),                /* HC part set up by init_pvt */
          2 volmap_lock_notify bit (1) unal,           /* TRUE if notify required when volmap lock is unlocked */
          2 volmap_idle_notify bit (1) unal,           /* TRUE if notify required when volmap state is idle */
          2 vtoc_map_lock_notify bit (1) unal,         /* TRUE if notify required when vtoc map lock is unlocked */


          2 n_free_vtoce       fixed bin (17),         /* number of free VTOC entries */
          2 vtoc_size          fixed bin (17),         /* size of the VTOC part of the disk - in records */

          2 dbmrp              (2) bit (18),           /* rel ptr to dumber bit maps for this volume */

          2 nleft              fixed bin (17),         /* number of records left */
          2 totrec             fixed bin (17)) unaligned, /* Total records in this map */

          2 dim_info           bit (36),               /* Information peculiar to DIM */

          2 curn_dmpr_vtocx    (3) fixed bin unaligned,/* current vtocx being dumped */
          2 n_vtoce            fixed bin unaligned,    /* number of vtoce on this volume */

          2 volmap_seg_sdw     fixed bin (71),         /* SDW describing volmap_seg */


          2 volmap_astep       ptr unal,               /* Packed pointer to ASTE for volmap_seg */

          2 volmap_offset      bit (18) unal,          /* Offset in volmap_seg of volume map */
          2 vtoc_map_offset    bit (18) unal,          /* Offset in volmap_seg of VTOC map */


          2 volmap_lock        bit (36) aligned,       /* Lock on volume map operations */

          2 vtoc_map_lock      bit (36) aligned,       /* Lock on VTOC map operations */

          2 volmap_stock_ptr   ptr unal,               /* Packed pointer to record stock */

          2 vtoce_stock_ptr    ptr unal,               /* Packed pointer to VTOCE stock */

          2 volmap_async_state fixed bin (17) unaligned, /* Asynchronous update state of Volume Map */
```

```
          2 volmap_async_page  fixed bin (17) unaligned; /* Page number for asynchronous update */

dcl      (VOLMAP_ASYNC_IDLE     init (0),               /* for volmap_async_state */
          VOLMAP_ASYNC_READ     init (1),
          VOLMAP_ASYNC_WRITE    init (2)) fixed bin int static options (constant);


/*       END INCLUDE FILE ...pvt.incl.pl1 */
```

```
"BEGIN INCLUDE FILE stock_seg.incl.alm

	"Created 02/23/82  1218.9 est Tue by convert_include_file,
	"   Version of 12/01/81  1540.3 est Tue.

	"Made from >user_dir_dir>Multics>Bongiovanni>htd>nsd>stock_seg.incl.pl1,
	"   modified 02/23/82  1218.9 est Tue

"
"	Structure stock_seg
"
	equu	stock_seg_size,28


	equ	stock_seg.meters,0		" LEVEL 2


	equ	stock_seg.free,27		" UPPER

"
"	Structure record_stock
"

	equ	record_stock.pvtep,0

	equ	record_stock.n_in_stock,1	" UPPER
	equ	record_stock.n_volmap_pages,1 " LOWER

	equ	record_stock.n_free_in_stock,2 " UPPER
	equ	record_stock.n_os_in_stock,2	" LOWER

	equ	record_stock.low_threshold,3	" UPPER
	equ	record_stock.high_threshold,3 " LOWER

	equ	record_stock.target,4		" UPPER
	equ	record_stock.stock_offset,4	" LOWER

	equ	record_stock.n_words_in_stock,5 " UPPER
	equ	record_stock.search_index,5	" LOWER

	equ	record_stock.volmap_page,6	" LEVEL 2

	equ	record_stock.n_free,6		" UPPER
	equ	record_stock.baseadd,6		" LOWER

	equ	record_stock.stock,0		" UPPER

"
"	Structure vtoce_stock
```

```
"
	equ	vtoce_stock.pvtep,0

	equ	vtoce_stock.n_in_stock,1	" UPPER
	equ	vtoce_stock.n_free_in_stock,1	" LOWER

	equ	vtoce_stock.stock,2		" UPPER

"
"	Structure rsmeters
"
	equ	rsmeters_size,27


	equ	rsmeters.lock_nowait_calls,0
	equ	rsmeters.lock_nowait_fails,1
	equ	rsmeters.lock_wait_calls,2
	equ	rsmeters.lock_wait_fails,3
	equ	rsmeters.read_complete_calls,4
	equ	rsmeters.post_stock_os_calls,5
	equ	rsmeters.total_cpu_overhead,6 " DOUBLE
	equ	rsmeters.low_thresh_detected,8
	equ	rsmeters.high_thresh_detected,9
	equ	rsmeters.low_thresh_fails,10
	equ	rsmeters.withdraw_stock_steps,11
	equ	rsmeters.withdraw_stock_losses,12
	equ	rsmeters.n_withdraw_attempt,13
	equ	rsmeters.n_withdraw_range,14
	equ	rsmeters.n_pages_withdraw_stock,15
	equ	rsmeters.n_pages_withdraw_async,16
	equ	rsmeters.n_v_withdraw_attempts,17
	equ	rsmeters.withdraw_volmap_steps,18
	equ	rsmeters.deposit_stock_steps,19
	equ	rsmeters.deposit_stock_losses,20
	equ	rsmeters.n_deposit_attempt,21
	equ	rsmeters.n_pages_deposit_stock,22
	equ	rsmeters.n_pages_deposit_volmap,23
	equ	rsmeters.n_v_deposit_attempts,24
	equ	rsmeters.reset_os_calls,25
	equ	rsmeters.reset_os_losses,26

"END INCLUDE FILE stock_seg.incl.alm
```

```
/*  START OF:        stock_seg.incl.pl1              *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */

     dcl      stock_segp               ptr;
     dcl      record_stockp            ptr;
     dcl      vtoce_stockp             ptr;
     dcl      stock_seg$               ext;

     dcl      n_in_record_stock        fixed bin;
     dcl      n_in_vtoce_stock         fixed bin;

     dcl      1 stock_seg              aligned based (stock_segp),

                2 meters               aligned like rsmeters,

                2 free                 bit (18) unal;          /* offset of first free word in segment */

     dcl      1 record_stock           aligned based (record_stockp),

                2 pvtep                ptr unal,               /* PVTE for this stock */

                2 n_in_stock           fixed bin (18) uns unal,/* Max number of addresses in stock */
                2 n_volmap_pages       fixed bin (18) uns unal,/* Number of pages in Volume Map */

                2 n_free_in_stock      fixed bin (18) uns unal,/* Number addresses currently free */
                2 n_os_in_stock        fixed bin (18) uns unal,/* Number addresses currently out-of-service */

                2 low_threshold        fixed bin (18) uns unal,/* Low threshold for withdrawing from volmap */
                2 high_threshold       fixed bin (18) uns unal,/* High threshold for depositing to volmap */

                2 target               fixed bin (18) uns unal,/* Target for stock */
                2 stock_offset         bit (18) unal,          /* Offset of stock in this structure */

                2 n_words_in_stock     fixed bin (18) uns unal,/* Number of words = Number of entries / 2 */
                2 search_index         fixed bin (18) uns unal,/* Roving pointer */

                2 volmap_page          (record_stock.n_volmap_pages) aligned,
                  3 n_free             fixed bin (18) uns unal,/* Number free records in this volmap page */
                  3 baseadd            fixed bin (18) uns unal,/* First record address described by this page */

                2 stock                (n_in_record_stock refer (record_stock.n_in_stock)) bit (18) unal; /* Stock array of addresses *
/
                                                               /* bit 0 ON => out-of-service */

     dcl      1 vtoce_stock            aligned based (vtoce_stockp),
                2 pvtep                ptr unal,               /* PVTE for this stock */
                2 n_in_stock           fixed bin (18) uns unal,/* Max number addresses in stock */
                2 n_free_in_stock      fixed bin (18) uns unal,/* Number addresses currently free */
                2 stock                (n_in_vtoce_stock refer (vtoce_stock.n_in_stock)) fixed bin (18) uns unal; /* Stock array of VTO
CE indices */

     dcl      1 rsmeters               aligned based,
```

```
2 lock_nowait_calls  fixed bin (35),        /* Number calls to lock_wired_nowait */
2 lock_nowait_fails  fixed bin (35),        /* Number calls which did not acquire lock */
2 lock_wait_calls    fixed bin (35),        /* Number calls to lock_wired_wait */
2 lock_wait_fails    fixed bin (35),        /* Number calls which waited */
2 read_complete_calls fixed bin (35),       /* Number times read complete (async) detected */
2 post_stock_os_calls fixed bin (35),       /* Number times write complete (async) detected */
2 total_cpu_overhead fixed bin (71),        /* Total overhead in all routines */
2 low_thresh_detected fixed bin (35),       /* Number of times stock below low threshold */
2 high_thresh_detected fixed bin (35),      /* Number of times stock above high threshold */
2 low_thresh_fails   fixed bin (35),        /* Number of times no records in volmap */
2 withdraw_stock_steps fixed bin (35),      /* Number steps thru stock in withdraw */
2 withdraw_stock_losses fixed bin (35),     /* Number lockless losses */
2 n_withdraw_attempt fixed bin (35),        /* Number attempts to withdraw a page */
2 n_withdraw_range   fixed bin (35),        /* Number attempts to withdraw within range */
2 n_pages_withdraw_stock fixed bin (35),    /* Number pages withdrawn from stock */
2 n_pages_withdraw_async fixed bin (35),    /* Number pages withdrawn from volmap */
2 n_v_withdraw_attempts fixed bin (35),     /* Number attempts to withdraw from volmap */
2 withdraw_volmap_steps fixed bin (35),     /* Number steps thru volmap in withdraw */
2 deposit_stock_steps fixed bin (35),       /* Number steps thru stock in deposit */
2 deposit_stock_losses fixed bin (35),      /* Number lockless losses */
2 n_deposit_attempt  fixed bin (35),        /* Number attempts to deposit a page */
2 n_pages_deposit_stock fixed bin (35),     /* Number pages deposited to stock */
2 n_pages_deposit_volmap fixed bin (35),    /* Number pages deposited to volmap */
2 n_v_deposit_attempts fixed bin (35),      /* Number attempts to deposit to volmap */
2 reset_os_calls     fixed bin (35),        /* Number calls to reset_os */
2 reset_os_losses    fixed bin (35);        /* Number lockless losses */


/*  END OF:       stock_seg.incl.pl1              *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */
```

```
/*  START OF:       volmap.incl.pl1                     * * * * * * * * * * * * * * * * * */

     dcl      volmapp                  ptr;
     dcl      volmap_sectionp          ptr;

     dcl      volmap_section_map_size fixed bin;             /* Size of one volmap section map in words */

     dcl      1 volmap                 aligned based (volmapp),
                2 section              (0:1) aligned like volmap_section;

     dcl      1 volmap_section         aligned based (volmap_sectionp),
                2 pvid                 bit (36),                /* PVID to catch errors */
                2 checksum             bit (36),
                2 baseadd              fixed bin (17) unal,     /* First record number in map */
                2 map_n_words          fixed bin (17) unal,     /* Number of words in this section's map */
                2 map_n_records        fixed bin (17) unal,     /* Number of records in map */
                2 map_n_free           fixed bin (17) unal,     /* Number of records currently free */
                2 map                  (volmap_section_size refer (volmap_section.map_n_words)) bit (36) aligned;

/*  END OF:         volmap.incl.pl1                     * * * * * * * * * * * * * * * * * */
```

```
/*        BEGIN INCLUDE FILE ...vtoce.incl.pl1 ... last modified Feb 1979 to increase quota/used */

/* Template for a VTOC entry. Length = 192 words. (3 * 64). */

dcl  vtocep ptr;

dcl 1 vtoce based (vtocep) aligned,


    (2 pad1 bit (36),

    2 uid bit (36),                                   /* segment's uid - zero if vtoce is free */

    2 msl bit (9),                                    /* maximum segment length in 1024 word units */
    2 csl bit (9),                                    /* current segment length - in 1024 word units */
    2 records bit (9),                                /* number of records used by the seg in second storage */
    2 pad2 bit (9),

    2 dtu bit (36),                                   /* date and time segment was last used */

    2 dtm bit (36),                                   /* date and time segment was last modified */

    2 nqsw bit (1),                                   /* no quota switch - no checking for pages of this seg */
    2 deciduous bit (1),                              /* true if hc_sdw */
    2 nid bit (1),                                    /* no incremental dump switch */
    2 dnzp bit (1),                                   /* Dont null zero pages */
    2 gtpd bit (1),                                   /* Global transparent paging device */
    2 per_process bit (1),                            /* Per process segment (deleted every bootload) */
    2 damaged bit (1),                                /* TRUE if contents damaged */
    2 fm_damaged bit (1),                             /* TRUE if file map damaged */
    2 pad3 bit (10),
    2 dirsw bit (1),                                  /* directory switch */
    2 master_dir bit (1),                             /* master directory - a root for the logical volume */
    2 pad4 bit (16),                                  /* not used */

    2 fm_checksum bit (36),                           /* Checksum of used portion of file map */

    2 quota (0:1) fixed bin (18) unsigned,            /* sec storage quota - (0) for non dir pages */

    2 used (0:1) fixed bin (18) unsigned,             /* sec storage used  - (0) for non dir pages */

    2 received (0:1) fixed bin (18) unsigned,         /* total amount of storage this dir has received */

    2 trp (0:1) fixed bin (71),                       /* time record product - (0) for non dir pages */

    2 trp_time (0:1) bit (36),                        /* time time_record_product was last calculated */



    2 fm (0:255) bit (18),                            /* file map - 256 entries - 18 bits per entry */
```

```
     2 pad6 (10) bit (36),                                /* not used */

     2 ncd bit (1),                                       /* no complete dump switch */
     2 pad7 bit (17),
     2 pad8 bit (18),

     2 dtd bit (36),                                      /* date-time-dumped */

     2 volid (3) bit (36),                                /* volume ids of last incremental, consolidated, and complete dumps */

     2 master_dir_uid bit (36),                           /* superior master directory uid */



     2 uid_path (0:15) bit (36),                          /* uid pathname of all parents starting after the root */

     2 primary_name char (32),                            /* primary name of the segment */

     2 time_created bit (36),                             /* time the segment was created */

     2 par_pvid bit (36),                                 /* physical volume id of the parent */

     2 par_vtocx fixed bin (17),                          /* vtoc entry index of the parent */
     2 branch_rp bit (18)) unaligned,                     /* rel pointer of the branch of this segment */

     2 cn_salv_time bit (36),                             /* time branch - vtoce connection checked */

     2 access_class bit (72),                             /* access class in branch */
     2 pad9 bit (36),
     2 owner bit (36);                                    /* pvid of this volume */

dcl  vtoce_parts (3) bit (36 * 64) aligned based (vtocep);

dcl 1 seg_vtoce based (vtocep) aligned,                   /* Overlay for vtoce of segments, which don't have quota */
    2 pad1 bit (7*36),
    2 usage fixed bin (35),                               /* page fault count: overlays quota */
    2 pad2 bit (184*36);

/*        END INCLUDE FILE vtoce.incl.pl1 */
```