Files-11 On-Disk Structure Specification

Andrew C. Goldstein
Advanced-11 Software Devlopment
ML3-4/E88   Ext. 2405

1-Apr-1978

Document   130-958-032-03

1.0 Scope

This document is a specification of the on-media structure that is used by Files-11. Files-11 is a general purpose file structure which is intended to be the standard file structure for all medium to large PDP-11 systems. Small systems such as RT-11 have been specifically excluded because the complexity of Files-11 would impose too great a burden on their simplicity and small size.

This document describes structure level 2 of Files-11, also referred to as ODS-2 (on-disk structure 2). It contains feature and reliability improvements over structure level 1 (ODS-1).

1.1 Conventions

All numerical values given in this document are in decimal radix, unless indicated otherwise.

Within the file structure are fields containing binary integers of various lengths. Unless otherwise indicated, all these fields are in the standard numerical format, which means that the most significant bits are stored in the highest numbered address.

In the descriptions of various structures on the disk, there are fields that are labeled "not used". These fields must be zero, so that they can be made non-zero for future use. Since they are reserved for future use, programs reading these structures should not assume that these fields are in fact zero.

2.0 Medium

Files-11 is a structure which is imposed on a medium. That medium must have certain properties, which are described in the following section. Generally speaking, block addressable storage devices such as disks and Dectape are suitable for Files-11; hence Files-11 structured media are generically referred to as disks.

2.1 Volume

The basic medium that carries a Files-11 structure is referred to as a volume. A volume (also often referred to as a unit) is defined as an ordered set of logical blocks. A logical block is an array of 512 8-bit bytes. The logical blocks in a volume are consecutively numbered from 0 to n-1, where the volume contains n logical blocks. The number assigned to a logical block is called its logical block number, or LBN. Files-11 is capable of describing volumes up to 2**32 blocks in size. In practice, a volume should be at least 100 blocks in size to be useful.

The logical blocks of a volume must be randomly addressable. The volume must also allow transfers of any length up to 65K bytes, in multiples of four bytes. When a transfer is longer than 512 bytes, consecutively numbered logical blocks are transfered until the byte count is satisfied. In other words, the volume can be viewed as a partitioned array of bytes. It must allow reads and writes of arrays of any length less than 65K bytes, provided that they start on a logical block boundary and that the length is a multiple of four bytes. When only part of a block is written, the contents of the remainder of that logical block will be undefined.

The logical blocks of a volume are grouped into clusters. The cluster is the basic unit of space allocation on the volume. Each cluster contains one or more logical blocks; the number of blocks in a cluster is known as the volume cluster factor, or storage map cluster factor.

A volume is identified as a Files-11 volume by the home block. The home block is located at a defined physical location on the volume, and is identified by the presence of checksums and predictable values. The home block is described in detail in section 5.1. To identify the volume, the home block contains a volume label, which is a string of up to 12 ASCII characters. The characters are restricted to the printing ASCII set (i.e., excluding control characters and rubout). Further, it is reccommended that volume labels be restricted to alphanumerics only to avoid conflicts with the command languages of supporting systems. The volume label of a volume may not be null.

2.2 Volume Sets

A volume set is a collection of related volumes that are normally treated as one logical device in the usual operating system concept. Each volume contains its own Files-11 structure; however, files on the various volumes in a volume set may be referenced with a relative volume number, which uniquely determines which volume in the set the file is located on.

A volume set has associated with it a structure name, which is an string of up to 12 ASCII characters which identifies the volume set. The character set limitations of the volume label also apply to the structure name. The structure name may not be null.

2.2.1 Tightly Coupled Volume Set -

A tightly coupled volume set is a volume set which is consistent and self identifying. The volume labels of the volumes making up the set must be unique within the set, and must be different from the structure name. Relative volume one of the set contains a file which lists the volume labels of all the volumes in the set, thus associating volume labels with relative volume numbers. Each volume is identified as being part of the set by carrying the structure name, its volume label, and its relative volume number.

2.2.2 Loosely Coupled Volume Set -

A loosely coupled volume set is a collection of volumes which is not self identifying. There is no file listing the volume labels. Only one file may cross from any one volume in the set to another, and files in the set which cross volumes may be processed only sequentially. Correct sequencing of the volumes that hold a particular file is the responsibility of the system operator. There are checks that will catch most handling errors, but they cannot be foolproof. The purpose of the loosely coupled volume set is to emulate multi-volume magtape, and allow a file to be read or written sequentially with only one volume mounted at a time.

3.0 Files

Any data in a volume or volume set that is of any interest (i.e., all blocks not available for allocation) is contained in a file. A file is an ordered set of virtual blocks, where a virtual block is an array of 512 8 bit bytes. The virtual blocks of a file are consecutively numbered from 1 to n, where n is the highest numbered block that has been allocated to the file. The number assigned to a virtual block is called (obviously) its virtual block number, or VBN. Virtual blocks are mapped to unique logical blocks in the volume set by Files-11. Virtual blocks may be processed in the same manner as logical blocks. Any array of bytes less than 65K in length may be read or written, provided that the transfer starts on a virtual block boundary and that its length is a multiple of four.

For most files, all VBN's less than or equal to the highest VBN allocated map to some LBN in the volume set. Such files are said to be dense. Files which are sparse contain virtual blocks which have not been allocated logical blocks.

3.1 File ID

Each file in a volume set is uniquely identified by a File ID. A File ID is a binary value consisting of 48 bits (3 PDP-11 words). It is supplied by the file system when the file is created, and must be supplied by the user whenever he wishes to reference a particular file.

The three words of the File ID are used as follows:

Word 1 File Number

Locates the file within a particular volume of the volume set. File numbers ordinarily lie in the range 1 through 65535. The set of file numbers on a volume is moderately (but not totally) dense; at any instant in time a file number uniquely identifies one file within that volume.

Word 2 File Sequence Number

Identifies the current use of an individual file number on a volume. File numbers are re-used; when a file is deleted its file number becomes available for future use for some other file. Each time a file number is re-used, a different file sequence number is assigned to distinguish the uses of that file number. The file sequence number is essential since it is perfectly legal for users to remember and attempt to use a File ID long after that file has been deleted.

Word 3 Relative Volume Number

Identifies which volume of a volume set the file is located on. If the volume in question is not a member of a volume set, then this word is zero. If the volume is part of a volume set, then the relative volume number, or RVN, lies in the range from 1 to 65535. In any context where a particular volume of a volume set can be identified as the "current volume", such as a file extension linkage, a relative volume number of zero means "the current volume". When a file is referred to in the context of the volume on which it lies, it should be referred to with a relative volume number of zero, regardless of the RVN that may be assigned to that volume.

File Number Extension

If the maximum number of files permitted on the volume (as recorded in the home block) is greater than 65535, then the high byte of the relative volume number becomes a high order extension to the file number. The volume set size is then limited to 255 volumes, while the range of allowable file numbers extends from 1 to 2**24-1. When 24 bit file numbers are used, the file system should not create files whose file number is an integer multiple of 65536 (i.e., whose low 16 bits are zero). Such file numbers will break existing PDP-11 software (such as FCS-11).

3.2 File Header

Each file on a Files-11 volume is described by a file header. The file header is a block that contains all the information necessary to access the file. It is not part of the file; rather, it is contained in the volume's index file. (The index file is described in section 5.1). The header block is organized into six areas, of which the first five are variable in size.

3.2.1 Header Area -

The information in the header area permits the file system to verify that this block is in fact a file header and, in particular, is the header being sought by the user. It contains the file number and file sequence number of the file, as well as its ownership and protection codes. This area also contains offsets to the other areas of the file header, thus defining their size.

3.2.2 Ident Area -

The ident area of a file header contains identification and accounting data about the file. Stored here are the primary name of the file, its creation date and time, revision count, date, and time, and expiration date.

3.2.3 Map Area -

The map area describes the mapping of virtual blocks of the file to the logical blocks of the volume. The mapping data consists of a list of retrieval pointers. Each retrieval pointer describes one group of consecutively numbered logical blocks that are allocated to the file. Retrieval pointers are arranged in the order of the virtual blocks they represent.

3.2.4 Access Control List -

The access control list is an optional area that contains a list of users that are allowed access to the file. The access control list makes it possible to describe user communities for a particular file that cannot be expressed with the regular protection classes.

3.2.5 Reserved Area -

This optional area is reserved for the use by CSS or special applications. It will not be used by standard Files-11 systems.

3.2.6 End Checksum -

The last two bytes of the file header contain a 16 bit additive checksum of the preceding 255 words of the file header. The checksum is used to help verify that the block is in fact a file header.

3.3 Multi-header Files

Since the file header is of fixed size, it is inevitable that for some files the mapping information will not fit in the allocated space. A file with a large amount of mapping data is therefore represented with a chain of file headers. Each header maps a consecutive set of virtual blocks; the extension linkage in the header area links the headers together in order of ascending virtual block numbers. The extension pointer in each file header is the File ID of the next header in sequence.

3.4 Multi-volume Files

Multiple headers are also needed for files that span volumes in a volume set. A header may only map logical blocks located on its volume; therefore a multi-volume file is represented by headers on all volumes that contain portions of that file. In a multi-volume file contained on a loosely coupled volume set, the File ID of the first header on each continuation volume always has the value 9,9,n, where n is the RVN of the volume on which the file starts plus the number of preceding volumes containing portions of the file.

3.5 File Header - Detailed Description

This section describes in detail the items contained in the file header. Each item is identified by a symbol which represents the offset address of that item within its area in the file header. Any item may be located in the file header by locating the area to which it belongs and then adding the value of its offset address. Users who concern themselves with the contents of file headers are strongly urged to use the offset symbols. The symbols may be defined in assembly language programs by calling and invoking the macro FHDL2$, which may be found in the macro library of any system that supports Files-11. Alternatively, one may find the macro in the file F11MAC.MAC, which may be obtained from the author.

3.5.1 Validity -

A valid file header is defined by the following rules:

1. The header checksum is correct, unless SC.CHK is set in H.SCHA, in which case the checksum word contains the value 125252.
2. The contents of H.IDOF is no less than the offset H.FOWN/2.
3. The four offset bytes are related in the manner (H.IDOF) <= (H.MPOF) <= (H.ACOF) <= (H.RSOF).
4. The high byte of H.FLEV contains the value 2.
5. The low byte of H.FLEV contains a value greater or equal to 1.
6. The word H.FNUM contains the file number.
7. The word H.FSEQ contains the file sequence number.

8. The high byte H.FRVN contains the extended part of the file number, if any.

9. The contents of the byte H.USE must be less than or equal to (H.ACOF) - (H.MPOF).

A deleted file header conforms to the format of a valid file header with the following exceptions:

1. SC.MDL is set in H.FCHA.

2. H.FNUM and H.FRVN contain zero.

3. The file header checksum contains zero.

3.5.2 Header Area Description -

The header area of the file header always starts at byte 0. It contains the basic information needed for checking the validity of accesses to the file.

3.5.2.1 H.IDOF - 1 Byte Ident Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the ident area. It defines the location of the ident area and the size of the header area.

3.5.2.2 H.MPOF - 1 Byte Map Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the map area. It defines the location of the map area and, together with H.IDOF, the size of the ident area.

3.5.2.3 H.ACOF - 1 Byte Access Control List Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the access control list. It defines the location of the ACL and, together with H.MPOF, the size of the map area.

3.5.2.4 H.RSOF - 1 Byte Reserved Area Offset

This byte contains the number of 16 bit words between the start of the header and the start of the reserved area. The reserved area will not be used by Files-11 itself, and may be used by CSS or special applications. Together with H.ACOF, this byte defines the size of the access control list. The size of the reserved area is implied by the contents of H.RSOF and the end of the header block.

The presence of the ident, map, ACL, and reserved areas is optional. Absence of any area is signalled not by a zero offset, but by the equality of the two offsets that define that area's size. All five areas are variable in length; implementations of Files-11 must check the length of a particular area before attempting to reference a particular entry.

3.5.2.5 H.FSEG - 2 Bytes Extension Segment Number

This word contains the value n, where this header is the n+1th header of the file; i.e., headers of a file are numbered sequentially starting with 0.

3.5.2.6 H.FLEV - 2 Bytes Structure Level and Version

The file structure level and version is used to identify different versions of Files-11 as they affect the structure of the file header. This permits upwards compatibility of file structures as Files-11 evolves, in that the structure level word identifies the version of Files-11 that created this particular file. This document describes structure level 2 of Files-11. The high byte of H.FLEV must contain the value 2. The low byte contains the version number, which must be greater or equal to 1. The version number will be incremented whenever compatible additions are made to the Files-11 structure that may be safely ignored by an old version of the file system. This document describes version 1 of structure level 2.

3.5.2.7 H.FNUM - 2 Bytes File Number

This word contains the file number of the file.

3.5.2.8 H.FSEQ - 2 Bytes File Sequence Number

This word contains the file sequence number of the file.

3.5.2.9 H.FRVN - 2 Bytes Relative Volume Number

This word is used to hold part of the third word of the File ID when appropriate. This word is usually referred to as the relative volume number. When used as such (i.e., to indicate the volume of a volume set), it is not recorded in the file header, since the RVN of a volume may change during a file's life, and the RVN portion of a File ID may be zero or non-zero, depending on the context. However, when the high byte of the RVN is used as an extension to the file number, then it is recorded in the high byte of this word. The low byte of H.FRVN is always zero.

3.5.2.10 H.EFNU - 2 Bytes Extension File Number

This word contains the file number of the next sequential extension header for this file. If there is no extension header, this word contains 0.

3.5.2.11 H.EFSQ - 2 Bytes Extension File Sequence Number

This word contains the file sequence number of the next sequential extension header for this file. If there is no extension header, this word contains 0.

3.5.2.12 H.ERVN - 2 Bytes Extension Relative Volume No.

This word contains the relative volume number of the volume in the volume set that contains the next sequential extension header for this file. If there is no extension header, or if the extension header is located on the same volume as this header, this word contains 0.

3.5.2.13 H.UFAT - 32 Bytes User File Attributes

This area is used by the record manager, or any other higher level access mechanism, to store information necessary for processing the file, e.g., record control data, end of file mark, etc.

3.5.2.14 H.FCHA - 4 Bytes File Characteristics
H.UCHA = H.FCHA+0 User Controlled Char.
H.SCHA = H.FCHA+1 System Controlled Char.

The file characteristics words contain the following flag bits:

UC.CON	Set if the file is logically contiguous; i.e., if for all virtual blocks in the file, virtual block i maps to logical block k+i on one volume for some constant k. This bit may be implicitly set or cleared by file system operations that allocate space to the file; the user may only clear it explicitly.
UC.CNB	Set if the file is allocated contiguous best effort; i.e., as contiguous as possible.
UC.DLK	Set if the file is deaccess-locked. This bit is used as a flag warning that the file was not properly closed and may contain inconsistent data. Access to the file is denied if this bit is set.
UC.RCK	Set if the file is to be read-checked. All read operations on the file, including reads of the file header(s), will be performed with a read, read-compare to assure data integrity.
UC.WCK	Set if the file is to be write-checked. All write operations on the file, including modifications of the file header(s), will be performed with a write, read-compare to assure data integrity.

UC.NID	Set if incremental dump (backup) is to be disabled for this file.
UC.WBC	Set if the file is to be write-back cached; i.e., if a cache is used for the file data, data written by a user is only written back to the disk when is it removed from the cache. Clear for write-through cache operation.

The second byte of the file characteristics words is historically known as the system controlled file characteristics. It contains the following flag bits, defined as referenced within the byte:

SC.MDL	Set if the file is marked for delete. If this bit is set, further accesses to the file are denied, and the file will be physically deleted when no users are accessing it.
SC.BAD	Set if there is a bad data block in the file. This bit is as yet unimplemented. It is intended for dynamic bad block handling.
SC.DIR	Set if the file is a directory.
SC.ACL	Set if an access control list exists for this file.
SC.CHK	This bit is set if the file header checksum was not computed. If this bit is set, the checksum word must contain the octal value 125252. This "feature" is for small systems that cannot afford the millisecond or two that it takes to compute the header checksum; its use is strongly discouraged.

3.5.2.15 - 2 Bytes (not used)

3.5.2.16 H.USE - 1 Byte Map Words in Use

This byte contains a count of the number of map area words currently in use.

3.5.2.17 H.PRIV - 1 Byte Accessor Privilege Level

This byte defines the lowest privilege level at which an accessor must be running in order to be allowed access to the file. Each privilege level is a two bit integer; zero refers to the lowest privilege and 3 is the highest. Privilege levels may be assigned separately to the basic file access modes, using the following bit assigment in this byte:

Read	Bits 0 - 1
Write	Bits 2 - 3
Execute	Bits 4 - 5
Delete	Bits 6 - 7

An operating system should map its privilege level coding onto this code in the smoothest manner possible. For example, the four access modes of VMS, user, supervisor, exec, and kernel, are coded as 0 through 3, respectively. A system such as RSX-11M which has only two levels (privileged and non-privileged), should map the two onto 3 and 0, respectively.

Privilege levels are meant to confine access to the contents of files to suitably trustworthy procedures. Thus, a user might be denied the ability to write a record structured file directly (on a virtual block basis), but would be permitted to write the file through the record manager, which would be suitably privileged.

For a record structured file, an appropriate set of privilege levels would be 0,2,0,0, expressed in the order read - write - execute - delete.

3.5.2.18 H.FOWN - 4 Bytes File Owner UIC
H.PROG = H.FOWN+0 Programmer (Member) Number
H.PROJ = H.FOWN+2 Project (Group) Number

These words contain the binary user identification code (UIC) of the owner of the file. The file owner is usually (but not necessarily) the creator of the file.

3.5.2.19 H.FPRO - 2 Bytes File Protection Code

This word controls what access all users in the system may have to the file. Accessors of a file are categorized according to the relationship between the UIC of the accessor and the UIC of the owner of the file. Each category is controlled by a four bit field in the protection word. The category of the accessor is selected as follows:

System	Bits 0 - 3
	The accessor is subject to system protection if the project number of the UIC under which he is running is 10 octal or less.
Owner	Bits 4 - 7
	The accessor is subject to owner protection if the UIC under which he is running exactly matches the file owner UIC.
Group	Bits 8 - 11
	The accessor is subject to group protection if the project number of his UIC matches the project number of the file owner UIC.
World	Bits 12 - 15
	The accessor is subject to world protection if he does not fit into any of the above categories.

Four types of access are defined in Files-11: read, write, execute, and delete. Each four bit field in the protection word is bit encoded to permit or deny any combination of the four types of access to that category of accessors. Setting a bit denies that type of access to that category. The bits are defined as follows (these values apply to a right-justified protection field):

FP.RDV	Deny read access
FP.WRV	Deny write access
FP.EXE	Deny execute access
FP.DEL	Deny delete access

When a user attempts to access a file, protection checks are performed in all the categories to which he is eligible, in the order system - owner

- group - world. The user is granted access to the file if any of the categories to which he is eligible grants him access.

Recommended defaults for file protection for an "open shop" system are [RWD,RWD,RW,R] (expressed in the order of system - owner - group - world, where each letter denotes the presence of that permission). Observe that only files which contain executable programs should have execute protection turned on. Recommended defaults for a "closed shop" system are [RWD,RWD,R,].

3.5.2.20 H.RPRO - 2 Bytes Record Protection Code

This word controls what access all users in the system may have to the records in a file. Accessors are categorized into System, Owner, Group, and World in the same manner as for file protection. The record protection word is likewise divided into four four bit fields to control each accessor category. The four bits in the record protection field are defined as follows:

RP.RDV	Deny reading records
RP.WRV	Deny writing new records
RP.UPD	Deny writing existing records
RP.DEL	Deny deleting records

Recommended defaults for record protection for an "open shop" system are [RWUD,RWUD,RWUD,R]. Recommended defaults for a "closed shop" system are [RWUD,RWUD,R,].

3.5.2.21 - 4 Bytes (not used)

3.5.2.22 H.SEMK - 4 Bytes Security Mask

The security mask is a bit encoded field that represents information categories that may be present in this file. An accessor also carries a security mask that represents information categories that he may possess. To read a file, the accessor's security mask must be a superset of the security mask of the file; to write a file, the security mask of the file must be equal to that of the accessor. (Technically, in security mask protocols, the mask of the file must be a superset of that of the writer, but since Files-11 systems do not allow writing without read permission, both conditions apply for a writer.)

Individual bits in the security mask are defined system wide by the system manager. The installation manager is responsible for ensuring consistency and coherency of security masks when volumes are used on multiple operating systems.

Note that the traditional security level system (confidential, secret, etc.) can be achieved by a unary encoding in several bits of the mask.

3.5.2.23 S.HDHD - 74 Bytes Size of Header Area

This symbol represents the total size of the header area containing all of the above entries.

3.5.3 Ident Area Description -

The ident area of the file header begins at the word indicated by H.IDOF. It contains identification and accounting data about the file.

3.5.3.1 I.FNAM - 20 Bytes File Name

This area contains the name of the file in ASCII. A dot separates name from type, and a semicolon separates type from version; both are always present. If the name is shorter than 20 bytes, it is padded with blanks; if it is longer, it is truncated.

3.5.3.2 I.RVNO - 2 Bytes Revision Number

This word contains the revision count of the file in binary. The revision count is the number of times the file has been accessed for write.

3.5.3.3 I.CRDT - 8 Bytes Creation Date and Time

These eight bytes contain the date and time at which the file was created. The time is expressed in the standard internal time format, which is a 64 bit integer representing tenths of microseconds elapsed since midnight, 17 November 1858.

3.5.3.4 I.RVDT - 8 Bytes Revision Date and Time

The revision time is the time at which the file was last deaccessed after being accessed for write. It is expressed as the same format as the creation time above.

3.5.3.5 I.EXDT - 8 Bytes Expiration Date and Time

These eight bytes contain the date and time at which the file becomes eligible to be deleted. The format is the same as that of the creation and revision times above.

3.5.3.6 I.BKDT - 8 Bytes Backup Date and Time

These eight bytes contain the date and time at which the file was last backed up. The format is the same as the other dates and times.

3.5.3.7 I.ULAB - 80 Bytes User Label

This optional area contains any label a user may wish to associate with the file.

3.5.3.8 S.IDHD - 116 Bytes Size of Ident Area

This symbol represents the size of the ident area containing all of the above entries.

3.5.4 Map Area Description -

The map area of the file header starts at the word indicated by H.MPOF. It contains the information necessary to map the virtual blocks of the file to the logical blocks of the volume. This area contains the retrieval pointers that actually map the virtual blocks of the file to the logical blocks of the volume. Each retrieval pointer describes a consecutively numbered group of logical blocks which is allocated to the file. The count field contains the binary value n to represent a group of n+1 logical blocks. The logical block number field contains the logical block number of the first logical block in the group. Thus each retrieval pointer maps virtual blocks j through j+n into logical blocks k through k+n, respectively, where j is the total number plus one of virtual blocks represented by all preceding retrieval pointers in this and all preceding headers of the file, n is the value contained in the count field, and k is the value contained in the logical block number field. Observe that j, k, and n+1 must always be integer multiples of v, the volume cluster factor.

If the LBN field of a retrieval pointer contains all ones (i.e., points to block 2**22-1 or 2**32-1), then that retrieval pointer represents an unallocated portion of a sparse file. The count field describes the number of unallocated virtual blocks in the normal manner.

There are four formats of retrieval pointers, identified by escape codes. The different formats may be intermixed within a file header.

Format 0 - two bytes

```
|--|------------------|
|00|    Placement     |
|--|------------------|
```

Retrieval pointer format 0 is used to store placement data in the file header. It describes the placement control supplied with the allocation that created the following retrieval pointer, allowing the placement of a file to be replicated when the file is copied or backed up and restored. The coding of the placement data is at present undefined. Format 0 is identified by bits 15 and 14 of the first word being set to 00.

Format 1 - four bytes.

```
|--|-------|----------|
|01| High  |  Count   |
|--|-     -|----------|
|    Low Order LBN    |
|---------------------|
```

Retrieval pointer format 1 provides an 8 bit count field and a 22 bit LBN field. It is therefore capable of representing a group of up to 256 blocks on a volume up to 2**22 blocks in size. Format 1 is identified by bits 15 and 14 of the first word being set to 01.

Format 2 - six bytes.

```
|--|------------------|
|10|      Count       |
|--|------------------|
|                     |
|-        LBN        -|
|                     |
|---------------------|
```

Retrieval pointer format 2 provides a 14 bit count field and a 32 bit LBN field. It is capable of representing a group of up to 16384 blocks on a volume up to 2**32 blocks in size. Format 2 is identified by bits 15 and 14 of the first word being set to 10.

Format 3 - eight bytes.

```
|--|------------------|
|11|       High       |
|--|-                -|
|   Low Order Count   |
|---------------------|
|                     |
|-        LBN        -|
|                     |
|---------------------|
```

Retrieval pointer format 3 provides a 30 bit count field and a 32 bit LBN field. It is capable of describing a group of up to 2**30 blocks on a volume up to 2**32 blocks in size. Format 3 is identified by bits 15 and 14 of the first word being set to 11.

3.5.5 Access Control List -

The access control list starts at the word indicated by the byte H.ACOF. Note that the entire ACL must be contained in the primary header of the file, and is thus limited to about 65 entries. Each access control list entry consists of a control word which identifies the type of the entry and contains the access rights given by the list entry. Following the control word is a value field whose size and interpretation depends on the type code of the ACL entry.

```
|---------|---------|----|-|--|---------|
!  RPRO   |  FPRO   | AM |O|  !  Type   !
|---------|---------|----|-|--|---------|
!                                       !
|--                Value              --!
!                                       !
|---------------------------------------|
```

The four bit type field controls the size the interpretation of the entry's value field, and to some extent, the interpretation of the entry. The type field may assume one of the following values:

A.UIC	The value field is a 4 byte UIC. The ACL entry is applicable to the accessor if the accessor UIC matches the value field.
A.GRP	The value field is a 2 byte group number. The ACL entry is applicable if the group number of the accessor UIC matches the value field.
A.MEM	The value field is a 2 byte member number. The ACL entry is applicable if the member number of the accessor UIC matches the value field.
A.PSWD	The value field is an 8 byte password, hashed by some algorithm to be determined. The ACL entry is applicable if a password supplied by the accessor matches the value field (under hashing).
A.ACF	The value field is the file ID (6 bytes) of an access control file, whose contents are access control list entries. The access rights granted by this ACL entry are the intersection of the rights coded in this entry and the rights granted by the entries of the access control file.

The one bit O field, when set, grants ownership privileges (change protection, etc.) as part of the access rights granted by this ACL entry.

The two bit AM field specifies the least privileged access mode permitted to access the file.

The four bit FPRO field specifies the file protection granted by the ACL entry, if the accessor is eligible. Its contents are interpreted in the same way as the H.FPRO field of the file header.

The four bit RPRO field specifies the record protection granted by the ACL entry, if the accessor is eligible. Its contents are interpreted in the same way as the H.RPRO field of the file header.

Note that the access control list augments the permissions of the file: i.e., it grants permission rather than restricting it. This means that ignoring the ACL does not compromise file protection.

3.5.6 Reserved Area -

The reserved area of the file header starts at the word indicated by the byte H.RSOF. This area is not used by standard Files-11 file managers, but is available for use by CSS and special applications.

3.5.7 End Checksum Description -

The header check sum occupies the last two bytes of the file header. It is verified every time a header is read, and is recomputed every time a header is written. If the bit SC.CHK is set in the system controlled characteristics word H.SCHA, then the checksum has not been computed, and the checksum word must contain the octal value 125252.

3.5.7.1 H.CKSM - 2 Bytes Block Checksum

This word is a simple additive checksum of all other words in the block. It is computed by the following PDP-11 routine or its equivalent:

```
MOV Header-address,R0
CLR R1
MOV #255.,R2
```

```
        10$:        ADD (R0)+,R1
                    SOB R2,10$
                    MOV R1,(R0)
```

3.6 File Header Layout

The following is a graphical layout of the fields in the file header.

Header Area

```
              |------------------|------------------|
H.MPOF        |  Map Area Offset | Ident Area Offset |      H.IDOF
              |------------------|------------------|
H.RSOF        | Resv. Area Offset |  ACL Area Offset  |     H.ACOF
              |------------------|------------------|
              |          File Segment Number          |     H.FSEG
              |--------------------------------------|
              |          File Structure Level         |     H.FLEV
              |--------------------------------------|
              |              File Number              |     H.FNUM
              |--------------------------------------|
              |          File Sequence Number         |     H.FSEQ
              |--------------------------------------|
              |         Relative Volume Number        |     H.FRVN
              |--------------------------------------|
              |         Extension File Number         |     H.EFNU
              |--------------------------------------|
              |       Extension File Seq. Num.        |     H.EFSQ
              |--------------------------------------|
              |             Extension RVN             |     H.ERVN
              |--------------------------------------|
              |                                      |      H.UFAT
              |                                      |
              |                                      |
              |          User Attribute Area          |
              |                                      |
              |                                      |
              |                                      |
              |--------------------------------------|      H.FCHA
H.SCHA        |                                      |      H.UCHA
              |--        File Characteristics      --|
              |                                      |
              |--------------------------------------|
              |              (not used)               |
              |------------------|------------------|
H.PRIV        |   Access Level   | Map Words in Use |       H.USE
              |------------------|------------------|       H.FOWN
              |                                      |      H.PROG
              |--          File Owner UIC          --|
              |                                      |      H.PROJ
              |--------------------------------------|
              |            File Protection            |     H.FPRO
              |--------------------------------------|
              |           Record Protection           |     H.RPRO
              |--------------------------------------|
              |                                      |
              |--            (not used)            --|
              |                                      |
              |--------------------------------------|
              |                                      |      H.SEMK
              |--          Security Mask           --|
        |                                       |
        |---------------------------------------|      S.HDHD
```

Ident Area

```
        |---------------------------------------|
        |                                       |      I.FNAM
        |--                                   --|
        |                                       |
        |--                                   --|
        |               File Name               |
        |--                                   --|
        |                                       |
        |--                                   --|
        |                                       |
        |---------------------------------------|
        |            Revision Number            |      I.RVNO
        |---------------------------------------|
        |                                       |      I.CRDT
        |--                                   --|
        |                                       |
        |--           Creation Date           --|
        |                                       |
        |--                                   --|
        |                                       |
        |---------------------------------------|
        |                                       |      I.RVDT
        |--                                   --|
        |                                       |
        |--           Revision Date           --|
        |                                       |
        |--                                   --|
        |                                       |
        |---------------------------------------|
        |                                       |      I.EXDT
        |--                                   --|
        |                                       |
        |--          Expiration Date          --|
        |                                       |
        |--                                   --|
        |                                       |
        |---------------------------------------|
        |                                       |      I.BKDT
        |--                                   --|
        |                                       |
        |--            Backup Date            --|
        |                                       |
        |--                                   --|
        |                                       |
        |---------------------------------------|
        |                                       |      I.ULAB
        |                                       |
        |                                       |
        |            User File Label            |
        |                                       |
        |                                       |
        |                                       |
        |---------------------------------------|   S.IDHD
```

Map, ACL, and Reserved Areas

```
|---------------------------------------|
|                                       |
|                                       |
|                                       |
|          Retrieval Pointers           |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|                                       |
|                                       |
|                                       |
|          Access control List          |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|                                       |
|                                       |
|                                       |
|            Reserved Area              |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|         File Header Checksum          |     H.CKSM
|---------------------------------------|
```

4.0 Directories

Files-11 provides directories to allow the organization of files in a meaningful way. While the File ID is sufficient to locate a file uniquely on a volume set, it is hardly mnemonic. Directories are files whose function is to associate symbolic names with File ID's.

The directory format also contains hooks for extensions in future systems. One of these is a construct known as a symbolic link. A symbolic link allows a directory to contain a pointer to a file which is not on the same volume set, and can therefore not be represented by a File ID. A symbolic link therefore associates the file name with another ASCII string.

4.1 Directory Heirarchies

Since directories are files with no special attributes, directories may list files that are in turn directories. Thus the user may construct directory heirarchies of arbitrary depth and complexity to structure his files as he pleases.

4.1.1 Two Level Directory Heirarchy -

Implementations of Files-11 on existing PDP-11 systems all support a two level directory heirarchy which is tied in with the user identification mechanism of the operating system. Each UIC known to the system is associated with a user file directory (UFD). References to files that do not specify a directory are generally defaulted to the UFD associated with the user's UIC. The syntax used to refer to UIC's is the same as that used to identify the directory in a file name string. The construct "[n,m]" is used to refer to group number n, member number m. All UFD's are listed in the volume's master file directory (MFD) under a file name constructed from the directory string. (See section 5.0 for a description of the MFD.) A string of "[n,m]" associates with a directory name of "nnnmmm.DIR;1", where nnn and mmm are n and m padded out to three digits each with leading zeroes. Note that all number conversions are done in octal.

Two points should be noted here. The UFD structure described here is not intrinsically part of the Files-11 on-disk structure; rather, it is a convenient cataloging system applied by various operating systems. Also, there is no hard and fast relationship between the owner UIC of a file and the UFD in which it is listed. Generally, they will correspond, but not necessarily.

4.1.2 Multi-level Directory Heirarchy -

New implementations of Files-11 use a multi-level directory heirarchy, where the first level below the MFD is referred to as the user file directory (UFD) and subsequent levels are referred to as sub file directories (SFD's). Users are identified at the command level by ASCII names; the system translates user names into UIC's internally. Thus MFD entries will correspond to the ASCII user names. A directory specifier will have the format "[name1.name2.name3. ...]". Each name in the list translates to a directory file name of the form "name.DIR;1" and is searched for in the current directory level.

Observe that the directory protocol is not tied to the structure level of the disk. Thus new systems will always have to handle the "[n,m]" construct, which maps to a UFD name of "nnnmmm.DIR;1" and provides only two levels of directory. Old systems will not be able to handle volumes

4.1.3 Multi-Volume Directory Structure -

In a volume set, the MFD for the all of the user files on the volume set is the MFD of relative volume 1. Its entries can point to UFD's located on any volume in the set, whose entries can in turn point to files and sub directories on any volume in the set. The MFD's of the remaining volumes in the set only list the reserved files on each volume.

The assignment of volumes to specific directories and files is not covered by this specification. Different systems may implement different policies to trade off factors such as performance, reliability, and separability. Optimizing for performance, for example, usually means scattering the files as randomly as possible across the volume set to make the most use of the available multiple positioners. Maximum separability (the ability to make use of only part of the volume set) is achieved by locating files on the same volume as their directories, and possibly entering the directories in the MFD's of the volumes on which they reside.

4.2 Directory Protection

For directory operations, the record protection field is interpreted specially by the directory manager. The four bits (described in the section on record protection) are interpreted as follows:

RP.RDV	Deny lookups
RP.WRV	Deny entering new files
RP.UPD	Deny entering new versions of files
RP.DEL	Deny removing files

By setting the accessor privilege level of a directory file appropriately, the system (or user) may prevent users from rummaging through the directory using the normal file access methods.

If record protection is not present for a directory file, then the basic file access protection is used if it exists. Lookups require

4.3 Directory Structure

A directory is a contiguous file, organized as a sequential file with variable length records, with the attribute set that records do not cross block boundaries, and no carriage control attributes. Directory entries within each block are packed together to conform to the variable length record format; a -1 byte count signals the end of records for that block. (See section 6 for a discussion of record formats.) The entries in a directory are sorted alphabetically, permitting the use of an optimized search. Entries which are multiple versions of the same name and type are arranged in order of decreasing version number to optimize version related operations. Each directory record consists of the following:

```
|-------------------------------------|
|          Record Byte Count          |
|-------------------------------------|
|            Version Limit            |
|-----------------|-------------------|
| Name Byte Count |       Flags       |
|-----------------|-------------------|
|                                     |
|                                     |
|                                     |
|          File Name String           |
|                                     |
|                                     |
|                                     |
|-------------------------------------|
|                                     |
|                                     |
|             Value Field             |
|                                     |
|                                     |
|-------------------------------------|
```

Count	This two byte field is the standard byte count field of a variable length record.
Limit	This word contains the maximum number of versions that are to be retained for this name and type. An attempt to enter more versions than the limit will result in the deletion of the least recent version, or an error return, at the implementing system's option.
Flags	This byte contains the type code of the directory entry and assorted flag bits. The type code is

contained in the three low bits of the flags byte. It is one of the following values:

DV.FID The value field is a list of version numbers and 48 bit File Id's.

DV.SLK The value field is a symbolic link string.

The following flag bits are defined:

DF.PRV Set if the preceding directory record contains the same name and type as this one.

DF.NXV Set if the next directory record contains the same name and type as this one.

Name This field contains the file name and type in ASCII, separated by a a dot. The dot is present even if either name, or type, or both, are null. Only upper case alphabetic and numeric characters may be present in the name and type. If the length of the name is odd, it is padded with a single null.

Value This field contains the "value" of the directory entry: i.e., the information returned to the user from a lookup operation. If the directory record is a File ID list (the type field is DV.FID), the value field is a list of version numbers and corresponding file ID's, appearing in descending order by version number. The number of entries in the list is deduced from the record byte count.

```
|---------------------------------------|
!            Version Number             !
|---------------------------------------|
!                                       !
|--                                   --|
!                File ID                !
|--                                   --|
!                                       !
|---------------------------------------|
!            Version Number             !
|---------------------------------------|
!                                       !
|--                                   --|
!                File ID                !
|--                                   --|
!                                       !
|---------------------------------------|
!                                       !
```

```
                        :
                        :
                        :
                        :
                        :
  |                                       |
  |---------------------------------------|
  |            Version Number             |
  |---------------------------------------|
  |                                       |
  |--                                   --|
  |               File ID                 |
  |--                                   --|
  |                                       |
  |---------------------------------------|
```

Version   This word contains the version number of the directory entry in binary. Version numbers must lie in the range from 1 to 32767.

File ID   These three words are the file ID that the directory entry points to.

If the directory entry is a symbolic link (the flags byte contains DV.SLK), then the value field is variable length. Its first byte is a byte count, and the remainder is an ASCII string which describes the linkage. The string is padded to the next word boundary with a null if necessary. The format is the following:

```
  |------------------|------------------|
  | |     Byte Count |
  |-- |------------------|
  |  |
  |  |
  |  |
  |         Symbolic Link String |
  |  |
  |  |
  |  |
  |---------------------------------------|
```

5.0 Reserved Files

Clearly any file system must maintain some data structure on the medium which is used to control the file organization. In Files-11 this data is kept in several files. These files are created when a new volume is initialized. They are unique in that their File ID's are known constants. Note, however, that the relative volume number used when accessing one of these files depends upon the context. The exact number of these files which is present on a particular volume may vary; however, at least five must be present. All of these files are non-deletable. These files have the following uses:

File ID 1,1 is the index file. The index file is the root of the entire Files-11 structure. It contains the volume's bootstrap block and the home block, which is used to identify the volume and locate the rest of the file structure. The index file also contains all of the file headers for the volume, and a bitmap to control the allocation of file headers.

File ID 2,2 is the storage bitmap file. It is used to control the allocation of logical blocks on the volume.

File ID 3,3 is the bad block file. It is a file containing all of the known bad blocks on the volume.

File ID 4,4 is the volume master file directory (or MFD). It forms the root of the volume's directory structure. The MFD lists the five known files, all first level user directories, and whatever other files the user chooses to enter.

File ID 5,5 is the system core image file. Its use is operating system dependent; its basic purpose is to provide a file of known File ID for the use of the operating system.

File ID 6,6 is the volume's free space file. The blocks contained in this file are available for allocation by an alternate allocation scheme that does not drive off the storage bitmap.

File ID 7,7 is the volume set list file. If this volume is relative volume one of a tightly coupled volume set, this file contains a list of the labels of all the volumes in the set.

File ID 8,8 is the volume backup journal file. It contains a log of full volume and incremental backups performed on the volume.

File ID 9,9 is the standard continuation file. If this volume is part of a loosely coupled volume set, this file contains the first segment of the portion of the multi-volume file that resides on this volume.

More File ID's may be reserved in the future; users should not make any assumptions about the values of user created File ID's.

5.1 Index File

The index file is File ID 1,1. It is listed in the MFD as INDEXF.SYS;1. The index file is the root of the Files-11 structure in that it provides the means for identification and initial access to a Files-11 volume, and contains the access data for all files on the volume (including itself). This file has the FCS record format of 512 byte fixed length records, with no carriage control. (See section 6 for a description of the FCS file format.)

5.1.1 Bootstrap Block -

Virtual block 1 of the index file is the volume's boot block. It is almost always mapped to logical block 0 of the volume. If the volume is the system device of an operating system, the boot block contains an operating system dependent program which reads the operating system into memory when the boot block is read and executed by a machine's hardware bootstrap. If the volume is not a system device, the boot block contains a small program that outputs a message on the system console to inform the operator to that effect. If block 0 of a volume is bad, it is permissible to map virtual block 1 of the index file to some other block. In this case, obviously, volume cannot be used as a system volume.

5.1.2 Home Block -

Virtual block 2 of the index file is the volume's home block. The purpose of the home block is to identify the volume as Files-11, establish the specific identity of the volume, and serve as the ground zero entry point into the volume's file structure. The home block is recognized as a home block by the presence of checksums in known places and by the presence of predictable values in certain locations.

The home block is located on the first good block of the home block search sequence. The search sequence is of the form

1 + n * delta, n = 0, 1, 2, 3, 4 ......

The home block search delta is computed from the geometry of the volume such that, if the volume is viewed as a three dimensional space, the search sequence will travel approximately down the body diagonal of the space. Since volume failures tend to occurr across one dimension, this minimizes the chance of a single failure destroying both home blocks of the volume. The search delta is computed from the volume geometry, expressed in sectors, tracks (surfaces), and cylinders, according to the following rules, to handle the cases where one or two dimensions of the volume have a size of 1.

Geometry:	Delta
s x 1 x 1:	1
1 x t x 1:	1
1 x 1 x c:	1
s x t x 1:	s+1
s x 1 x c:	s+1
1 x t x c:	t+1
s x t x c:	(t+1)*s+1

In most cases, the home block is located on LBN 1.

5.1.3 Cluster Filler -

If v, the cluster factor of the volume, is greater than 1, then the next v*2-2 blocks of the index file are copies of the home block used to fill out the first two clusters of the index file. Note that, for cluster factors greater than 1, this results in a wasted disk cluster. The benefit of this technique is a much simpler rule for finding the VBN of interesting parts of the index file.

5.1.4 Backup Home Block -

The backup home block is a second copy of the home block located farther down the home block search sequence. It permits the volume to be used even if the primary home block is destroyed.

In general, the backup home block should be allocated on the second good block of the search sequence. If it is not, then all preceding blocks on the sequence must not be available for allocation. This is to prevent the situation of a malicious user constructing a counterfeit index file, which would be used if the primary home block ever went bad.

The cluster which contains the backup home block is mapped into the index file as virtual blocks v*2+1 through v*3, where v is the volume cluster factor. Observe that the backup home block may be located anywhere within this cluster, because there is no hard and fast relationship between the cluster factor and the volume's track and cylinder boundaries. The entire cluster is therefore filled out with copies of the home block.

5.1.5 Backup Index File Header -

The next cluster of the index file contains a backup copy of the index file header, so that data on the volume can be recovered if the index file header goes bad. The cluster occupies virtual blocks v*3+1 through v*4, where v is the volume cluster factor. The LBN of the backup index file header is stored in location H.IHLB in the home block. The backup index file header occupies the first block of this cluster; the remaining blocks are not used and their contents are undefined.

5.1.6 Index File Bitmap -

The index file bitmap is used to control the allocation of file numbers (and hence file headers). It is simply a bit string of length n, where n is the maximum number of files permitted on the volume (contained in offset H.FMAX in the home block). The bitmap spans over as many blocks as is necessary to hold it, i.e., max number of files divided by 4096 and rounded up. The number of blocks in the bitmap is contained in offset H.IBSZ of the home block.

The bits in the index file bitmap are numbered sequentially from 0 to n-1 in the obvious manner, i.e., from right to left in each byte, and in order of increasing byte address. Bit j is used to represent file number j+1: if the bit is 1, then that file number is in use; if the bit is 0, then that file number is not in use and may be assigned to a newly created file.

The index file bitmap starts at virtual block v*4+1 of the index file and continues through VBN v*4+m, where m is the number of blocks in the bitmap, and v is the storage map cluster factor. It is located at the logical block indicated by offset H.IBLB in the home block.

5.1.7 File Headers -

The rest of the index file contains all the file headers for the volume. The first 16 file headers (for file numbers 1 to 16) are logically contiguous with the index file bitmap to facilitate their location; the rest may be allocated wherever the file system sees fit. Thus the first 16 file headers may be located from data in the home block (H.IBSZ and H.IBLB) while the rest must be located through the mapping data in the index file header. The file header for file number n is located at virtual block v*4+m+n (where m is the number of blocks in the index file bitmap, and v is the storage map cluster factor).

The FCS end of file mark for the index file is located at the last file header ever used. All header blocks located before the EOF are subject to validation when used to create a new file. If the block contains garbage, the new header is assigned a file sequence number of 1, being the first use of this header block. If the block contains a deleted file header, the new header is assigned a sequence number one higher than the one contained in the block. A block containing a valid file header must never be used to create a new file, even if it is marked free in the index file bitmap. This prevents files from being lost if bits are dropped in the bitmap. Index file blocks beyond the EOF are assumed to contain garbage for the purpose of creating new file headers.

5.1.8 Index File Layout -

The following is a sketch of the blocks in the index file. Observe that this illustration assumes a storage map cluster factor greater than 2.

```
|------------------|---
!                  !   \
!       Boot       !    !
!       Block      !    !
!                  !    !
|------------------|    !
!                  !    !
!       Home       !    !
!       Block      !    ! Cluster 1
!                  !    !
|------------------|    !
!       More       !    !
!       Home       !    !
!      Blocks      !    !
!         :        !    !
!         :        !    !
!         :        !   /
|------------------|---
!                  !   \
!       More       !    !
!       Home       !    !
!      Blocks       !    ! Cluster 2
!        :          !    !
!        :          !    !
!        :          !   /
!-------------------!---
!      Backup       !   \
!       Home        !    !
!       Block       !    !
!                   !    !
!-------------------!    !
!       More        !    ! Cluster 3
!       Home        !    !
!      Blocks       !    !
!        :          !    !
!        :          !    !
!        :          !   /
!-------------------!---
!      Backup       !   \
!       Index       !    !
!       File        !    !
!      Header       !    !
!-------------------!    !
!                   !    ! Cluster 4
!                   !    !
!       (not)       !    !
!      (used)       !    !
!                   !    !
!                   !   /
!-------------------!---
!                   !   \
!       Index       !    !
!       File        !    !
!      Bitmap       !    !
!                   !    !
!                   !    !
!-------------------!    ! Contiguous
!                   !    !
!        16         !    !
!       File        !    !
!      Headers      !    !
!                   !    !
!                   !   /
!-------------------!---
!                   !
!                   !
!                   !
!       Lots        !
!       More        !
!       File        !
!      Headers      !
!                   !
!                   !
!                   !
```

```
|                   |
|-------------------|
```

5.1.9 Home Block Details -

The following is a detailed description of the home block. Note that all copies of the volume's home block contain the same data, with the exception of the cells containing the block's VBN and LBN.

Items contained in the home block are identified by symbolic offsets in the same manner as items in the file header. The symbols may be defined in assembly language programs by calling and invoking the macro HMBL2$, which may be found in the macro library of any system that supports Files-11. Alternatively, one may find the macro in the file F11MAC.MAC, which is available from the author.

5.1.9.1 H.HBLB - 4 Bytes Home Block LBN

This double word contains the logical block number of this particular copy of the home block.

5.1.9.2 H.AHLB - 4 Bytes Alternate Home Block LBN

This double word contains the LBN of the volume's secondary home block. One may determine, when scanning the home block sequence, whether the block read is the primary or secondary home block by comparing H.HBLB and H.AHLB. This value must be non-zero for a valid home block.

5.1.9.3 H.IHLB - 4 Bytes Backup Index File Header LBN

This double word contains the logical block on which the backup index file header is located. This value must be non-zero for a valid home block.

5.1.9.4 H.VLEV - 2 Bytes Structure Level and Version

The volume structure level and version is used to identify different versions of Files-11 as they affect the structure of all parts of the volume except the file header. This permits upwards compatibility of file structures as Files-11 evolves, in that the structure level word identifies the version of Files-11 that created this particular volume. This document describes structure level 2 of Files-11. The high byte of H.VLEV must contain the value 2. The low byte contains the version number, which must be greater or equal to 1. The version number will be incremented whenever compatible additions are made to the Files-11 structure that may be safely ignored by an old version of the file system. This document describes version 1 of structure level 2.

5.1.9.5 H.SBCL - 2 Bytes Storage Bitmap Cluster Factor

This word contains the cluster factor used in the storage bitmap file. The cluster factor is the number of blocks represented by each bit in the storage bitmap. This value is also referred to as the volume cluster factor.

5.1.9.6 H.HBVB - 2 Bytes Home Block VBN

This word contains the virtual block that this particular copy of the home block occupies in the index file. This value must be non-zero for a valid home block.

5.1.9.7 H.AHVB - 2 Bytes Backup Home Block VBN

This word contains the virtual block number that the cluster containing the secondary home block occupies in the index file. The contents of this word is v*2+1, where v is the storage map cluster factor.

5.1.9.8 H.IHVB - 2 Bytes Backup Index File Header VBN

This word contains the virtual block number that the backup index file header occupies in the index file. The contents of this word is v*3+1, where v is the storage map cluster factor.

5.1.9.9 H.IBVB - 2 Bytes Index File Bitmap VBN

This word contains the starting virtual block number of the index file bitmap. The contents of this word is the value v*4+1, where v is the storage map cluster factor.

5.1.9.10 H.IBLB - 4 Bytes Index File Bitmap LBN

This double word contains the starting logical block address of the index file bitmap. Once the home block of a volume has been found, it is this value that provides access to the rest of the index file and to the volume. This value must be non-zero for a valid home block.

5.1.9.11 H.FMAX - 4 Bytes Maximum Number of Files

This double word contains the maximum number of files that may be present on the volume at any time. This value must be greater than the contents of H.RSVF for the home block to be valid. If the maximum number of files is less than 65536, then the third word of File ID's referencing files on this volume is simply the relative volume number, and the volume set of which this volume is a member may contain up to 65535 volumes. If the maximum number of files is greater than or equal to 65536, however, then the high byte of the third word of File ID's is the high byte of the file number, and the volume set may consist of up to 255 volumes. Under no circumstances may the maximum number of files be greater than 2**24-1.

5.1.9.12 H.IBSZ - 2 Bytes Index File Bitmap Size

This 16 bit word contains the number of blocks that make up the index file bitmap. This value must be non-zero for a valid home block.

5.1.9.13 H.RSVF - 2 Bytes Number of Reserved Files

This word contains the number of of reserved file on the volume. The file sequence number of each reserved file is always equal to its file number. Reserved files may not be deleted. This word must contain a minimum value of 5 to be valid.

5.1.9.14 H.DVTY - 2 Bytes Disk Device Type

This word is an index identifying the type of disk that contains this volume. It is currently not used and always contains 0.

5.1.9.15 H.RVN - 2 Bytes Relative Volume Number

This word contains the relative volume number that this volume has been assigned in a volume set. If the volume is not part of a volume set, then this word contains zero.

5.1.9.16 H.NVOL - 2 Bytes Number of Volumes

This word contains the total number of volumes in this volume set if the contents of H.RVN is 1 (i.e., if this volume is the first volume of the volume set). Otherwise, this word contains zero.

5.1.9.17 H.VCHA - 2 Bytes Volume Characteristics

This word contains bits which provide additional control over access to the volume. The following bits are defined:

CH.NDC Set if device control functions are not permitted on this volume. Device control functions are those which can threaten the integrity of the volume, such as direct reading and writing of logical blocks, etc.

CH.NAT	Set if the volume may not be attached, i.e., reserved for the sole use by one task or user.
CH.RCK	Set if the volume is to be read checked. All block reads done on this volume, both for data and for file structure, will be performed with a read, read-compare sequence to insure data integrity.
CH.WCK	Set if the volume is to be write checked. All block writes done on this volume, both for data and for file structure, will be performed with a write, read-compare sequence to insure data integrity.

5.1.9.18 H.VOWN - 4 Bytes Volume Owner UIC

This double word contains the binary UIC of the owner of the volume. The format is the same as that of the file owner UIC stored in the file header.

5.1.9.19 H.VSMX - 4 Bytes Volume Security Mask

These four bytes contain the security mask for the volume. In the same manner as the security mask of a file, the volume security mask controls the information categories that may be stored on the volume. Only files whose security mask is a subset of the volume security mask may be written on the volume. Note, however, that the security mask of a user accessing files on the volume does not have to be a superset of the volume mask, since he must still pass the security mask check on the individual files. Further, if such a check were made, the security masks of all files written on the volume would have to be equal to the volume mask, which is not very useful.

5.1.9.20 H.VPRO - 2 Bytes Volume Protection Code

This word contains the protection code for the entire volume. All operations on all files on the volume must pass both the volume and the file protection check to be permitted. Accessors to the volume are categorized into system, owner, group, and world with respect to the volume owner UIC in the same manner as for file protection. Each category is controlled by the familiar four bit field. The four access modes are bit encoded as follows:

VP.RDV	Deny reading files
VP.WRV	Deny writing existing files
VP.CRE	Deny creating files
VP.DEL	Deny deleting files

5.1.9.21 H.DFPR - 2 Bytes Default File Protection

This word contains the file protection that will be assigned to all files created on this volume if no file protection is specified by the user.

5.1.9.22 H.DRPR - 2 Bytes Default Record Protection

This word contains the record protection that will be assigned to all files created on this volume if no file protection is specified by the user.

5.1.9.23 H.CHK1 - 2 Bytes First Checksum

This word is an additive checksum of all entries preceding in the home block (i.e., all those listed above). It is computed by the same sort of algorithm as the file header checksum (see section 3.5.7.1).

5.1.9.24 H.VDAT - 8 Bytes Volume Creation Date

This area contains the date and time that the volume was initialized. It is in the same binary format used in the file header (see section 3.5.3.3 3.4.2).

5.1.9.25 H.WISZ - 1 Byte Default Window Size

This byte contains the number of retrieval pointers that will be used for the "window" (in core file access data) when files are accessed on the volume, if not otherwise specified by the accessor.

5.1.9.26 H.LRUC - 1 Byte Directory Pre-access Limit

This byte contains a count of the number of directories to be stored in the file system's directory access cache. More generally, it is an estimate of the number of concurrent users of the volume and its use may be generalized in the future.

5.1.9.27 H.FIEX - 2 Bytes Default File Extend

This word contains the number of blocks that will be allocated to a file when a user extends the file and asks for the system default value for allocation.

384

5.1.9.28 - - 388 Bytes Not Used

H.SERN - 4bytes Volume Serial [illegible]

5.1.9.29 H.SNAM - 12 Bytes Structure Name

This area contains the ASCII name of the volume set to which this volume belongs, padded out to 12 bytes with spaces. If this volume is not a member of a volume set, then this area is filled with nulls.

5.1.9.30 H.INDN - 12 Bytes Volume Name

This area contains the volume label in ASCII. It is padded out to 12 bytes with spaces. It is placed here in accordance with the proposed volume identification standard.

5.1.9.31 H.INDO - 12 Bytes Volume Owner

This area contains an ASCII string identifying the owner of the volume. The area is padded out to 12 bytes with trailing spaces. It is placed here in accordance with the proposed volume identification standard.

5.1.9.32 H.INDF - 12 Bytes Format Type

This field contains the ASCII string "DECFILE11B" padded out to 12 bytes with spaces. It identifies the volume as being of Files-11 format, structure level 2. It is placed here in accordance with the proposed volume identification standard.

5.1.9.33 - - 2 Bytes Not Used

5.1.9.34 H.CHK2 - 2 Bytes Second Checksum

This word is the last word of the home block. It contains an additive checksum of the preceding 255 words of the home block, computed according to the algorithm listed in section 3.5.7.1.

5.1.9.35 Home Block Layout -

```
         |---------------------------------------|
         |                                       |     H.HBLB
         |--          LBN of This Block        --|
         |                                       |
         |---------------------------------------|
         |                                       |     H.AHLB
         |--     LBN of Secondary Home Block   --|
         |                                       |
         |---------------------------------------|
         |            LBN of Secondary           |     H.IHLB
         |--                                   --|
         |           Index File Header           |
         |---------------------------------------|
         |        Volume Structure Level         |     H.VLEV
         |---------------------------------------|
         |    Storage Bitmap Cluster Factor      |     H.SBCL
         |---------------------------------------|
         |           VBN of This Block           |     H.HBVB
         |---------------------------------------|
         |        Backup Home Block VBN          |     H.AHVB
         |---------------------------------------|
         |       Backup Index Header VBN         |     H.IHVB
         |---------------------------------------|
         |         Index File Bitmap VBN         |     H.IBVB
         |---------------------------------------|
         |              Index File               |     H.IBLB
         |--                                   --|
         |              Bitmap LBN               |
         |---------------------------------------|
         |                                       |     H.FMAX
         |--       Maximum Number of Files     --|
         |                                       |
         |---------------------------------------|
         |        Index File Bitmap Size         |     H.IBSZ
         |---------------------------------------|
         |       Number of Reserved Files        |     H.RSVF
         |---------------------------------------|
         |           Disk Device Type            |     H.DVTY
         |---------------------------------------|
         |        Relative Volume Number         |     H.RVN
         |---------------------------------------|
         |       Number of Volumes in Set        |     H.NVOL
         |---------------------------------------|
         |        Volume Characteristics         |     H.VCHA
         |---------------------------------------|
         |                                       |     H.VOWN
         |--          Volume Owner UIC         --|
         |                                       |
         |---------------------------------------|
         |                                       |     H.VSMX
```

```
                 |--         Volume Security Limit           --|
                 |                                             |
                 |-------------------|-------------------------|
                 |           Volume Protection                 |     H.VPRO
                 |---------------------------------------------|
                 |        Default File Protection              |     H.DFPR
                 |---------------------------------------------|
                 |       Default Record Protection             |     H.DRPR
                 |---------------------------------------------|
                 |            First Checksum                   |     H.CHK1
                 |---------------------------------------------|
                 |                                             |     H.VDAT
                 |--                                         --|
                 |                                             |
                 |--         Volume Creation Date            --|
                 |                                             |
                 |--                                         --|
                 |                                             |
                 |---------------------------------------------|
H.LRUC           | Directory Limit  | Def. Window Size         |     H.WISZ
                 |------------------|--------------------------|
                 |          Default File Extend                |     H.FIEX
                 |---------------------------------------------|
                 |                                             |
                 |                                             |
                 |                                             |
                 |                                             |
                 |               (not used)                    |
                 |                                             |
                 |                                             |
                 |                                             |
                 |                                             |
                 |---------------------------------------------|
                 |                                             |     H.SNAM
                 |--                                         --|
                 |                                             |
                 |--                                         --|
                 |                                             |
                 |--           Structure Name                --|
                 |                                             |
                 |--                                         --|
                 |                                             |
                 |--                                         --|
                 |                                             |
                 |---------------------------------------------|
                 |                                             |     H.INDN
                 |--                                         --|
                 |                                             |
                 |--                                         --|
                 |                                             |
                 |--             Volume Name                 --|
                 |                                             |
                 |--                                         --|
        |                                           |
        |--                                       --|
        |                                           |
        |-------------------------------------------|
        |                                           |     H.INDO
        |--                                       --|
        |                                           |
        |--                                       --|
        |                                           |
        |--             Volume Owner              --|
        |                                           |
        |--                                       --|
        |                                           |
        |--                                       --|
        |                                           |
        |-------------------------------------------|
        |                                           |     H.INDF
        |--                                       --|
        |                                           |
        |--                                       --|
        |                                           |
        |--              Format Type              --|
        |                                           |
        |--                                       --|
        |                                           |
        |--                                       --|
        |                                           |
        |-------------------------------------------|
        |                (not used)                 |
        |-------------------------------------------|
        |             Second Checksum               |     H.CHK2
        |-------------------------------------------|
```

5.2 Storage Bitmap File

The storage bitmap file is File ID 2,2. It is listed in the MFD as BITMAP.SYS;1. The storage bitmap is used to control the available space on a volume. It consists of a storage control block which contains summary information about the volume, and the bitmap itself which lists the availablilty of individual blocks. This file has the FCS record format of 512 byte fixed length records, with no carriage control. The end of file mark is positioned to point to the last block used. The storage bitmap file must be contiguous.

5.2.1 Storage Control Block -

Virtual block 1 of the storage bitmap is the storage control block. It contains summary information about the volume. Note that implementation of some of the features in the storage control block may require it to be written at mount and dismount.

5.2.1.1 C.VLEV - 2 Bytes Storage Map Structure Level

This word contains the structure level of the storage control block. The high byte contains the value 2 to indicate Files-11 structure level 2. The low byte contains the version number, which must be equal to or greater then 1.

5.2.1.2 C.SBCL - 2 Bytes Storage Map Cluster Factor

This word contains the storage map cluster factor of the volume. Its contents are identical to the contents of H.SBCL in the home block. It is placed here for convenience.

5.2.1.3 C.VSIZ - 4 Bytes Volume Size

These four bytes contain the volume size expressed in logical blocks.

5.2.1.4 C.BLKF - 4 Bytes Blocking Factor

These words contain the blocking factor of the volume; i.e., the number of physical blocks or sectors that make up one logical block.

5.2.1.5 C.SECT - 4 Bytes Sectors Per Track

These words contain the number of logical blocks in each track of the volume.

5.2.1.6 C.TRAK - 4 Bytes Tracks Per Cylinder

These words contain the number of tracks contained in each cylinder of the volume.

5.2.1.7 C.CYLN - 4 Bytes Number of Cylinders

These words contain the total number of cylinders on the volume. The above three quantities are present to assist optimized allocation of space on physical boundaries in the volume.

5.2.1.8 C.STAT - 2 Bytes Status Word

This word contains the following status bits:

CS.TRN	Volume in transition. This bit is set if the volume may be in an inconsistent state because it was not dismounted properly. A system which does write on replace caching of the storage map, for example, should set this bit on mount and clear it on dismount.

5.2.1.9 - - 488 Bytes (not used)

5.2.1.10 C.CKSM - 2 Bytes Block Checksum

This word contains the ubiquitous block checksum. It is computed using the same algorithm as the file header checksum (section 3.5.7.1).

5.2.1.11 Storage Control Block Layout -

```
|---------------------------------------|
|            Structure Level            |   C.VLEV
|---------------------------------------|
|     Storage Map Cluster Factor        |   C.SBCL
|---------------------------------------|
|                                       |   C.VSIZ
|--        Volume Size in Blocks      --|
        |                                         |
        |-----------------------------------------|
        |                                         |     C.BLKF
        |--          Blocking Factor            --|
        |                                         |
        |-----------------------------------------|
        |                                         |     C.SECT
        |--          Sectors Per Track          --|
        |                                         |
        |-----------------------------------------|
        |                                         |     C.TRAK
        |--         Tracks Per Cylinder         --|
        |                                         |
        |-----------------------------------------|
        |                                         |     C.CYLN
        |--         Cylinders on Volume         --|
        |                                         |
        |-----------------------------------------|
        |             Volume Status               |     C.STAT
        |-----------------------------------------|
        |                                         |
        |                                         |
        |                                         |
        |                                         |
        |                                         |
        |               (not used)                |
        |                                         |
        |                                         |
        |                                         |
        |                                         |
        |                                         |
        |-----------------------------------------|
        |             Block Checksum              |     C.CKSM
        |-----------------------------------------|
```

5.2.2 Storage Bitmap -

Virtual blocks 2 through n+1 are the storage bitmap itself. It is best viewed as a bit string of length m, numbered from 0 to m-1, where m is the total number of allocatable clusters on the volume rounded up to the next multiple of 4096. Each cluster contains v logical blocks, where v is the storage map cluster factor (also referred to as the volume cluster factor) contained in location H.SBCL in the home block. The bits are addressed in the usual manner (packed right to left in sequentially numbered bytes). Since each virtual block holds 4096 bits, n blocks, where n = m/4096, are used to hold the bitmap. Bit j of the bitmap represents logical blocks j*v through j*v+v-1 of the volume; if the bit is set, the blocks are free; if clear, the blocks are allocated. Clearly the last k bits of the bitmap are always clear, where k is the difference between the true size of the volume and m, the length of the bitmap.

Rounding the storage map file up to the next multiple of the volume cluster factor may result in some unused blocks at the end of the file. The FCS end of file mark points to the last block used.

5.3 Bad Block File

The bad block file is File ID 3,3. It is listed in the MFD as BADBLK.SYS;1. The bad block file is simply a file containing all of the known bad blocks on the volume. This file has the FCS record format of 512 byte fixed length records, with no carriage control. The end of file mark may be placed as the operating system's bad block handling strategy finds useful. Volume initialization should place the EOF at the end of the bad blocks found during initialization. At all times, the EOF should at least point past the bad block descriptor data, described below. This ensures that the bad block data is preserved for future re-initialization of the volume.

5.3.1 Factory Bad Block Descriptor -

On disks which have factory generated last track bad block data, such as the RK06, RK07, and RM03, the first several clusters of the bad block file should include the last track of the volume. This track contains redundantly recorded descriptions of the bad blocks on the volume, as described in DEC STD. 144, "Disk Standard for Recording and Handling Manufacturing Detected Bad Sectors".

5.3.2 Software Bad Block Descriptor -

On disks that do not have factory last track bad block data, the first cluster of the bad block file contains the bad block descriptor for the volume. It is always located on the last good block of the volume. This block may contain a listing of the bad blocks on the volume produced by a bad block scan program or diagnostic. The software bad block descriptor is most of a Files-11 Structure Level 1 header map area. The first two bytes contain the constants 1 and 3, respectively. The third byte contains the number of words that contain data. The fourth byte contains the number of words available for bad block data. The last word of the block contains the usual additive checksum. The retrieval pointers are structure level 1 format 1 pointers, as described below.

Bad Block Descriptor Layout

```
|------------------|------------------|
|        3         |        1         |
|------------------|------------------|
| Map Words Avail. | Map Words in Use |
|------------------|------------------|
|                                     |
|                                     |
|                                     |
|          Retrieval Pointers         |
|                                     |
|                                     |
|                                     |
|-------------------------------------|
|            Block Checksum           |
|-------------------------------------|
```

Each retrieval pointer is four bytes in length. Byte 1 contains the high order bits of the 24 bit LBN. Byte 2 contains the count field, and bytes 3 and 4 contain the low 16 bits of the LBN.

```
|----------|----------|
|  Count   |   High   |
|----------|-        -|
|    Low Order LBN    |
|---------------------|
```

5.4 Master File Directory

The master file directory is File ID 4,4. It is listed in the MFD (itself) as 000000.DIR;1. The MFD is the root of the volume's directory structure. It lists the reserved files, plus whatever the user chooses to enter. The format of the MFD is the same as all directory files, and is described in section 4.3. In the UFD structures described in sections 4.1.1 and 4.1.2, the MFD contains entries for all user file directories.

5.5 Core Image File

The core image file is File ID 5,5. It is listed in the MFD as CORIMG.SYS;1. Its use is operating system dependent. In general, it provides a file of known File ID for the use of the operating system, for use as a swap area, for example, or as a monitor overlay area, etc. This file has the FCS record format of 512 byte fixed length records, with no carriage control. The end of file mark is positioned to point to the physical end of file.

5.6 Free Space File

The free space file is File ID 6,6. It is listed in the MFD as FREFIL.SYS;1. The space it contains is available for allocation to other files. The presence of this file allows individual implementations of Files-11 to use an alternate scheme of space allocation which is more complex than using the storage bitmap alone, but has potentially much better performance. Systems which do not support this method of allocation should truncate this file to zero and return the space it maps to the storage bitmap before using the volume. This file has the FCS record format of 512 byte fixed length records, with no carriage control. Its end of file mark is undefined.

5.7 Volume Set List

The volume set list is File ID 7,7. It is listed in the MFD as VOLSET.SYS;1. It is used only on relative volume one of a tightly coupled volume set. There, it contains a list of the volume labels of the volumes contained in this volume set. The format of this file is FCS 64 byte fixed length records with implied carriage control. The first 12 bytes of record 1 contain the structure name of the volume set. The first 12 bytes of record n contain the volume label of relative volume n-1. The remaining 52 bytes of each record are reserved for future use.

5.8 Backup Log File

The backup log file is File Id 8,8,0. It is listed in the MFD as BACKUP.SYS;1. This file contains a history of volume and incremental backups performed on the volume. Its format is at present undefined.

5.9 Continuation File

The standard continuation file is File ID 9,9. It is listed in the MFD as EXTFIL.SYS;1. It is used as the extension File ID when a file crosses from one volume of a loosely coupled volume set to another. The purpose of this reserved File ID is allow a multi-volume file to be written sequentially with only one volume mounted at a time. Ordinarily, when a file is extended onto another volume, the new header must be created first to obtain the new File ID before the extension linkage in the current header can be written. The use of this reserved File ID allows the extension linkage to be written with a known constant before the next volume is even on line.

6.0 FCS File Structure

File Control Services (FCS) is a user level interface to Files-11 implemented in the RSX-11 systems. Its principal feature is a record control facility that allows sequential processing of variable length records and sequential and random access to fixed length record files. FCS interfaces to the virtual block facility provided by the basic Files-11 structure.

6.1 FCS File Attributes

FCS stores attribute information about the file in the file's user attribute area (H.UFAT - see section 3.5.2.13). It uses only the first 7 words; the rest are ignored by FCS. The following items are contained in the attribute area; they are identified by the usual symbolic offsets (relative to the start of the attribute area). The offsets may be defined in assembly language programs by calling and invoking the macro FDOFF$ DEF$L. Flag values and bits may be defined by calling and invoking the macro FCSBT$. These macros are in the system macro library of any operating system that supports Files-11. Alternatively, all these values are defined in the system object library of any system that supports Files-11, and may be obtained at link time.

6.1.1 F.RTYP 1 Byte Record Type -

This byte identifies which type of records are contained in this file. The following three values are legal;

R.FIX	Fixed length records.
R.VAR	Variable length records.
R.SEQ	Sequenced variable length records

6.1.2 F.RATT 1 Byte Record Attributes -

This byte contains record attribute bits that control the handling of records under various contexts. The following flag bits are defined:

FD.FTN	Use Fortran carriage control if set. The first byte of each record is to be interpreted as a standard Fortran carriage control character when the record is copied to a carriage control device.
FD.CR	Use implied carriage control if set. When the file is copied to a carriage control device, each record is to be preceded by a line feed and followed by a carriage return. Note that the FD.FTN and FD.CR bits are mutually exclusive.
FD.BLK	Records do not cross block boundaries if set. Generally, there will be dead space at the end of each block; how this is handled is explained in the description of record formats in section 6.2.
FD.PRN	Use print file carriage control. Legal only if the record type is R.SEQ. The leading two byte field of each record is used as carriage control instead of as a sequence number. The first and second bytes are used as leading and trailing carriage formatting, respectively. The interpretation of the carriage control bytes is described below in section 6.2.3.

6.1.3 F.RSIZ 2 Bytes Record Size -

In a fixed length record file, this word contains the size of the records in bytes. In a variable length record file, this word contains the size in bytes of the longest record in the file.

6.1.4 F.HIBK 4 Bytes Highest VBN Allocated -

This 32 bit number is a count of the number of virtual blocks allocated to the file. Since this value is maintained by FCS, it is usually correct, but it is not guaranteed since FCS is a user level package.

6.1.5 F.EFBK 4 Bytes End of File Block -

This 32 bit number is the VBN in which the end of file is located. Both F.HIBK and F.EFBK are stored with the high order half in the first two bytes, followed by the low order half.

6.1.6 F.FFBY 2 Bytes First Free Byte -

This word is a count of the number of bytes in use in the virtual block containing the end of file; i.e., it is the offset to the first byte of the file available for appending. Note that an end of file that falls on a block boundary may be represented in either of two ways. If the file contains precisely n blocks, F.EFBK may contain n and F.FFBY will contain 512, or F.EFBK may contain n+1 and F.FFBY will contain 0.

6.1.7 S.FATT 14 Bytes Size of Attribute Block -

This symbol represents the total number of bytes in the FCS file attribute block.

6.2 FCS File Attributes Layout

```
          |------------------|------------------|
F.RATT    |   Record Attr.   |   Record Type    |    F.RTYP
          |------------------|------------------|
          |         Record Size (Bytes)         |    F.RSIZ
          |-------------------------------------|
          |            Highest VBN              |    F.HIBK
          |--                                 --|
          |             Allocated               |
          |-------------------------------------|
          |            End of File              |    F.EFBK
          |--                                 --|
         |                   VBN                  |
         |----------------------------------------|
         |            First Free Byte             |   F.FFBY
         |----------------------------------------|   S.FATT
```

6.3 Attribute Standardization

To assure a certain consistency of file record structures, certain fields in the record attributes area are standardized, and must contain well defined values regardless of the record structure or file organization in use.

1. The record type byte (F.RTYPE) must contain a code that identifies the file organization and record structure. All codes must be registered with this specification.

2. The record attribute bits should be used as described above when applicable. New attributes should be registered with this specification.

3. The high VBN field (F.HIBK) must contain the number of blocks allocated to the file. File managers may modify this field during some operations on the file.

4. The end of file mark (F.EFBK and F.FFBY) should describe the end of data in the file when applicable.

6.4 Record Structure

This section describes how records are packed in the virtual blocks of a disk file. In general, FCS treats a disk file as a sequentially numbered array of bytes. Records are numbered consecutively starting with 1.

6.4.1 Fixed Length Records -

In a file consisting of fixed length records, the records are simply packed end to end with no additional control information. If the record length is odd, each record is padded with a single null. For direct access, the address of a record is computed as follows:

Let: n = record number

```
            k = record size (in bytes)
            m = byte address of record in file
            q = number of records per block
            j = VBN containing the start of the record
            i = byte offset within VBN j

Then        h = ((k+1)/2)*2 (rounded up record length)
            m = (n-1)*h
            j = m/512+1 (truncated)
            i = m mod 512
```

The previous discussion assumes that records cross block boundaries (that is, FD.BLK is not set). If records do not cross block boundaries, they are limited to 512 bytes, and the following equations apply (the variables are defined as above):

```
            h = ((k+1)/2)*2 (rounded up record length)
            q = 512/h (truncated)
            j = (n-1)/q+1 (truncated)
            i = ((n-1) mod q)*h
```

6.4.2 Variable Length Records -

In a file consisting of variable length records, records may be up to 32767 bytes in length. Each record is preceded by a two byte binary count of the bytes in the record (the count does not include itself). For example, a null record is represented by a single zero word. The byte count is always word aligned; i.e., if a record ends on an odd byte boundary, it is padded with a single null.

If records do not cross block boundaries (FD.BLK is set), they are limited to a size of 510 bytes. A byte count of -1 is used as a flag to signal that there are no more records in a particular block. The remainder of that block is then dead space and the next record in the file starts at the beginning of the next block.

6.4.3 Sequenced Variable Length Records -

The format of a sequenced file is identical to a variable length record file except that a two byte sequence number field is located immediately after the byte count field of each record. This field contains a binary value which is usually interpreted as the line number of that record (see Section 6.1.2 FD.PRN and Section 6.2.3.1). The sequence number is not returned as part of the data when a record is read, but is available separately. Note that the record byte count field counts the sequence number field as well as the data of the record.

6.4.3.1 Format of Two Byte Print Control Field in R.SEQ Records -

If the FD.PRN bit is set in the record attribute then the two byte "sequence number" field is used to contain carriage control data for the record. Byte 0 is print control information to act upon before the record data is output to a unit record device; byte 1 is print control information to act upon after the record data has been output to a unit record device.

The format of each byte is as follows:

Bit 7	Bits 6-0	Meaning
0	0	No carriage control
0	count(1-127)	"count" new lines (CR/LF)

Bit 7	Bit 6	Bit 5	Bits 4-0	Meaning
1	0	0	ASCII C0 set	ASCII char to output (CR,FF etc.)
1	0	1	ASCII C1 set	ASCII char (8 bit code) to output
1	1	0	code (0-63)	Device specific code
1	1	1	-	Reserved

NOTE

The print control field is not currently supported by FCS or RMS-11.

7.0 Record Management Services (RMS)

Record Management Services (RMS) is a user level interface to Files-11. It provides a flexible means of data storage, retrieval, and modification through a combination of file organization and record access modes. File organization is the structure of data within the virtual blocks of a Files-11 file, and record access mode is the manner in which storing and retrieving the data in the file occurs.

RMS supports/defines three file organizations which are:

- Sequential - compatible with FCS fixed, variable, and sequenced variable record files (see Section 6)
- Relative - RMS only
- Indexed - RMS only

RMS interfaces to the virtual block facility provided by the Files-11 structure.

7.1 Data Formats and Representation

RMS supports file organizations which require a more complex degree of structuring than that required by FCS. RMS also stores binary values in a different manner in general than Files-11 defines. For these reasons the data format and representations used by RMS are given in the following sections.

7.1.1 String Storage -

All strings are stored left justified. The left most character is in byte N and the right most character is in byte N+M-1 where M is the length of the string.

7.1.2 String Character Code Set -

All string values are assumed to be in the 7-bit ASCII code set.

7.1.3 String Collating Sequence -

The collating sequence used is the 7-bit ASCII code set where NUL is the lowest valued character and DEL is the highest valued character.

NOTE

The internal representation of ASCII characters on PDP-11 systems is 7-bit ASCII. The string compare routine of RMS-11 however, performs a full 8-bit unsigned compare per character. RMS does not perform any "clear bit 7" code on input or output operations. This allows the support of user binary byte strings, the KANA character set used in Japan, and in the future 8-bit ASCII when defined, without RMS modifications since the true colating sequence is lowest character = Ø and highest character = 255.

7.1.4 Unsigned Binary Value Storage -

All unsigned binary values are stored with the Least Significant Bits (LSB) in byte N and the Most Significant Bits (MSB) in byte N+M-1 where M is the length of the binary value.

EXAMPLE: 2 byte unsigned binary value

```
---------
!  LSB  ! N
!-------!
!  MSB  ! N+1
---------
```

7.1.5 Signed Binary Value Storage -

All signed binary values are stored as unsigned binary values except that most significant bit (bit 7 of byte N+M-1) of the value is interpreted as the sign of the value. Negative numbers are stored as the two's complement of the positive value.

EXAMPLE: 2 byte signed binary value

```
-------------
|    LSB     | N
|------------|
|S|  MSB     | N+1
-------------
```

7.1.6 Pointer Values -

All pointers are stored as unsigned binary values. Pointers are stored variable length. The length of a pointer value is specified by the control bits associated with the pointer. The length requirement for a pointer is determined by the range of VBN values it falls in as follows:

2 bytes start VBN 1 - 65,535

3 bytes start VBN 65,536 - 16,777,215

4 bytes start VBN 16,777,216 - 4,294,967,295

7.1.7 Bucket Pointers -

A bucket pointer is a pointer value which specifies the start VBN of the bucket. The length of the bucket (number of VBN's in bucket) is interpreted in the context of its usage within the file, and is specified in the file's prolog data.

EXAMPLE: 2 byte bucket pointer

```
---------
|  LSB  | N
|-------|
|  MSB  | N+1
---------
```

7.1.8 Record Pointers -

Record pointers are composed of two fields, a one byte record ID field followed by a bucket pointer. The ID is used as a unique record identifier for records within a bucket. The records are tagged with their ID'S when stored in the bucket.

EXAMPLE: 3 byte record pointer

```
                    ---------
                    |   ID   | N RECORD ID
                    |--------|
                    |  LSB   | N+1  BUCKET POINTER
                    |--------|
                    |  MSB   | N+2
                    ---------
```

7.1.9 Packed Decimal Strings -

Packed decimal strings are from 1 to 16 bytes in length. The format is as follows:

```
         7      4 3       0
         -----------------
         |  d1   |  d2   | A
         -----------------
         |  d3   |  d4   | A+1
         -----------------
                 .
                 .
                 .
         -----------------
         |  di   | sign  | A+N-1
         -----------------
```

where:

d = digit in the range of 0 thru 9 (binary value)

sign is plus if value is 10, 12, 14, or 15

sign is minus if value is 11 or 13

N is length of strings in bytes

i = (N-1)*2+1 and is an odd number in the range of 1 thru 31

d1 is most significant digit (may be a leading zero)

di is least significant digit

7.2 RMS File Attributes

RMS stores attribute information about the file in the file's user attribute area (H.UFAT - see Section 3.4.1.9). It uses the first ten (10) words; the rest are reserved by RMS. The following items are contained in the attribute area; they are identified by symbolic offsets into an RMS internal structure. The relative offset into the attribute area may be calculated by subtracting F$FORG from the given offset name/value. The offset definitions may be defined in assembly language programs by calling and invoking the macro IFAOF$ RMS$L. Flag values and bits may be defined by calling and invoking the FAB$BT DFIN$L macro. These macros can be found in the RMSMAC.MLB macro library on all PDP-11 systems supporting RMS.

7.2.1 F$FORG 1 Byte - Record Format and File Organization

This byte identifies the file's organization and which type of record format it contains. The record format is contained in bits 0 - 3, and the file's organization is contained in bits 4 - 7. The symbolic values are defined such that they may be OR'ED to yield the contents of the F$FORG field.

Record Formats:

FB$UDF	Undefined record format (Block I/O only file)
FB$FIX	Fixed length records
FB$VAR	Variable length records
FB$VFC	Variable with Fixed Control (VFC) records (the FCS R.SEQ is a special case form of the record format i.e., the fixed control area is two bytes long and contains the records sequence number)
FB$STM	ASCII stream records. RMS-11 used only as a means for RSTS/E ASCII data interchange. Records are delimited by vertical form effecter characters (LF, VT, FF and CR/LF pairs).

File Organizations:

FB$SEQ	Sequential File organization (FB$SEQ = 0 to maintain compatibility with FCS)

FB$REL	Relative File organization
FB$IDX	Index File organization
FB$HSH	Hashed File Organization (not implemented)

7.2.2 F$RATT 1 Byte - Record Attributes

This byte contains record attributes bits that control the handling of records under various contexts. The following flag bits are defined:

FB$FTN	See Section 6.1.2 FD.FTN
FB$CR	See Section 6.1.2 FD.CR
FB$PRN	See Section 6.1.2 FD.PRN and 6.2.3.1
FB$BLK	Record do not cross block boundaries for the Sequential file organization if set. See Section 6.1.2 FD.s1LK for more detail.

7.2.3 F$RSIZ 2 Bytes - Record Size

In file containing fixed length format records this word contains the size of the records in bytes. In Sequential files containing variable or variable with fixed control formatted records this field contains the size in bytes of the longest record in the file. This field is undefined for Relative and Indexed files containing variable or variable with fixed control format records.

7.2.4 F$HVBN 4 Bytes - Highest VBN Allocated

RMS updates this field whenever the file is opened for write access. For details on this field see Section 6.1.4 F.HIBK.

7.2.5 F$HEOF 4 Bytes - End of File Block

This 32 bit number is the VBN in which the end of file is located for the Sequential file organization. Both F$HVBN and F$HEOF are stored with the high order half in the first two bytes, followed by the low order half. The low order half is symbolically referenced by F$LVBN and F$LEOF respectively. These are the only two places that RMS stores block numbers in this manner (see Section 7.1), and is done so to maintain compatibility with FCS. The Relative and Index file does not use this field and its value is usually (but not guaranteed) either the contents of F$HVBN or the contents of F$HVBN plus one.

7.2.6 F$FFBY 2 Bytes - First Free Byte

This field is used for the Sequential file organization as a count of the number of bytes in use in the virtual block containing the end of file. The Relative and Indexed file organization do not use this field and its value will be either Ø or 512. For more details on this field see Section 6.1.6 F.FFBY.

7.2.7 F$BKSZ 1 Byte - Bucket Size

This field contains the bucket size or maximum bucket size for the Relative and Indexed file organization respectively. The bucket size is represented as the number of virtual blocks it contains. Legal values are from 1 - 32. For compatibility with FCS a value of Ø is interpreted as 1.

7.2.8 F$HDSZ 1 Byte - Fixed Header Size

This field contains the number of bytes (1 - 255) in the fixed control area when the file contains Variable with Fixed Control format records. A value of Ø is interpreted as 2 so that compatibility with FCS'S Sequenced Variable length record format file (R.SEQ) is maintained.

7.2.9 F$MRS 2 Bytes - Maximum Record Size

This field contains a user specified maximum record size limit in bytes, to be enforced on output operations. Files containing Fixed length format records have F$MRS set equal to F$RSIZ. For all other record formats F$MRS is set to the user specified value given when the file was created. A value of 0 is interpreted as no maximum record size limit specified.

7.2.10 F$DEQ 2 Bytes - default Extend Quantity

This field contains a user specified default file extend quantity to be used whenever RMS needs to extend the file. A value of 0 is interpreted as use the volumes default extend.

7.2.11 RMS File Attributes Layout -

```
            |----------------------------------------|
            |    Record Attr.    | File Org./rec fmt |   F$FORG
            |----------------------------------------|
            |          Record Size (bytes)           |   F$RSIZ
            |----------------------------------------|
            |              Highest VBN               |   F$HVBN
            |--                                    --|
            |               Allocated                |
            |----------------------------------------|
            |              End of file               |   F$HEOF
            |--                                    --|
            |                  VBN                   |
            |----------------------------------------|
            |            First Free Byte             |   F$FFBY
            |----------------------------------------|
F$HDSZ      |  Fixed Ctr. Size   |    Bucket Size    |   F$BKSZ
            |----------------------------------------|
            |       Maximum Record Size Limit        |   F$MRS
            |----------------------------------------|
            |        Default Extend Quantity         |   F$DEQ
            |----------------------------------------|
```

To calculate the offset into the User Attributes area in the file header subtract F$FORG from all symbolic offsets.

7.3 Prologue Blocks

The RMS Relative and Indexed file organizations use the first several virtual blocks of the file to contain additional file description data. This area of the file is called the file prologue. In the Relative file organization, the prologue is exactly one block long; in the Indexed organization its length varies. The symbolic offset names, and flag values and bits used in the file prologue blocks and record formats may be obtained by calling and invoking the following macros from the RMSMAC.MLB macro library on all PDP-11 systems supporting RMS.

ARDOF$	RMS$L
BKTOF$	RMS$L
KDXOF$	RMS$L
KDX$BT	DFIN$L
XAB$BT	DFIN$L
BKT$BT	DFIN$L

The last word of every prologue block contains the standard Files-11 check sum (see Section 3.4.4.1).

7.3.1 Prologue Block 1 (VBN 1) -

Prologue Block 1 contains common data for both the Indexed and Relative files, and file organization dependent data. The major Indexed file dependent data is the primary key definition (the K$XXXX symbols). The major Relative file dependent data are the maximum record number, the address of the first data bucket, and the "real" End of File Block (last initialized, zeroed, VBN). The primary key definition offsets (K$XXXX) are used for all key definitions within the prologue of the index file and are relative to the start of each key descriptor.

The key definitions supply all the information needed by RMS to retrive, insert, update, and delete records for the Indexed file organization. The basic data which are contained in a key definition are as follows:

- Where the associated key field is positioned in the record, and how long it is.
- The VBN address of the associated Root bucket.
- Various key field options

The key definitions are linked into a chain by the VBN address and byte offset within the prologue block for the next key definition. The Indexed file organization can be viewed as a multi-partitioned file. The first partition is the prologue, the second partition is the index associated with the primary key definition, and the third partition is the user data associated with the primary index. Every indexed organized file contains these three partitions. In addition when alternate keys are defined then two additional partitions per alternate key are created. The first partition is the index associated with the alternate key definition, and the second partition is the RMS data associated with the index. The RMS data contain pointers into the user data partition for the records meeting the various key values. The index is structured as an n'ary tree where the nodes of the index are buckets. The index structure is the same for all key definitions.

7.3.1.1 K$NLVB 4 Bytes - VBN for Next Key Descriptor

This field contains the virtual block address in which the next key descriptor may be found. This field is only looked at when the K$BNYT field contains a 0. When K$NLVB and K$NBYT = 0 the last key descriptor has been found. The least significant 16 bits of the VBN are stored in K$NLVB and the most significant 16 bits are stored in K$NLVB+2 (K$NHVB).

7.3.1.2 K$NBYT 2 Bytes - Byte Offset for Next Key Descriptor

This word field contains the byte offset relative to the beginning of the VBN contained in K$NLVB for the next key descriptor in the chain of key descriptors. The first key descriptor contained in a VBN starts at byte offset 0, and the chain will thread through the current VBN before going to the next VBN. This means that the VBN will only change when K$NBYT contains a 0.

7.3.1.3 K$IAN 1 Byte - Index Area Number

This byte contains the number of the Allocation Area to use for the index buckets associated with this key starting at level 2 going up to and including the Root bucket.

7.3.1.4 K$LAN 1 Byte - Lowest Level Index Area Number

This byte contains the number of the Allocation Area to use for Level 1 of the index buckets associated with this key (a value of 0 means use the contents of K$IAN).

7.3.1.5 K$DAN 1 Byte - Data Level Area Number

This field contains the number of the Allocation Area to use for the data level (level 0) of the index buckets associated with this key descriptor.

7.3.1.6 K$LVL 1 Byte - Level of Root

This field contains the level number of the Root bucket associated with this key descriptor. This field is not supported by RMS-11 release one.

7.3.1.7 K$IBKS 1 Byte - Index Bucket Size

This field contains the bucket size in VBN'S for all index level (level 1 through the root level) buckets (1 - 32) for this key descriptor.

7.3.1.8 K$DBKS 1 Byte - Data Bucket Size

This field contains the bucket size in VBN'S for all data level (level 0) buckets (1 - 32) for this key descriptor.

7.3.1.9 P$DBKS 1 Byte - Data Bucket Size

This is a symbolic redefinition of K$DBKS for use by the Relative file organization.

7.3.1.10 K$LVBN 4 Bytes - Address of Root Bucket

This field contains the bucket address of the Root bucket for the index associated with this key descriptor. The 32 bit VBN is stored in the manner described in Section 7.2.1.1.

7.3.1.11 K$FLGS 1 Byte - Key Descriptor Flags

This field contains a bit vector for the various key options supported by RMS as follows:

XB$DUP	Duplicate key values allowed
XB$CHG	Key value may change on $UPDATE operation
XB$NUL	Null key character enabled (K$NULL)
XB$INI	Index must be initialized

When the XB$INI bit is set the K$LVBN field contains the following:

```
K$LVBN   = C(K$DAN)
K$LVBN+1 = C(K$IAN)
K$LVBN+2 = C(K$LAN)
K$LVBN+3 = 0 not used
```

This information is used once only when the index for this key definition is created. Since the area number information is not normally stored in the in memory data base for an open indexed file the required area numbers to create the index are stored in the root bucket field for this once only operation. The area numbers are not needed in the in memory data base since on future bucket allocation the area number stored in the bucket which is "splitting" is used as the area number to allocate the new bucket from (see section 7.5.1.1.2).

7.3.1.12 P$FLGS 1 Byte - Prologue Flags

This field is a symbolic redefinition of the K$FLGS field for use by the Relative file organization. Bits defined for this field are:

PR$NEX	Error encounted extending Relative file no further extending is possible.

7.3.1.13 K$DTP 1 Byte - Data Type for Key

This field contains the data type of the key field within the user data records. The only legal value currently for RMS-11 is XB$STG. The following data types are defined.

XB$STG	String data type (unsigned 8-bit bytes)
XB$IN2	Signed 15 bit integer (2-bytes)
XB$BN2	Unsigned 16 bit binary (2 bytes)
XB$IN4	Signed 31 bit integer (4-bytes)
XB$BN4	Unsigned 32 bit binary (4-bytes)
XB$PAC	Packed decimal (1-16 bytes)

7.3.1.14 K$NSEG 1 Byte - Number of Segments in Key

This field contains the number of segments (1 - 8) that make up the definition of the logical key field. The XB$IN2, XB$BN2, XB$IN4, XB$BN4, and XB$PAC key field data types may only contain one (1) segment.

7.3.1.15 K$NULL 1 Byte - "NULL" Character

This field contains a user specified character. If the key field within the data record associated with this key descriptor contains only "null" characters the record will not be inserted into the associated Index. The "null" value for the XB$IN2, XB$BN2, XB$IN4, XB$BN4, and XB$PAC key field data types is defined as zero (0). This field is enabled by the XB$NUL bit in the K$FLGS and is only valid for alternate keys.

7.3.1.16 K$KYSZ 1 Byte - Total Key Size

This field contains the sum of all the key segment sizes to yield the total size of the key field in bytes (1 - 255).

7.3.1.17 K$KEY 1 Byte - Key of Reference

This field contains the key of reference number (0 - 254) for this key descriptor. Primary key = 0; alternate keys = 1 - 254.

7.3.1.18 K$MINL 2 Bytes - Minimum Record Length

This field contains the minimum length record in bytes to contain the complete key field.

7.3.1.19 K$IFIL 2 Bytes - Index Fill Quantity

This field contains the number of bytes to use for index level buckets (levels 1 - n) before a bucket split is considered when the user requests RMS to follow fill quantities.

7.3.1.20 K$DFIL 2 Bytes - Data Fill Quantity

This field contains the number of bytes to use for user level buckets (level 0) before a bucket split is considered when the user requests RMS to follow fill quantities.

7.3.1.21 K$POS 16 Bytes - Key Segment Offset Positions

This is a set of eight (8) 2 byte fields (K$POS0-K$POS7) which contain the relative offset (0 - n) into the data record for each key segment.

7.3.1.22 K$SIZ 8 Bytes - Key Segment Size

This is a set of 8 1 byte fields (K$SIZ0-K$SIZ7) which contain the size in bytes for the key segment.

7.3.1.23 K$KNM 32 Bytes - Key Name

This is a 32 byte string supplied by the user when the key was defined. If not supplied will contain NULLS.

7.3.1.24 K$LDVB 4 Bytes - First Data Bucket

This field contains the bucket address of the first bucket at the data level (level 0) associated with this key descriptor. This field is not supported by RMS-11 release 1 and contains a zero.

7.3.1.25 14 Spare Bytes -

7.3.1.26 P$AVBN 1 Byte - VBN of First Area Descriptor

This field contains the VBN (2 - 255) of the first Allocation Area descriptor block. Allocation Area descriptor blocks are virtually contiguous and are directly accessed by area number. See Section 7.2.3.

7.3.1.27 P$AMAX 1 Byte - Maximum Number of Areas

This field contains the maximum number of defined Allocation Area descriptors (1 - 255) for this file. Eight (8) Allocation Area descriptor can fit in a virtual block since each area descriptor is 64 bytes long. The file address of any Area descriptor may be calculated as follows:

```
Let:        a = area number (0 - 254)
            v = VBN address for a
            o = offset into v for a

Then:       v = a/8 (truncated) + c(P$AVBN)
            o = (a mod 8)*64
```

7.3.1.28 P$DVBN 4 Bytes - Address of First Data Bucket

This field contains the 32 bit VBN of the first data bucket in a Relative file.

7.3.1.29 P$LMRN 4 Bytes - Maximum Record Number

This field contains the user specified maximum record number which will be allowed on $PUT operations to the Relative file organization. If the user specifies 0 then this field will contain the maximum record number possible (2**31-1).

7.3.1.30 P$LEOF 4 Bytes - EOF VBN

This field contains the last initialized (i.e., zeroed) VBN (i.e., the EOF VBN) for the Relative file organzation.

7.3.1.31 P$VERN 2 Bytes - Prologue Version Number

This field contains a prologue version nubmer. The only legal value at this time is one (1).

7.3.1.32 392 Bytes - Reserved for Future Use

7.3.1.33 2 Bytes - Prologue Checksum (see 7.2)

7.3.1.34 Prologue Block 1 Layout -

```
            |--------------------------------------|
            !          VBN For Next Key            ! K$NLVB
            |--                                  --|
            !              Descriptor              !
            |--------------------------------------|
            !        Offset To Next Key Descp.     ! K$NBYT
            |--------------------------------------|
            !                                      ! K$IAN
            |--------------------------------------|
K$LVL       !    Root Level      !    Data Area #  ! K$DAN
            |--------------------------------------|
K$DBKS      !     Data Bkts      !    Index Bkts   ! K$IBKS
P$DBKS      !      Size          !      Size       !
            |--------------------------------------|
            !              Root Bucket             ! K$LVBN
            |--                                  --|
            !                Pointer               !
            |--------------------------------------| K$FLGS
K$DTP       !     Data Type      !       Flags     ! P$FLGS
            |--------------------------------------|
K$NULL      ! "NULL" Character   ! # of key segments ! K$NSEG
            |--------------------------------------|
K$KEY       !    Key Of Ref.     !  Total Key Size ! K$KYSZ
            |--------------------------------------|
            !         Minimum Record Length        ! K$MINL
            |--------------------------------------|
            !         Index Fill Quantity          ! K$IFIL
            |--------------------------------------|
            !          Data Fill Quantity          ! K$DFIL
            |--------------------------------------|
            !                                      !
            !          Key Field Segment           !
            !                                      !
            !           Offset Positions           ! K$POS
            !                                      !
            !          (K$POS0-K$POS7)             !
            !                                      !
            !                                      !
            |--------------------------------------|
            !                  :                   ! K$SIZ
            !       Key Field Segment Sizes        !
            !          (K$SIZ0-K$SIZ7)             !
            !                  :                   !
            |--------------------------------------|
            !            Key Name String           !
            !              (32 Bytes)              ! K$KNM
            !                                      !
            |--------------------------------------|
            !           First Data Bucket          ! K$LDVB
            |--                                  --|
             !                 Pointer                  !
             !------------------------------------------!
             /              Spare (14 Bytes)            /
             !                                          !
             !------------------------------------------!
P$AMAX       !       Max Area #    ! VBN Of 1st Area    !P$AVBN
             !------------------------------------------!
             !       Start VBN of 1st Data Bucket       ! P$DVBN
             !--                                      --!
             !           (relative file only)           !
             !------------------------------------------!
             !              Maximum Record              ! P$LMRN
             !--                                      --!
             !                  Number                  !
             !------------------------------------------!
             !          Relative File EOF VBN           ! P$LEOF
             !--                                      --!
             !    (Last Initialized VBN - Zeroed)       !
             !------------------------------------------!
             !         Prologue Version Number          ! P$VERN
             !------------------------------------------!
             !                                          !
             /             Spare (392 Bytes)            /
             !                                          !
             !------------------------------------------!
             !              Block Checksum              !
             !              Byte Offset 510             !
             !------------------------------------------!
```

7.3.2 Alternate Key Prologue Blocks -

Alternate key prologue blocks are chained together through the K$NLVB field of the key descriptors (see Section 7.2.1.1). Five alternate key descriptors can fit in a VBN.

7.3.3 Area Descriptor Prologue Blocks -

The Indexed file organization requires a method of allocating the virtual blocks of the file to the various usages within the file (e.g., Index buckets and Data buckets). The structure which allows this virtual block allocation management is the Area Descriptor. The Indexed file supports multiple allocation areas to achieve the following user file design capabilities:

1. Different bucket sizes between the index buckets

and associated data buckets.

2. Different index and data bucket sizes on a per key basis.

3. Allocation placement control for the various elements of the file.

Eight area descriptor can be contained in a virtual block, and all the area descriptor prologue blocks are virtually contiguous (see Sections 7.2.1.26 and 7.2.1.27 for more details).

7.3.3.1 Spare 1 Byte -

7.3.3.2 A$FLG 1 byte - Flags (not used)

7.3.3.3 A$AID 1 Byte - Area Number (0 - 254)

This byte contains the Area's number and is used as a redundancy check since all area descriptors are located at a fixed relative position to the start of the Area Descriptor prologue blocks.

7.3.3.4 A$BKZ 1 Byte - Bucket Size for Area

This field contains the areas's bucket size in blocks (1 - 32) which is the granularity of allocation.

7.3.3.5 A$VOL 2 Byte - Relative Volume Number

This field contains the relative volume number for the last file extend for this area when placement control was requested.

7.3.3.6 A$ALN 1 Byte - Extend Allocation Alignment

This field contains the allocation alignment used for the last file extend for this area.

Legal values for this field are:

0	placement control not requested
XB$CYL	cylinder alignment (not implemented)
XB$LBN	logical block alignment
XB$VBN	virtual block alignment
XB$RFI	allocate close to related file by FID (not implemented)

7.3.3.7 A$AOP 1 Byte - Alignment Options

This field contains option bits to qualify the A$ALN field. Legal values are as follows:

XB$HRD	Alignment is absolute and fail if not available (note: illegal for XB$VBN or XB$RFI alignment).
XB$CTG	Allocation is to be contiguous.

7.3.3.8 A$AVL 4 Bytes - Available (Returned) Buckets

This field contains the 32 bit VBN of the first available bucket in a chain (linked through the first 4 bytes of the bucket) of buckets. This chain of buckets would be the result of returning buckets back to the area. The returning of buckets is not currently supported by RMS so that the only legal value for this field is zero (0).

7.3.3.9 A$CVB 4 Bytes - Start VBN for Current Extent

This field contains the 32 bit start VBN for the current extent. The current extent is the extent from which buckets will be allocated.

7.3.3.10 A$CNB 4 Bytes - Number of blocks in Current Extent

This field contains the number of blocks that were allocated to this current extent. The combination of A$CVB and A$CNB describes in virtual block terms the result of the file extend operation for the current extent.

7.3.3.11 A$NUS 4 Bytes - Number of blocks used

This field contains the number of blocks that have been allocated from the current extent.

7.3.3.12 A$NVB 4 Bytes - Next VBN to Use

This field contains the 32 bit VBN to use for the start VBN of the next bucket allocated from the current extent.

7.3.3.13 A$NXT 4 Bytes - Start VBN for Next Extent

This field contains the 32 bit start VBN for the next extent. When the current extent is used up the next extent is made the current extent and the next extent description is zeroed. The area can only be extended when the next extent description is zero.

7.3.3.14 A$XBY 4 Bytes - Number of blocks in Next Extent

This field contains the number of blocks that were allocated to this next extent. Thie combination of A$NXT and A$XBY describes in virtual block terms the result of the file extend operation for the next extent.

7.3.3.15 A$DEQ 2 Bytes - Default Extend Quantity

This field contains the user specified default file extend quantity to be used whenever the area is to be extended by RMS. A value of Ø means use the file's DEQ. However, in no case will less than one bucket size for this area be requested.

7.3.3.16 Reserved 2 Bytes -

7.3.3.17 A$LOC 4 Bytes - Start LBN on Volume

This field contains the start logical block number for the last extent performed for this area.

7.3.3.18 A$RFI 6 Bytes - Related File ID

This field contain the FID of a related file for the XB$RFI allocation alignment (A$ALN) (not implemented)

7.3.3.19 Spares 12 Bytes -

7.3.3.20 A$CRC 2 Bytes - Checksum

This field is a dummy field to pad out the area decriptor to 64 bytes. This also allows the standard Files-11 checksum to be stored in the last word of the Area Descriptor Prologue block.

7.3.3.21 Area Descriptor Layout -

```
          |---------------------------------------|
A$FLG     |        Flags        |     Spare       |
          |---------------------------------------|
A$BKZ     |    Bucket Size      |  Area Number    |   A$AID
          |---------------------------------------|
          |        Relative Volume Number         |   A$VOL
          |---------------------------------------|
A$AOP     |   Align Options     |  Alloc. Align.  |   A$ALN
          |---------------------------------------|
          |          Available Bucket             |   A$AVL
          |                List                   |
          |---------------------------------------|
          |            Start VBN For              |   A$CVB
          |            Current Extent             |
          |---------------------------------------|
          |          Number Of VBN's In           |   A$CNB
          |            Current Extent             |
          |---------------------------------------|
          |         Number Of VBN's Used          |   A$NUS
          |          In Current Extent            |
          |---------------------------------------|
          |          Next VBN To Use For          |   A$NVB
          |            Current Extent             |
          |---------------------------------------|
          |            Start VBN For              |   A$NXT
          |             Next Extent               |
          |---------------------------------------|
          |          Number Of VBN's In           |   A$XBY
          |             Next Extend               |
          |---------------------------------------|
          |        Default Extend Quantity        |   A$DEQ
          |---------------------------------------|
          |                Spare                  |
          |---------------------------------------|
          |         Start LBN For Last            |   A$LOC
          |         Extend For This Area          |
          |---------------------------------------|
          |             File ID For               |   A$RFI
          |           Related File For            |
          |             File Extends              |
          |---------------------------------------|
          |                                       |
          |                                       |
          |                Spares                 |
          |              (12 Bytes)               |
          |                                       |
          |                                       |
          |---------------------------------------|
          | Dummy Field To Allow Block Checksum   |   A$CRC
          |---------------------------------------|
```

7.4 Sequential File Format

The RMS Sequential file is compatible with the FCS Fixed and Variable length record files. Please refer to Section 6.2 through 6.2.3. The RMS variable with Fix Control record format is a generalization of the Sequenced Varialbe Length Records of FCS (Section 6.2.3) in that the fixed control area (always 2 bytes for FCS) can be varied between 1 to 255 bytes.

7.5 Relative File Format

The Relative file currently uses virtual block one (1) for its prologue, and starts its data buckets at virtual block 2. Records are stored in fixed length cells within unformated buckets (no overhead bytes in bucket) starting at byte 0 and packed end to end (i.e., byte aligned). The virtual blocks within the relative file must be initialized (zeroed) when they are allocated to the file to support deleted record control.

7.5.1 Relative File Record Formats -

Records are stored in fixed length cells. The first byte of each cell is a record control byte used to provide deleted record control. The following bits are defined:

DC$DEL	record has been deleted
DC$REC	record exists

A value of 0 indicates the cell has never contained a record.

The relative file supports variable and variable with fixed control length record up to the required user specified Maximum Record Size (MRS). In these cases the record control byte is followed with a two byte binary count of the bytes in the record (the count does not include itself). If the cell size does not evenly divide the bucket size then the remaining space in the bucket is dead space and the next record in the file will be stored in the first cell of the next bucket. In other words records never span bucket boundaries.

7.5.1.1 Fixed Length Records -

```
        ---------------------------
        | ctrl | data (mrs bytes) |
        ---------------------------

        cell size = MRS+1
```

7.5.1.2 Variable Length Records -

```
        ----------------------------------------
        | ctrl | size | data (size bytes) |    |
        ----------------------------------------

        cell size = MRS+3
```

7.5.1.3 Variable With Fixed Control Records -

```
-------------------------------------------------------------
| ctrl | size | fixed | data (size-fixed ctrl bytes) |      |
-------------------------------------------------------------

    cell size = MRS+fixed ctrl size+3
```

7.6 Indexed File Format

The Indexed File uses virtual blocks 1, 2 and if necessary up to and including 84 as a maximum for its prologue. The current implementation on the PDP-11 will result in a prologue of the following forms:

```
                Single Key

                     |-------------------------------|
                     |          Primary Key          |
VBN 1                |          Description          |
                     |                               |
                     |-------------------------------|
                     |     P$AMAX     |    P$AVBN    |----
                     |-------------------------------|   |
                     |                               |   |
                     |-------------------------------|   |
                                                         |
                                                         |
                     |-------------------------------|   |
VBN 2                |                               |<---
```

```
                        |        Area Descriptors         |
                        |         (Up To 8) For           |
                        |   single key 4 is all that      |
                        |          can be used            |
                        |---------------------------------|


VBN3-N                  |---------------------------------|
                        |          Index and Data         |
                        |             Buckets             |
                        |                                 |
                        |---------------------------------|



              Multiple Key

                        |---------------------------------|
VBN 1                   |           Primary Key           |----
                        |           Descriptor            |   |
                        |                                 |   |
                        |---------------------------------|   |
                        |     P$AMAX     |     P$AVBN     |   |
                        |---------------------------------|   |
                        |                                 |   |
                        |---------------------------------|   |
                                                              |
                        |---------------------------------|   |
VBN 2                   |                                 |<---
                        |                                 |
                        |           Up To 5 Key           |
                        |           Descriptors           |
                        |                                 |
                        |---------------------------------|
                        |         Key 5 Descriptor        |----
                        |                                 |   |
                        |---------------------------------|   |
If more than 5 alternate keys                                 |
                                                              |
                        |---------------------------------|   |
VBN 3                   |                                 |<---
                        |         Key Descriptors         |
                        |              etc.               |
                        |                                 |
                        |---------------------------------|
                                   .
                                   .
                                   .
                        |---------------------------------|
VBN P$AVBN              |        Area Descriptors         |
                        |        8 Per Block              |
                        |                                 |
                        |---------------------------------|
```

index and data bucket space starts at:

((P$AMAX/8(truncated))+P$AVBN)

Records are stored in formatted buckets (buckets have overh-ead bytes) and are packed end to end (i.e., byte aligned). The bucket format and the various record formats are given in the following sections.

7.6.1 Index Structure -

The Index is structured as a balanced tree. The nodes in the tree are buckets, and the nodes are serially searched. The Index node contains index records as specified in Section 7.5.2.1.

The bucket size is constant for index nodes, but may be different than the Data buckets. The Data buckets are all the same size.

Each level of the index is horizontaly linked via the Next bucket pointers. The horizontal linking is circular with the last bucket (noted by BC$LBK) pointing back to the first bucket. The Data buckets for an Index may be viewed as the data level (set) of the index and are linked in the same manner as buckets in any other level of the Index. Figure 7-2 shows the structure of the Index.

The key value associated with index records (see Section 7.5.2.1) is the highest or highest possible key value in the bucket pointed to by the bucket pointer in the record.

The basic search rule for an index search is to follow the first path for which the search key is equal to or less than the key value stored in the index record.

7.6.1.1 Primary Key Index Structure -

The primary key index for a file is structured as stated in Section 7.5.0 above where the data level is composed of buckets which contain the User's data records. The data buckets may also contain RRV records. See Section 7.5.3 and 7.5.2.3 for details on RRV records.

7.6.1.2 Alternate Key Index Structure -

An alternate key index for a file is structured as stated in Section 7.5.0 above where the data level is composed of buckets which contain pointer array records as specified in Section 7.5.2.4. Therefore the indices within the Indexed File Organization have the same structure, where only the interpretation of the records within the data level of an index is different.

7.6.2 Record Reference Vector (RRV) -

When a record is inserted in an Indexed file the record is assigned a reference vector address and this address is stored in the data record in the record pointer field (see Section 7.5.2.2). This address is the initial address of the record itself. Whenever the record is moved the record's reference vector record is updated with its new address. The record, in turn, points back to its reference vector so that it can be updated if the record is moved again. The reference vector record is created when the record is moved for the first time. Using this technique the worst case indirection for a record is kept at one, and we can always find the record via its reference vector address.

The record pointers used within the Indexed file organization, and the RFA (Record's File Address) returned to the user in the RFA field of the RAB are always the record's reference vector address.

The space required for RRV pointers in the data records of a file is required to insure RFA addressing and alternate keys. The RRV records are stored at the end of the data records in the user data buckets. The use of RRV's and secondary indices is graphically shown in Figure 7-3.

7.6.3 Bucket Format -

The Indexed organization uses a formatted bucket as its primary unit of seondary storage. A bucket is composed of some number of virtual blocks in the range of 1-32 and has a header starting at byte one of the bucket.

The Bucket is composed of three logical areas, a Header area, a Record storage area and a Free space area.

Each of these areas will be described in the sections that follow.

7.6.3.1 Header Area -

The bucket header area is composed of a RAS data section, a bucket storage control section, and a structure link section. The size of the bucket header is 14 bytes (S$BHD).

7.6.3.1.1 B$CHK 1 Byte - Check Byte

This is a one byte check character. Whenever a bucket is written the value in the check byte is changed and copied into the last byte of the bucket. Whenever a bucket is read the check byte is compared to the copy for equality. By this technique hardware failures during transfer are detectable (i.e., the BUS breaks etc.).

7.6.3.1.2 B$TAA 1 Byte - This Allocation Area

This field contains the allocation area number that this bucket was allocated from.

7.6.3.1.3 B$ADR 2 Bytes - Bucket Address Sample

This is a sample of the bucket's start VBN address, and is composed of the low order 16 bits of that address. This field is written upon bucket formatting, and is checked whenever the bucket is read into main memory.

7.6.3.1.4 B$NBY 2 Bytes - Next Available Byte

This field contains the byte address relative to the start of the bucket of the first free byte in the Free Storage Area of the bucket.

7.6.3.1.5 B$NID 1 Byte - Next Available ID

This field contains the ID number to use for the next record placed in the bucket.

7.6.3.1.6 B$LID 1 Byte - Last Available ID

This field contains the ID number of the last ID in the contiguous range of ID's specified by the contents of B$NID and B$LID. When the contents of B$NID are greater than the contents of B$LID or is zero then there is no "next" available ID. When this condition occurs the bucket is scanned to find the largest contiguous range of unused ID's and B$NID and B$LID are updated to describe that range.

7.6.3.1.7 B$NBK 4 Bytes - Next Bucket Pointer

This field contains the start VBN of the next bucket at this level of the index or data partition for the Indexed file organization. This pointer always points to a bucket of the same size.

7.6.3.1.8 B$LEV 1 Byte - Level Number for Bucket

This field contains the level number relative to the data level for this bucket, in the index. The Data level buckets contain a 0, the lowest level buckets of the index contain a 1, the next level buckets going towards the root contain a 2 etc.

NOTE

"Data buckets" refer to the buckets which contain the data records associated with the index. For the primary index these are the user data records, and for the alternate index these are system data records which contain an array of pointers to user data records.

7.6.3.1.9 B$BCB 1 Byte - Control Bits

This is a bit encoded byte field and is used in the processing of a bucket. The following bits are defined for the indexed file organization:

BC$LBK - last bucket in level
BC$ROT - root bucket of index

7.6.3.2 Record Storage Area -

The record storage area starts at the first byte after the bucket header area, and ends at the byte address stored in B$NBY minus one. The record structures in buckets vary with the use of the bucket. Section 7.5.2 specifies the various record structures used.

7.6.3.3 Free Storage Area -

The free storage area starts at the byte address stored in B$NBY and up to the check byte copy in the bucket. Any and all free storage statistics refer to this contiguous free storage area. However it is possible due to "fast" record deletions to have "free" space within the record storage area of the bucket. The reclaiming of this space is done on an as needed basis.

7.6.3.4 S$BHD 14 Bytes - Size of Header Area

This symbol represents the size of the bucket header area.

7.6.3.5 Bucket Format Layout -

```
          |-----------------------------------------|
B$TAA     |     This Area       |    Check Byte     |   B$CHK
          |-----------------------------------------|
          |          Bucket Address Sample          |   B$ADR
          |-----------------------------------------|
          |           Next Available Byte           |   B$NBY
          |-----------------------------------------|
B$LID     |      Last ID        |     Next ID       |   B$NID
          |-----------------------------------------|
          |           Next Bucket Pointer           |   B$NBK
          |              (Start VBN)                |
          |-----------------------------------------|
B$BCB     |        BCB          |      Level        |   B$LEV
          |-----------------------------------------|   S$BHD
```

```
        |                                       |
        |            Record Storage             |
        |                 Area                  |
        |                                       |
        |---------------------------------------| c(B$NBY)
        |                                       |
        |              Free Space               |
        |                 Area                  |
        |                                       |
        |-------------------|                   |
        |  Check Byte Copy  |                   |
        |-------------------|-------------------|
```

7.6.4 Record Structures -

The following record structures apply to the Indexed file organization.

7.6.4.1 Index bucket record -

```
        |-----------|
        |      |    |
        | IRCB | PS |  1 Byte
        |      |    |
        |-----------|
        |           |
        |  Bucket   |  n Bytes
        |  Pointer  |
        |           |
        |-----------|
        |           |
        | Key Value |  m Bytes
        |           |
        |-----------|
```

IRCB contains Index Record Control Bits

The following bits are defined in the IRCB byte:

IC$KCP	Compressed key value (not currently defined).
IC$EMP	Pointer to empty bucket.

PS is the pointer size as follows:

0 = 2 byte bucket pointer
1 = 3 byte bucket pointer
2 = 4 byte bucket pointer

3 = undefined

7.6.4.2 General Data Bucket Record -

```
|-------------|
| DRCB | PS   |  1 Byte
|-------------|
|     ID      |  1 Byte
|-------------|
|   Record    |  N Bytes Optional
|   Pointer   |
|-------------|
|    Size     |  No Size If Fixed Length Data
|-------------|
|             |
|    Data     |  M Bytes
|             |  M = Size or Fixed Length
|-------------|
```

DRCB contains Data Record Control Bits

The following bits are defined in the DRCB byte:

DC$DEL	Record deleted, or pointer to deleted record.
DC$RRV	Record reference vector record.
DC$NPS	No pointer size field present (qualifies PS)
DC$KDL	Pointer to record for this key no longer applies $UPDATE changed the key, but record exists; note ID will be zeroed on all systems starting with Release 1 on RSX-11M V3.
DC$NCP	Do not compress this deleted record.

PS is the pointer size for the Record pointer as follows:

0 = 3 byte record pointer
1 = 4 byte record pointer
2 = 5 byte record pointer
3 = undefined

7.6.4.3 RRV Records -

Record Reference Vector (RRV) records are records which point to the record associated with the reference vector. They function as "forwarding addresses" for the actual records when they are moved.

The format is as follows:

```
|-----------|
! DRCB | PS |
|-----------|
!    ID     !
|-----------|
!  Record   !
!  Pointer  !
|-----------|
```

where the DC$RRV bit is set in the DRCB field.

7.6.4.4 Deleted RRV Records -

The RRV record for a deleted record can be as small as the first two bytes of the RRV record. In this case the following DRCB bits are set:

```
DC$RRV
DC$NPS
DC$DEL
```

7.6.4.5 Secondary (or alternate) Index Data Record (SIDR) for which duplicate keys are allowed -

The data records associated with an alternate index are pointer arrays to the users data records. The format of the record is as follows:

```
- - - - -      |--------------------|
               !       DRCB    | PS |  1 Byte
               |--------------------|
Data Record    |         ID         !  1 Byte
               |--------------------|
               !  Duplicate Count   !  4 Bytes (DC$NPS=0)
               |--------------------|
Overhead       !        Size        !  2 Bytes
- - - - -      |--------------------|
               !     Key Value      !  M Bytes
               |--------------------|
Data            !      SIDR Record       !  X Byte     Pointer
 on             !      Pointer #1        !             Array
Record          |-----------------------|
                !      SIDR Record       !  Y Bytes     record
                !      Pointer #2        !
                |-----------------------|
                !           .            !
                !           .            !
                !           .            !
                |-----------------------|
                !      SIDR Record       !  Z Bytes
                !      Pointer #K        !
- - - - -       |-----------------------|
```

Fields within the pointer array record:

PS This field contains the size of the duplicate count field as follows

0 = 3 bytes
1 = 4 bytes **THIS IS THE ONLY VALUE USED**
2 = 5 bytes
3 = undefined

DRCB Bits used for pointer array records

NC$NPS If this bit is set then there is no duplicate count field. This is used for all array continuations records, since the count applies to the total array.

7.6.4.6 Secondary (alternate) Index Data Record - No Duplicates -

The data records associated with an alternate index for which duplicate key values are not allowed is shown in Section 7.5.2.4 except that the duplicate count field is omitted (DC$NPS=1) and there is only one SIDR Record Pointer.

NOTE

When a record is deleted the No Duplicates SIDR record is compressed out of the secondary index's data bucket at the time of the delete.

7.6.4.7 SIDR Record Pointers -

The format of the record pointers used in Secondary Index Data Records is as follows:

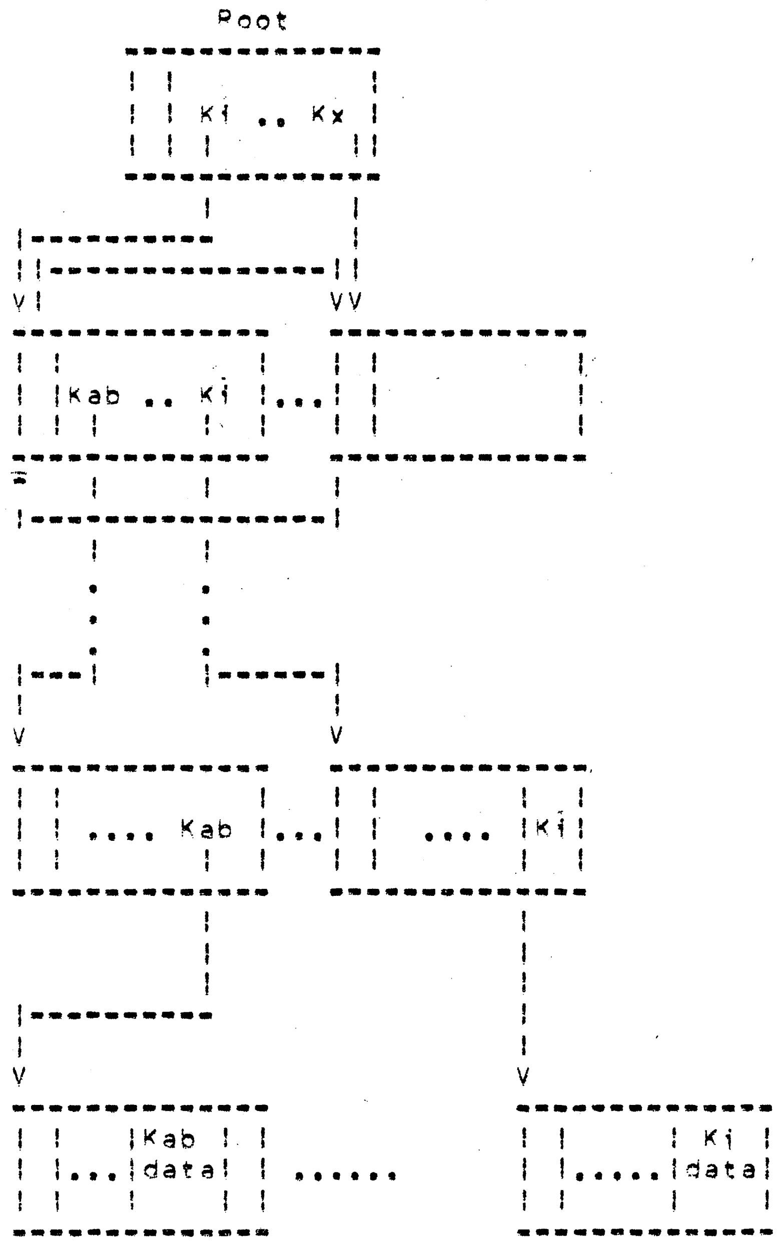
```
- - - - - -------------
Overhead  | DRCB | PS | 1 Byte
- - - - - -------------
Record    |  Record   | N Bytes
Pointer   |  Pointer  |
3-5 bytes |           |
- - - - - -------------
```

DRCB bits used for SIDR record pointers:

DC$KDL	Pointer has been deleted due to key change on a $UPDATE operation. In this case the ID portion of the record pointer will be zero.
DC$DEL	Record associated with this pointer has been deleted.

Figure 7-2

Index Structure



NOTES:

All buckets in a level are linked horizontally from left to right via next bucket pointers (see Section 7.5.1.1.7).

CASE 1: Record Has Never Moved

```
        ---- Pointer In Secondary Index
        !    Pointer Array
        !
        V
       -----------------
       !    DRCB   ! PS !  <---
       -----------------      !
       !       ID       !     !
       -----------------      !   User Data Record
       ! Record Pointer !------
       !    (RRVP)      !
       -----------------
       !      Data      !   RRVP = Records Reference
       -----------------    Vector Pointer
```

CASE 2: Record Has Moved

```
        ---- Pointer In Secondary Index
        !    Pointer Array
        !
        V
       -----------------
       !    DRCB   ! PS !<-------  Record Reference
       -----------------        !   Vector
       !       ID       !       !
       -----------------        !
       ! Record Pointer !----   !
       -----------------    !   !
                            !   !
        --------------------    !
        !                       !
        !                       !
        V                       !
       -----------------        !
       !    DRCB   ! PS !       !
       -----------------        !
       !       ID       !       !
       -----------------        !
       ! Record Pointer !--------
       !    (RRVP)      !        User Data Record
       -----------------
       !      Data      !
       -----------------
```

Figure 7-3

RRV Usage

[End of ODS2.RNO]