Files-11 On-Disk Structure Specification 28 NOV 77

Andrew C. Goldstein
6/11 Software Development
ML5/E40 Ext. 5054

19 Jun 1975

COMPANY CONFIDENTIAL

Revised 15 June 1977

E. H. Marison -- Addition of RMS structures

Document 130-958-032-02

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 1 of Files-11, also referred to as ODS-1 (on-disk structure version 1). This is the only version of Files-11 which is implemented on any of the supporting operating systems. A proposed structure level 2 (ODS-2) exists but has not been implemented anywhere.

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 theoretically 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; current implementations of Files-11 will handle volumes up to 2**24 blocks.

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.

2.2 Volume Sets

A volume set is a collection of related units that are normally treated as one logical device in the usual operating system concept. Each unit contains its own Files-11 structure; however, files on the various units in a volume set may be referenced with a relative volume number, which uniquely determines which unit in the set the file is located on. Other sections in this specification will make occasional reference to volume sets and relative volume numbers where hooks for their implementation exist. Since volume sets have not been implemented as yet, however, no complete specification is provided here.

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 blocks have been allocated to the file. The number assigned to a virtual block is called (obviously) its virtual block number, or VBN. Each virtual block is mapped to a unique logical block 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.

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 unit of the volume set. File numbers must lie in the range 1 through 65535. The set of file numbers on a unit is moderately (but not totally) dense; at any

instant in time a file number uniquely identifies one file within that unit.

Word 2 File Sequence Number

Identifies the current use of an individual file number on a unit. 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 unit of a volume set the file is located on. Volume sets are at present not implemented; the only legal value for the relative volume number in any context is zero.

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 four areas, of which the first three 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. Finally, the header area contains a user attribute area, which may be used by the user to store a limited amount of data describing the file.

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 logically contiguous segment of the file. The map area also contains the linkage to the next extension header of the file, if such exists.

3.2.4 End Checksum

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

3.3 Extension Headers

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 map area links the headers together in order of ascending virtual block numbers.

Multiple headers are also needed for files that span units in a volume set. A header may only map logical blocks located on its unit; therefore a multi-volume file is represented by headers on all units that contain portions of that file.

3.4 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 FHDOF$, 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.

3.4.1 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.

3.4.1.1 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.

3.4.1.2 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.

3.4.1.3 H.FNUM 2 Bytes File Number

This word contains the file number of the file.

3.4.1.4 H.FSEQ 2 Bytes File Sequence Number

This word contains the file sequence number of the file.

3.4.1.5 H.FLEV 2 Bytes File Structure Level

The file structure level 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 version 1 of Files-11; the only legal contents for H.FLEV is 401 octal.

3.4.1.6 H.FOWN 2 Bytes File Owner UIC

H.PROG = H.FOWN+0 Programmer (Member) Number
H.PROJ = H.FOWN+1 Project (Group) Number

This word contains 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.

3.4.1.7 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 intents are defined in Files-11: read, write, extend, 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.EXT Deny extend 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.

3.4.1.8 H.FCHA 2 Bytes File Characteristics
H.UCHA = H.FCHA+0 User Controlled Char.
H.SCHA = H.FCHA+1 System Controlled Char.

The user controlled characteristics byte contains 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 unit 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.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.

The system controlled characteristics byte contains the following flag bits:

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.

3.4.1.9 H.UFAT 32 Bytes User Attribute Area

This area is intended for the storage of a limited quantity of "user file attributes", i.e., any data the user deems useful for processing the file that is not part of the file itself. An example of the use of the user attribute area is presented in section 6.1 (FCS File Format).

3.4.1.10 S.HDHD 46 Bytes Size of Header Area

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

3.4.2 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.

3.4.2.1 I.FNAM 6 Bytes File Name

These three words contain the name of the file, packed three Radix-50 characters to the word. This name usually, but not necessarily, corresponds to the name of the file's primary directory entry.

3.4.2.2 I.FTYP 2 Bytes File Type

This word contains the type of the file in the form of three Radix-50 characters.

3.4.2.3 I.FVER 2 Bytes Version Number

This word contains the version number of the file in binary form.

3.4.2.4 I.RVNO 2 Bytes Revision Number

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

3.4.2.5 I.RVDT 7 Bytes Revision Date

The revision date is the date on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the form "DDMMMYY", where DD is two digits representing the day of the month, MMM is three characters representing the month, and YY is the last two digits of the year.

3.4.2.6 I.RVTI 6 Bytes Revision Time

The revision time is the time of day on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the format "HHMMSS", where HH is the hour, MM is the minute, and SS is the second.

3.4.2.7 I.CRDT 7 Bytes Creation Date

These seven bytes contain the date on which the file was created. The format is the same as that of the revision date above.

3.4.2.8 I.CRTI 6 Bytes Creation Time

These six bytes contain the time of day at which the file was created. The format is the same as that of the revision time above.

3.4.2.9 I.EXDT 7 Bytes Expiration Date

These seven bytes contain the date on which the file becomes eligible to be deleted. The format is the same as that of the revision and creation dates above.

3.4.2.10 - 1 Byte (unused)

This unused byte is present to round up the size if the ident area to a word boundary.

3.4.2.11 S.IDHD 46 Bytes Size of Ident Area

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

3.4.3 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.

3.4.3.1 M.ESQN 1 Byte Extension Segment Number

This byte 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.

3.4.3.2 M.ERVN 1 Byte Extension Relative Volume No.

This byte contains the relative volume number of the unit 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 unit as this header, this byte contains 0.

3.4.3.3 M.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.

3.4.3.4 M.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.

3.4.3.5 M.CTSZ 1 Byte Block Count Field Size

This byte contains a count of the number of bytes used to represent the count field in the retrieval pointers in the map area. The retrieval pointer format is described in section 3.4.3.9 below.

3.4.3.6 M.LBSZ 1 Byte LBN Field Size

This byte contains a count of the number of bytes used to represent the logical block number field in the retrieval pointers in the map area. The contents of M.CTSZ and M.LBSZ must add up to an even number.

3.4.3.7 M.USE 1 Byte Map Words In Use

This byte contains a count of the number of words in the map area that are presently occupied by retrieval pointers.

3.4.3.8 M.MAX 1 Byte Map Words Available

This byte contains the total number of words available for retrieval pointers in the map area.

3.4.3.9 M.RTRV variable Retrieval Pointers

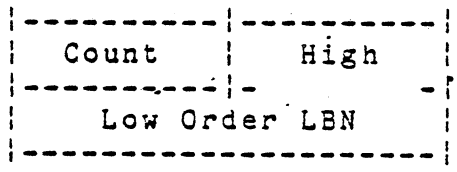
This area contains the retrieval pointers that actually map the virtual blocks of the file to the logical blocks of the volume. Each retrieval pointer describes a consecutively numbered group of logical blocks which is part of the file. The count field contains the binary value n to represent a group 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.

Although the data in the map area provides for arbitrarily extensible retrieval pointer formats, Files-11 has defined only three. Of these, only the first is currently implemented; the other two are presented out of historical interest and because they may be resurrected in the future.

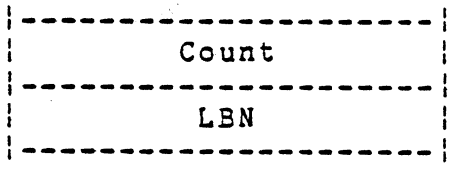
Format 1: M.CTSZ = 1
M.LBSZ = 3

The total retrieval pointer length is four bytes. 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.



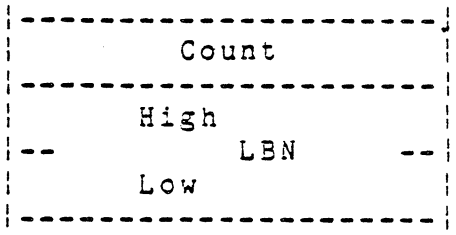
Format 2: M.CTSZ = 2
M.LBSZ = 2

The total retrieval pointer length is four bytes. The first word contains a 16 bit count field and the second word contains a 16 bit LBN field.



Format 3: M.CTSZ = 2
M.LBSZ = 4

The total retrieval pointer length is six bytes. The first word contains a 16 bit count field and the second and third words contain a 32 bit LBN field.



3.4.3.10 S.MPHD 10 Bytes Size of Map Area

This symbol represents the size of the map area, not including the space used for the retrieval pointers.

3.4.4 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.

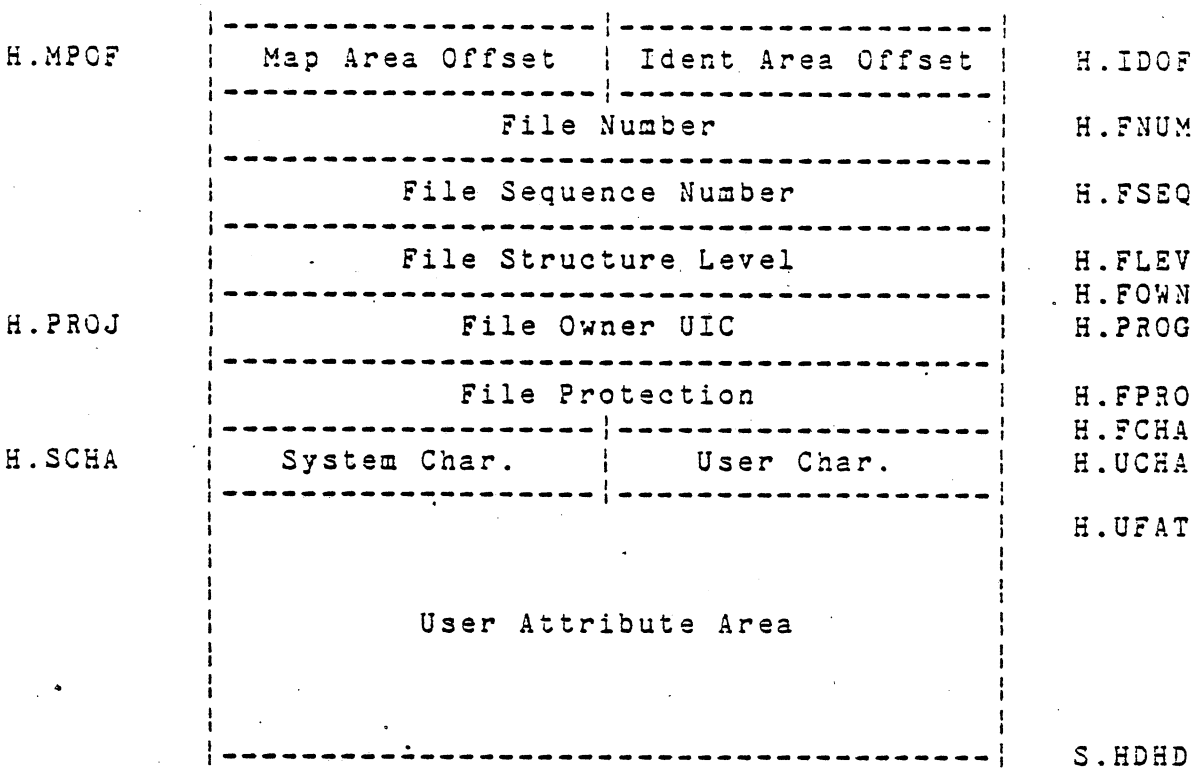
3.4.4.1 H.CKSM 2 Bytes Block Checksum

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:

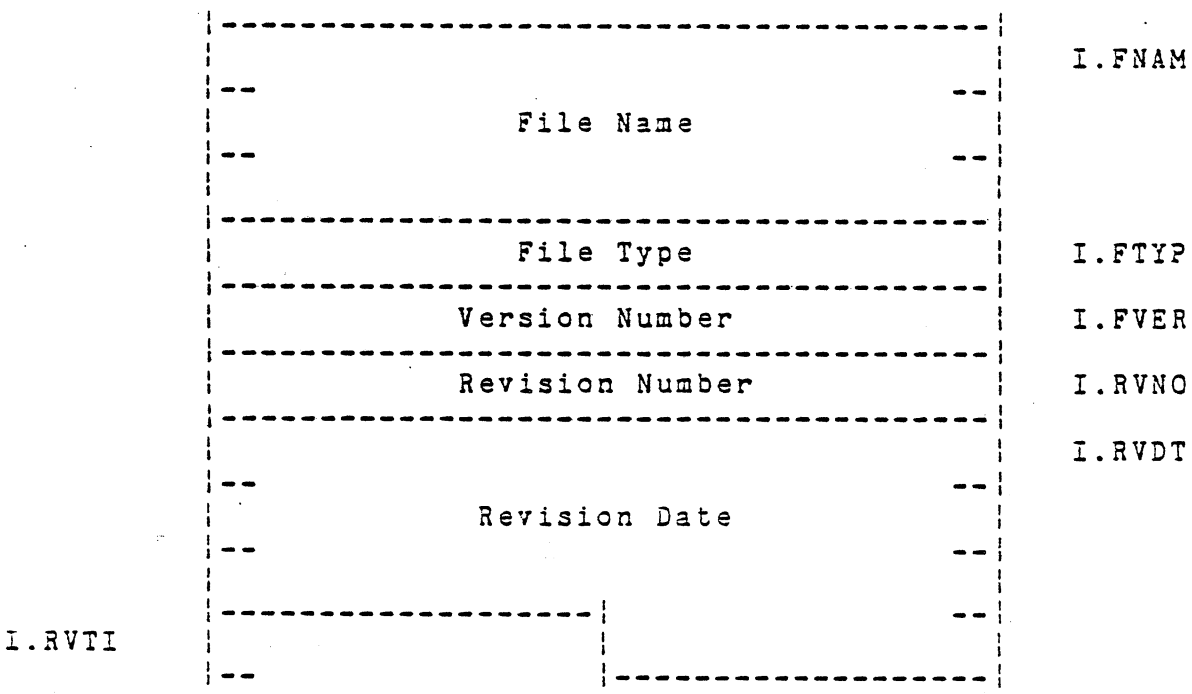
```
        MOV     Header-address,R0
        CLR     R1
        MOV     #255.,R2
10$:    ADD     (R0)+,R1
        SOB     R2,10$
        MOV     R1,(R0)
```

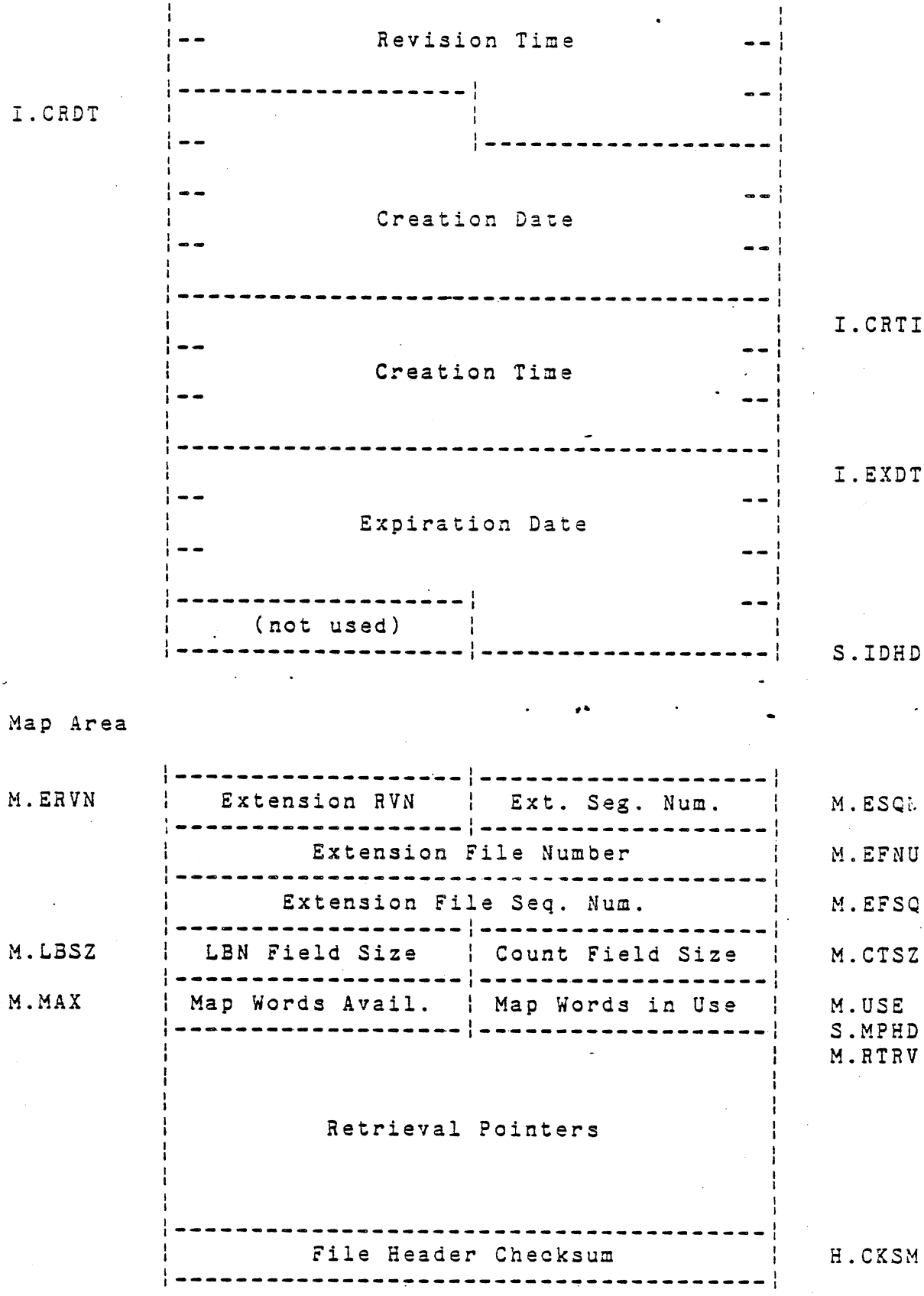
3.4.A File Header Layout

Header Area



Ident Area




Revision Time
I.CRDT
Creation Date
I.CRTI
Creation Time
I.EXDT
Expiration Date
(not used)
S.IDHD
Map Area
M.ERVN
Extension RVN
Ext. Seg. Num.
M.ESQN
Extension File Number
M.EFNU
Extension File Seq. Num.
M.EFSQ
M.LBSZ
LBN Field Size
Count Field Size
M.CTSZ
M.MAX
Map Words Avail.
Map Words in Use
M.USE
S.MPHD
M.RTRV
Retrieval Pointers
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 sole function is to associate file name strings with File ID's.

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 User File Directories

Current implementations of Files-11 all support a two level directory heirarchy which is tied in with the user identification mechanism of the operating system. Each UIC 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. All UFD's are listed in the volume's MFD under a file name constructed from the UIC. A UIC 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.

4.2 Directory Structure

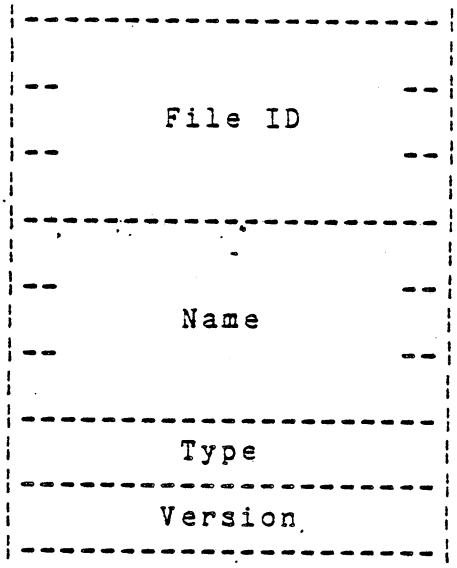
A directory is a file consisting of 16 byte records. It is structured as an FCS fixed length record file, with no carriage control attributes (see section 6 for a description of FCS files). Each record is a directory entry. The entries are not required to be ordered, or densely packed, nor do they have any other relationship to each other, except that no two entries in one directory may contain the same name, type, and version. Each entry contains the following:

File ID — The three word binary File ID of the file that this directory entry represents. If the file number portion of the File ID field is zero, then this record is empty and may be used for a new directory entry.

Name — The name of the file may be up to 9 characters. It is stored as three words, each containing three Radix-50 packed characters.

Type — The type of the file (also historically referred to as the extension) may be up to three characters. It is stored as one word of Radix-50 packed characters.

Version — The version number of the file is stored in binary in one word.



4.3 Directory Protection

Since directories are files with no special characteristics, they may be accessed like all other files, and are subject to the same protection mechanism. However, implementations of Files-11 support three special functions for the management of directories, namely FIND, REMOVE, and ENTER. A user performing such a directory operation must have the following privileges to be allowed the various functions:

FIND: READ
REMOVE: READ, WRITE
ENTER: READ, WRITE, EXTEND

5.0 Known 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 five files. These files are created when a new volume is initialized. They are unique in that their File ID's are known constants. These five files have the following uses:

File ID 1,1,0 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 2,2,0 is the storage bitmap file. It is used to control the allocation of logical blocks on the volume.

File ID 3,3,0 is the bad block file. It is a file containing all of the known bad blocks on the volume.

File ID 4,4,0 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 5,5,0 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.

5.1 Index File

The index file is File ID 1,1,0. 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).

5.1.1 Bootstrap Block

Virtual block 1 of the index file is the volume's boot block. It is 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.

5.1.2 Home Block

Virtual block 2 of the index file is the volume's home block. The logical block containing the home block is the first good block on the volume out of the sequence 1, 256, 512, 768, 1024, 1280, ... 256*n. 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.

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 HMBOF$, 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.

5.1.2.1 H.IBSZ 2 Bytes Index File Bitmap Size

This 16 bit word contains the number of blocks that make up the index file bitmap. (The index file bitmap is discussed in section 5.1.3.) This value must be non-zero for a valid home block.

5.1.2.2 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. The LBN is stored with the high order in the first 16 bits, followed by the low order portion. This value must be non-zero for a valid home block.

5.1.2.3 H.FMAX 2 Bytes Maximum Number of Files

This word contains the maximum number of files that may be present on the volume at any time. This value must be non-zero for a valid home block.

5.1.2.4 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. Volume clustering is not implemented at present; the only legal value for this item is 1.

5.1.2.5 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.

5.1.2.6 H.VLEV 2 Bytes Volume Structure Level

This word identifies the volume's structure level. Like the file structure level, this word identifies the version of Files-11 which created this volume and permits upwards compatibility of media as Files-11 evolves. The volume structure level is affected by all portions of the Files-11 structure except the contents of the file header. This document describes Files-11 version 1; the only legal value for the structure level is 401 octal.

5.1.2.7 H.VNAM 12 Bytes Volume Name

This area contains the volume label as an ASCII string. It is padded out to 12 bytes with nulls. The volume label is used to identify individual Files-11 volumes.

5.1.2.8 - 4 Bytes Not Used

5.1.2.9 H.VOWN 2 Bytes Volume Owner UIC

This 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.

5.1.2.10 H.VPRO 2 Bytes Volume Protection Code

This word contains the protection code for the entire volume. Its contents are coded in the same manner as the file protection code stored in the file header, and it is interpreted in the same way in conjunction with the volume owner UIC. All operations on all files on the volume must pass both the volume and the file protection check to be permitted. (Refer to the discussion on file

protection in section 3.4.1.7).

5.1.2.11 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.

5.1.2.12 H.FPRO 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.

5.1.2.13 - 6 Bytes Not Used

5.1.2.14 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.

5.1.2.15 H.FIEX 1 Byte Default File Extend

This byte 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.

5.1.2.16 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.

5.1.2.17 - 11 Bytes Not Used

5.1.2.18 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 3.4.4.1).

5.1.2.19 H.VDAT 14 Bytes Volume Creation Date

This area contains the date and time that the volume was initialized. It is in the format "DDMMMYYHHMMSS", followed followed by a single null. (The same format is used in the ident area of the file header, section 3.4.2).

5.1.2.20 - 398 Bytes Not Used

This area is reserved for the relative volume table for volume sets.

5.1.2.21 H.INDN 12 Bytes Volume Name

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

5.1.2.22 H.INDO 12 Bytes Volume Owner

This area contains an ASCII expansion of the volume owner UIC in the form "[proj,prog]". Both numbers are expressed in decimal and are padded to three digits with leading zeroes. The area is padded out to 12 bytes with trailing spaces. It is placed here in accordance with the proposed volume identification standard.

5.1.2.23 H.INDF 12 Bytes Format Type

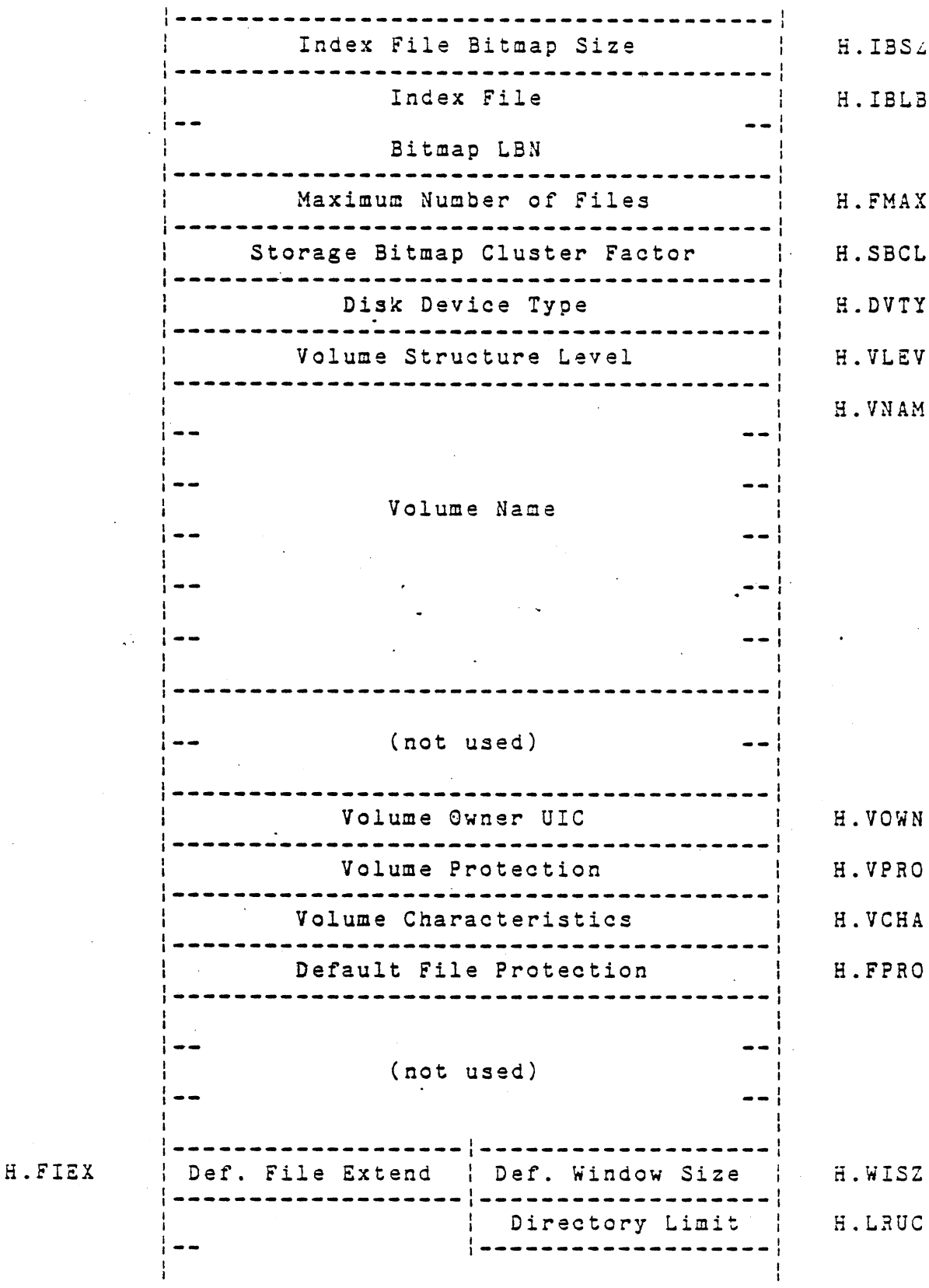
This field contains the ASCII string "DECFILE11A" padded out to 12 bytes with spaces. It identifies the volume as being of Files-11 format. It is placed here in accordance with the proposed volume identification standard.

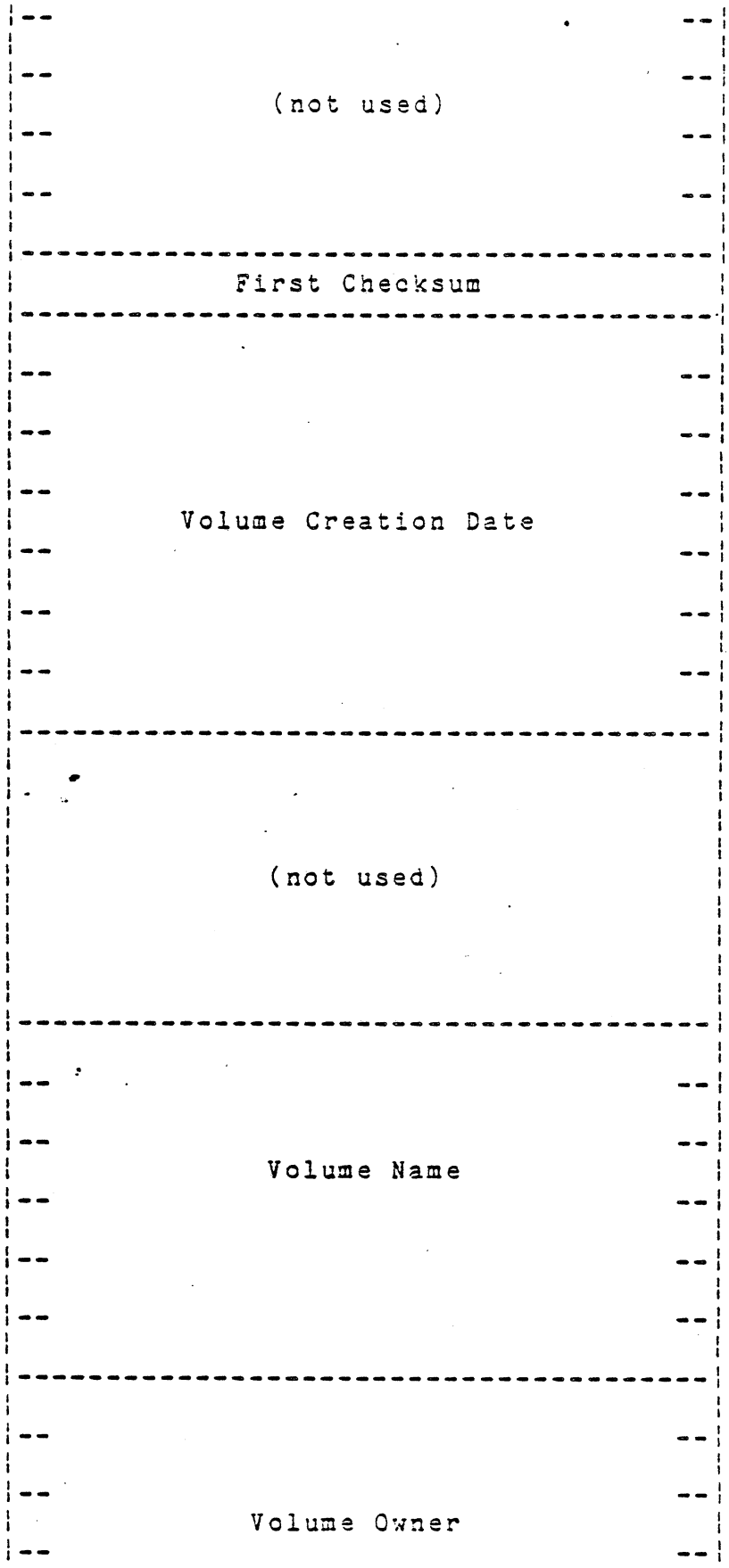
5.1.2.24 - 2 Bytes Not Used

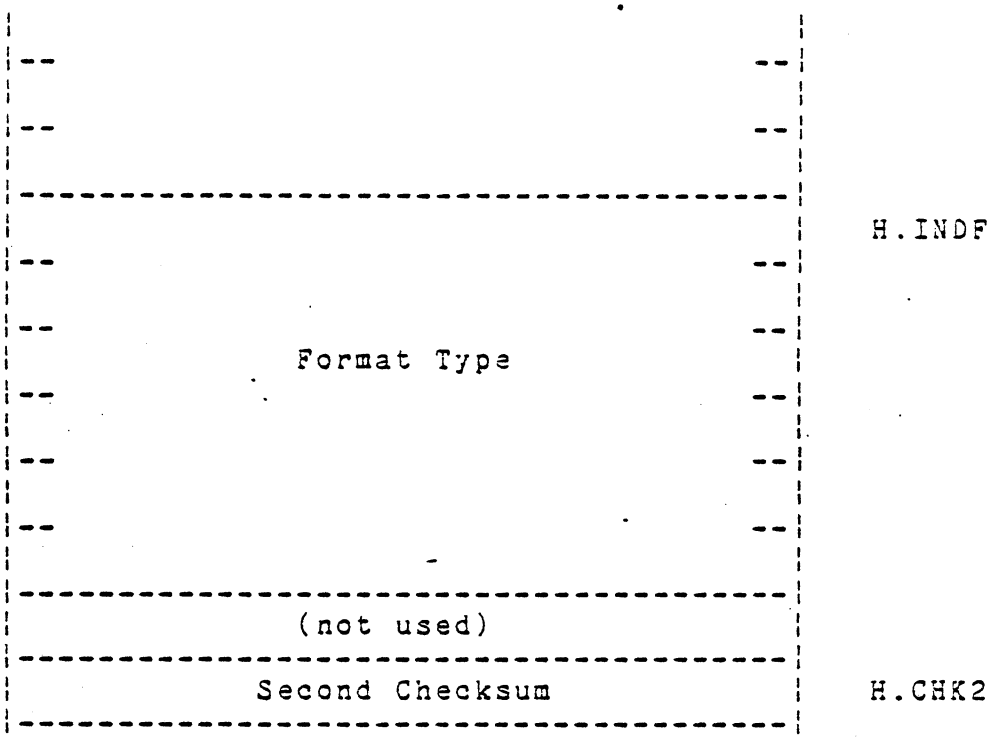
5.1.2.25 H.CHK2 2 Bytes Second Checksum

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 3.4.4.1.

5.1.2.A Home Block Layout




(not used)
First Checksum
H.CHK1
H.VDAT
Volume Creation Date
(not used)
H.INDN
Volume Name
H.INDO
Volume Owner



5.1.3 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 3 of the index file and continues through VBN 2+m, where m is the number of blocks in the bitmap. It is located at the logical block indicated by offset H.IBLB in the home block.

5.1.4 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 2+m+n (where m is the number of blocks in the index file bitmap).

5.2 Storage Bitmap File

The storage bitmap file is File ID 2,2,0. It is listed in the MFD as BITMAP.SYS;1. The storage bitmap is used to control the available space on a unit. It consists of a storage control block which contains summary information about the unit, and the bitmap itself which lists the availablilty of individual blocks.

5.2.1 Storage Control Block

Virtual block 1 of the storage bitmap is the storage control block. It contains summary information intended to optimize allocation of space on the unit. The following items are in the storage control block:

(3 bytes) Not used (zero)
(1 byte) Number of storage bitmap blocks
(2 bytes) Number of free blocks in 1st bitmap block
(2 bytes) Free block pointer in 1st bitmap block
.
.
.
(2 bytes) Number of free blocks in nth bitmap block
(2 bytes) Free block pointer in nth bitmap block
(4 bytes) Size of the unit in logical blocks

Note: Current implementations of Files-11 do not correctly initialize the word pairs containing number of free blocks and free block pointer for each bitmap block, nor are these values maintained as space is allocated and freed on the unit. They are therefore best looked upon as 2n garbage words and should not be used by future implementations of Files-11 until the disk structure is formally updated.

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 logical blocks on the unit rounded up to the next multiple

of 4096. 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 block j of the volume; if the bit is set, the block is free; if clear, the block is 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.

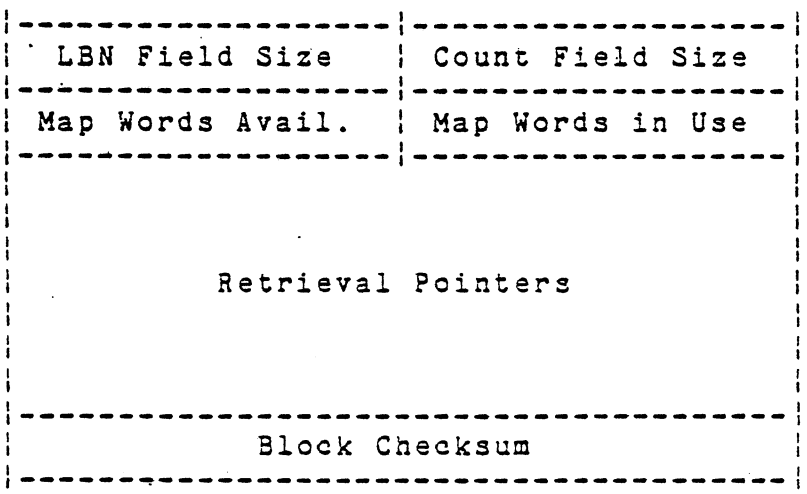
5.3 Bad Block File

The bad block file is File ID 3,3,0. 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.

5.3.1 Bad Block Descriptor

Virtual block 1 of the bad block file is 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 format of the bad block data is identical to the map area of a file header, except that the first four entries (M.ESQN, M.ERVN, M.EFNU, and M.EFSQ) are not present. The last word of the block contains the usual additive checksum. (See section 3.4.3 for a description of the map area.) This block is included in the bad block file to save the data it contains for future re-initialization of the volume.

Bad Block Descriptor Layout



5.4 Master File Directory

The master file directory is File ID 4,4,0. 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 five known files, plus whatever the user chooses to enter. In the two level UFD structure described in section 4.1.1, the MFD contains entries for all user file directories.

5.5 Core Image File

The core image file is File ID 5,5,0. 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.

6.0 FCS File Structure

File Control Services (FCS) is a user level interface to Files-11. 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 3.4.1.9). It uses only the first 7 words; the rest are ignored by FCS, but are reserved by DEC (see Section 7.1 RMS FILE ATTRIBUTES). 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.PRN Used to indicate that the two byte sequence number field for R.SEQ record format is to be interpreted as print control information (see Section 6.2.3.1 for format of print information).

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.

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 or sequenced 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.

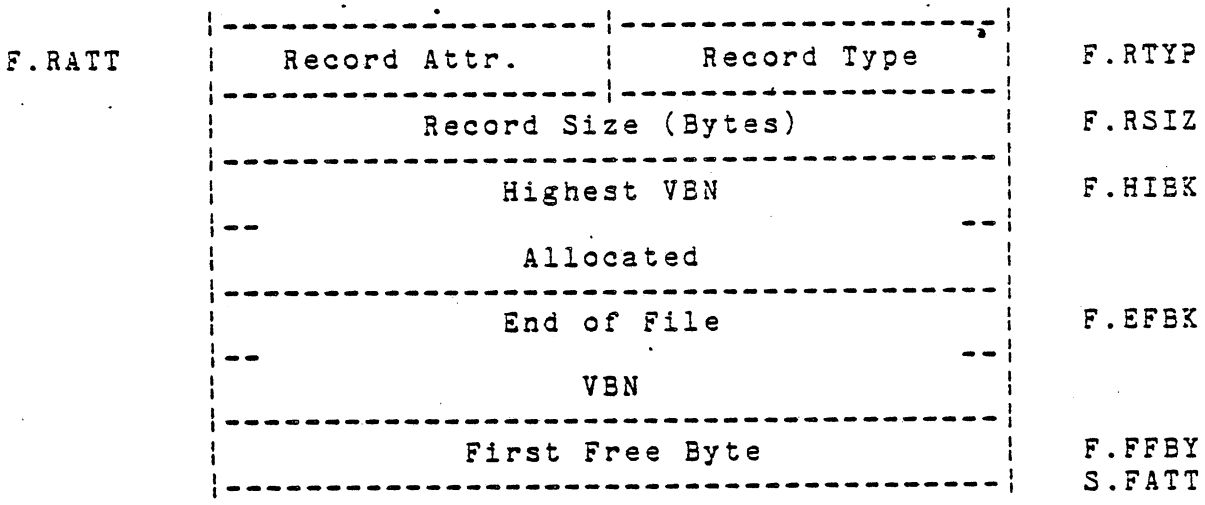
6.1.6 F.FFBY 2 Bytes First Free Byte

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.

6.1.A FCS File Attributes Layout



6.2 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.

6.2.1 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 byte. The content of the pad byte is undefined. 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/k (truncated)
        j = (n-1)/q+1 (truncated)
        i = ((n-1) mod q)*h
```

6.2.2 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 byte. The content of the pad byte is undefined.

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.

6.2.3 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.

6.2.3.1 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.

7.0.1 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.

7.0.1.1 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.

7.0.1.2 String Character Code Set

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

7.0.1.3 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.

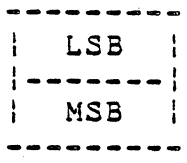
NOTE

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 = 0 and highest character = 255.

7.0.1.4 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.

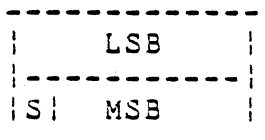
EXAMPLE: 2 byte unsigned binary value



7.0.1.5 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



7.0.1.6 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

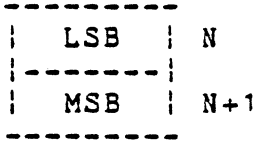
3 bytes start VBN 65,536 - 16,777,215

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

7.0.1.7 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.

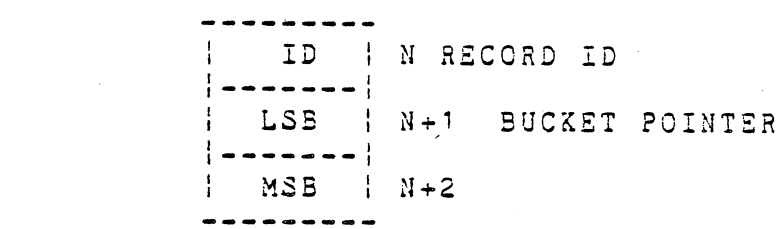
EXAMPLE: 2 byte bucket pointer



7.0.1.8 Record Pointers

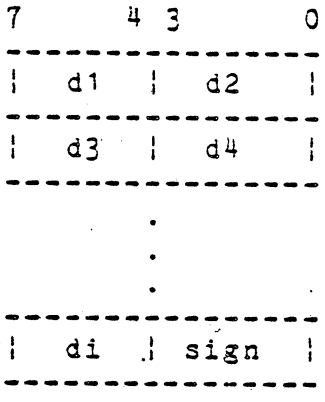
Record pointers are composed of two fields, a one byte record ID field followed by a bucket pointer. The ID is used as a unique record identifier for records within a bucket. The records are tagged with their ID'S when stored in the bucket.

EXAMPLE: 3 byte record pointer .



7.0.1.9 Packed Decimal Strings

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



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

7.1 RMS FILE ATTIRIBUTES

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.

7.1.1 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)

7.1.2 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.BLK for more detail.

7.1.3 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.

7.1.4 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.

7.1.5 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$LEOF respectively. These are the only two places that RMS stores block numbers in this manner (see Section 7.0.1), and is done so to maintain compatibility with FCS. The Relative and Index file does not use this field and its value is usually (but not guaranteed) either the contents of F$HVBN or the contents of F$HVBN plus one.

7.1.6 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 0 or 512. For more details on this field see Section 6.1.6 F.FFBY.

7.1.7 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 0 is interpreted as 1.

7.1.8 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 0 is interpreted as 2 so that compatibility with FCS'S Sequenced Variable length record format file (R.SEQ) is maintained.

7.1.9 F$MRS 2 Bytes Maximum Record Size

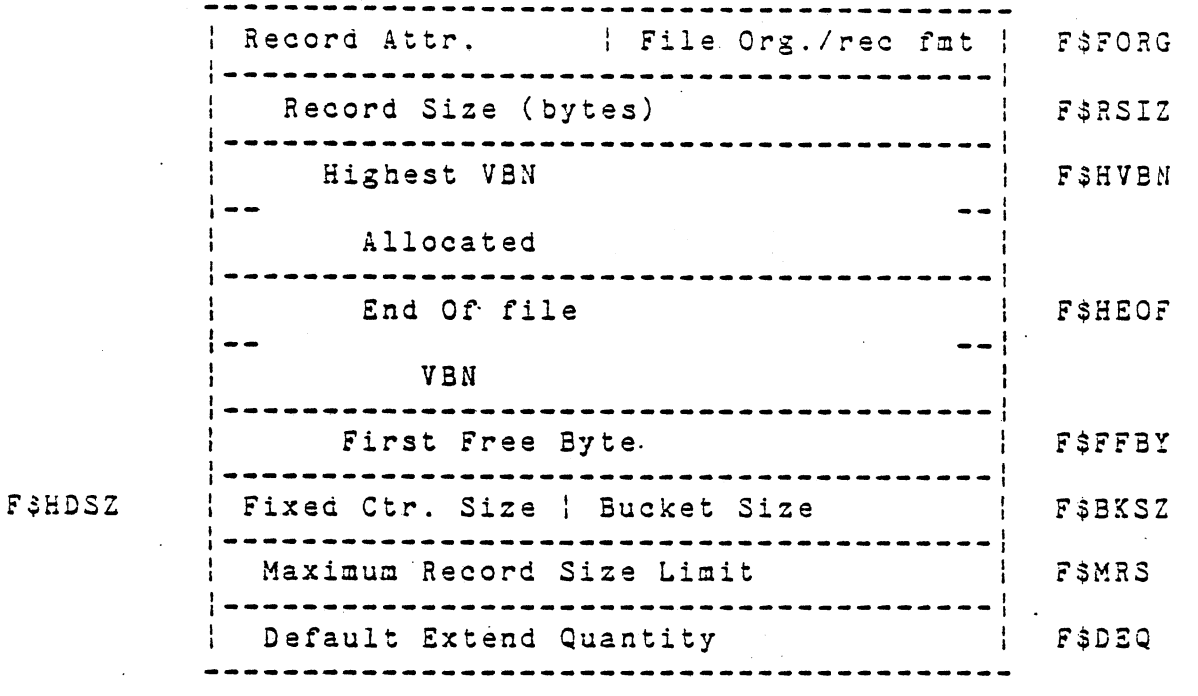
This field contains a user specified maximum record size limit in bytes, to be enforced on output operations. Files containing Fixed length format records have F$MRS set equal to F$RSIZ. For all other record formats F$MRS is set to the

user specified value given when the file was created. A value of 0 is interpreted as no maximum record size limit specified.

7.1.10 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.

7.1.A RMS File Attributed Layout



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

7.2 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).

7.2.1 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.

7.2.1.1 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).

7.2.1.2 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.

7.2.1.3 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.

7.2.1.4 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).

7.2.1.5 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.

7.2.1.6 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.

7.2.1.7 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.

7.2.1.8 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.

7.2.1.9 P$DBKS 1 Byte Data Bucket Size

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

7.2.1.10 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.

7.2.1.11 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).

7.2.1.12 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.

7.2.1.13 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)

7.2.1.14 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.

7.2.1.15 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.

7.2.1.16 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).

7.2.1.17 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.

7.2.1.18 K$MINL 2 Bytes Minimum Record Length

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

7.2.1.19 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.

7.2.1.20 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.

7.2.1.21 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.

7.2.1.22 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.

7.2.1.23 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.

7.2.1.24 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.

7.2.1.25 14 Spare Bytes

7.2.1.26 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.

7.2.1.27 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

7.2.1.28 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.

7.2.1.29 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).

7.2.1.30 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.

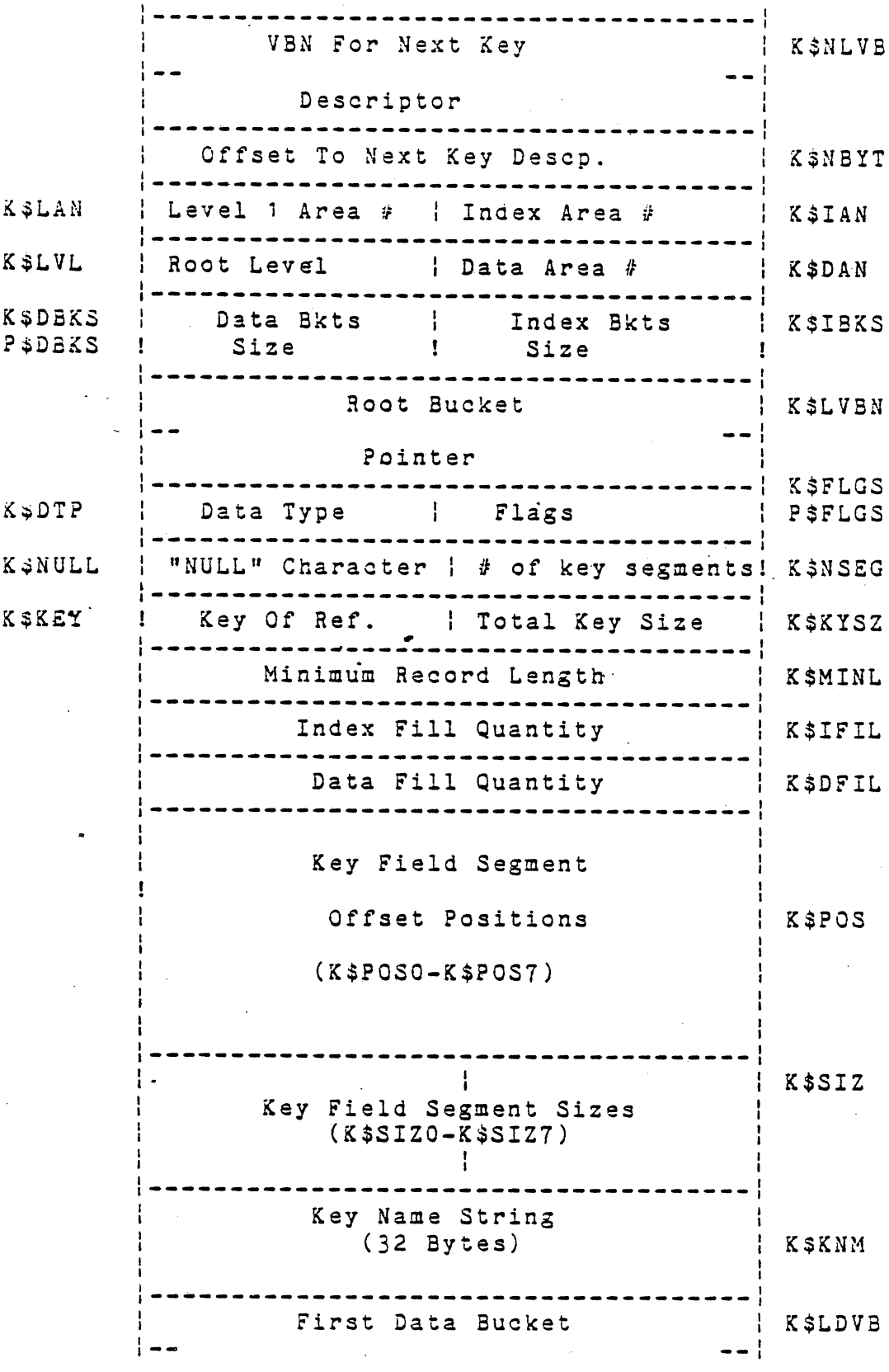
7.2.1.31 P$VERN 2 Bytes Prologue Version Number

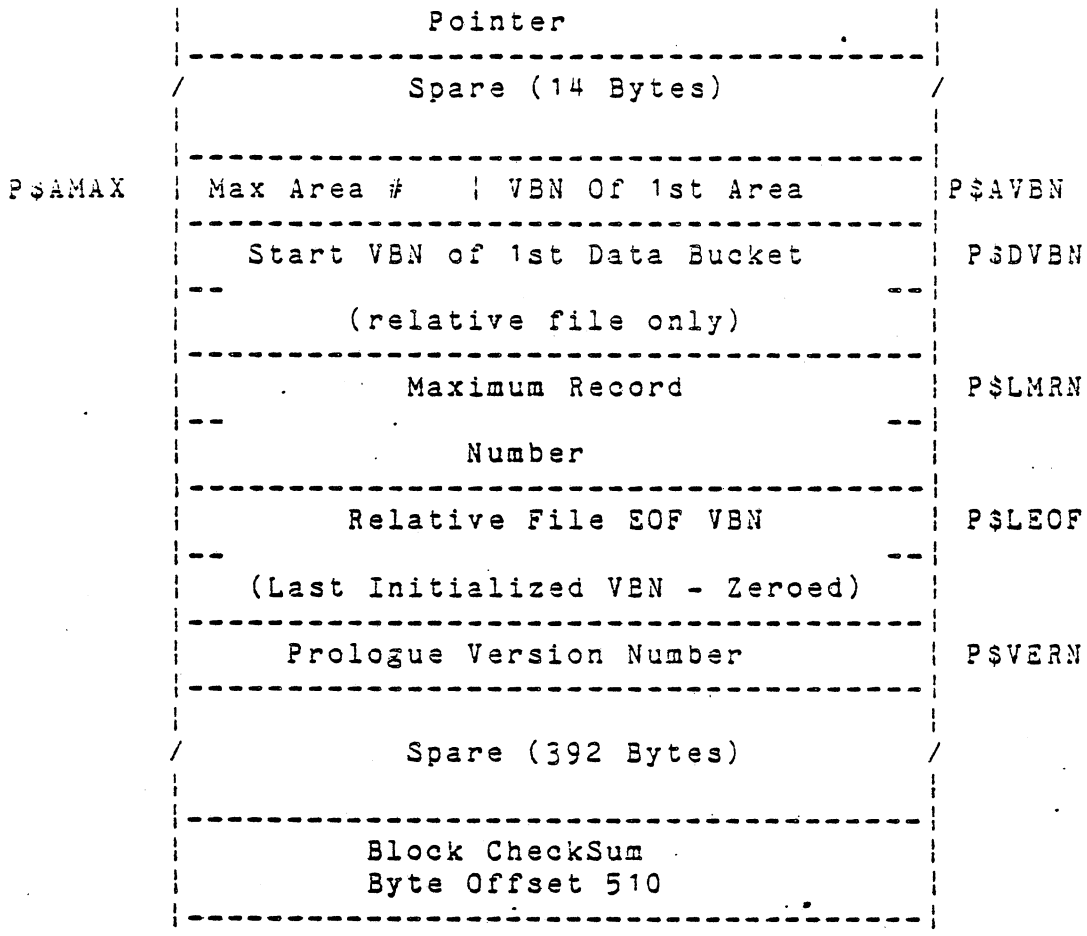
This field contains a prologue version nubmer. The only legal value at this time is one (1).

7.2.1.32 Reserved for Future Use 392 Bytes

7.2.1.33 Prologue Checksum 2 Bytes (see 7.2)

7.2.1.A Prologue Block 1 Layout





7.2.2 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.

7.2.3 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).

7.2.3.1 Spare 1 Byte

7.2.3.2 A$FLG 1 byte Flags (not used)

7.2.3.3 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.

7.2.3.4 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.

7.2.3.4 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.

7.2.3.5 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)

7.2.3.6 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.

7.2.3.7 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).

7.2.3.8 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.

7.2.3.9 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.

7.2.3.10 A$NUS 4 Bytes Number of blocks used

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

7.2.3.11 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.

7.2.3.12 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.

7.2.3.13 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.

7.2.3.14 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 0 means use the file's DEQ. However, in no case will less than one bucket size for this area be requested.

7.2.3.15 Reserved 2 Bytes

7.2.3.16 A$LOC 4 Bytes Start LBN on Volume

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

7.2.3.17 A$RFI 6 Bytes Related File ID

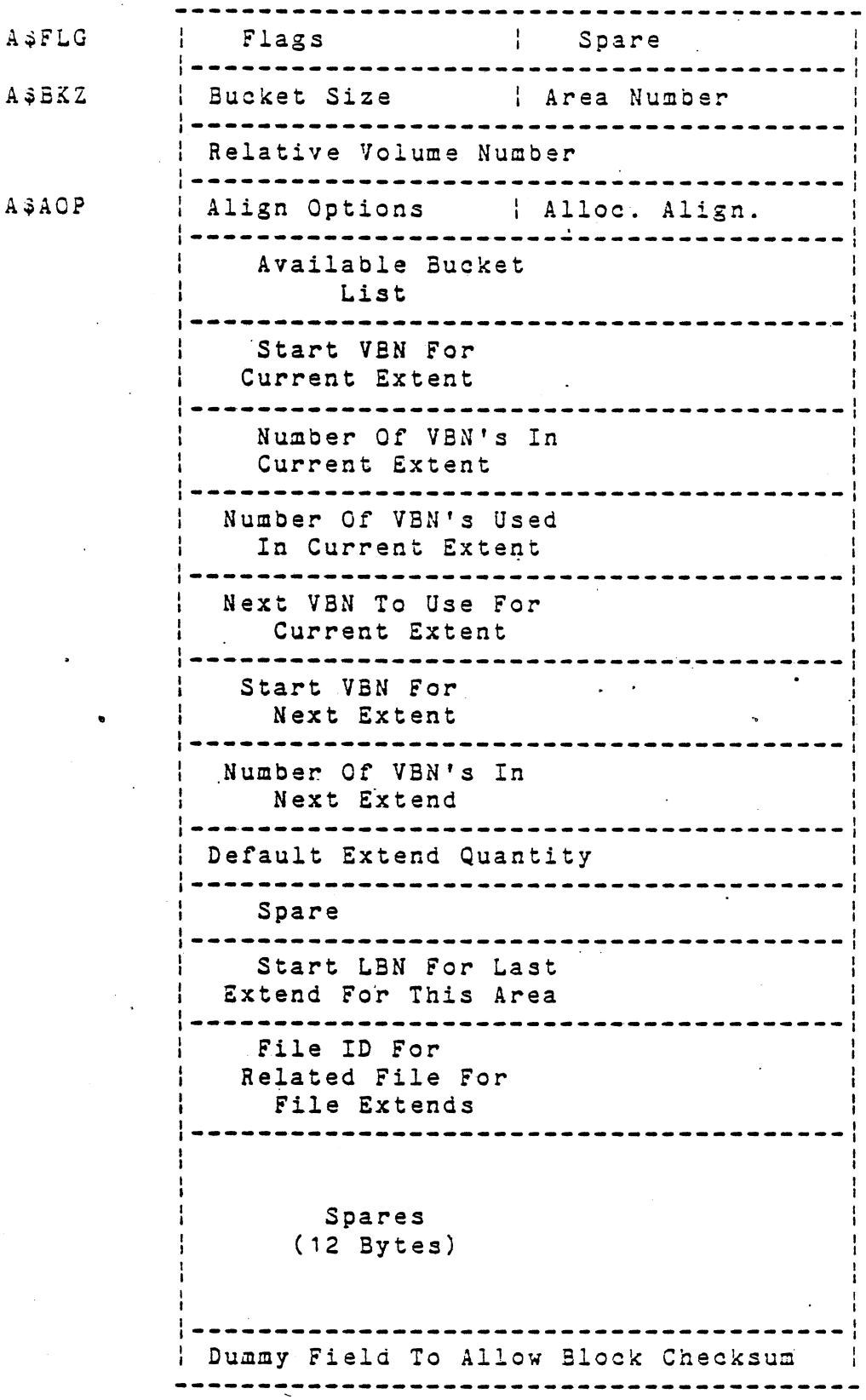
This field contain the FID of a related file for the XB$RFI allocation alignment (A$ALN) (not implemented)

7.2.3.18 Spares 12 Bytes

7.2.3.19 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.

7.2.3.A Area Descriptor Layout



7.3 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.

7.4 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.

7.4.1 Relative File Record Formats

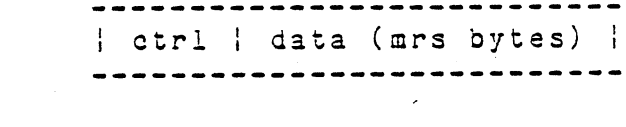
Records are stored in fixed length cells. The first byte of each cell is a record control byte used to provide deleted record control. The following bits are defined:

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

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

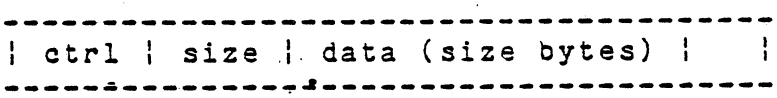
The relative file supports variable and variable with fixed control length record up to the required user specified Maximum Record Size (MRS). In these cases the record control byte is followed with a two byte binary count of the bytes in the record (the count does not include itself). If the cell size 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.

7.4.1A Fixed Length Records



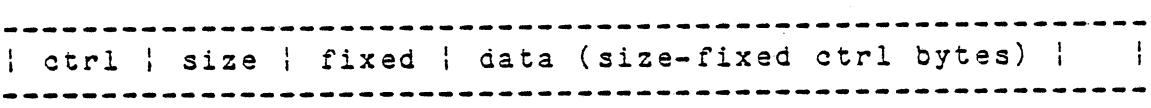
cell size = MRS+1

7.4.1.B Variable Length Records



cell size = MRS+3

7.4.1.C Variable With Fixed Control Records

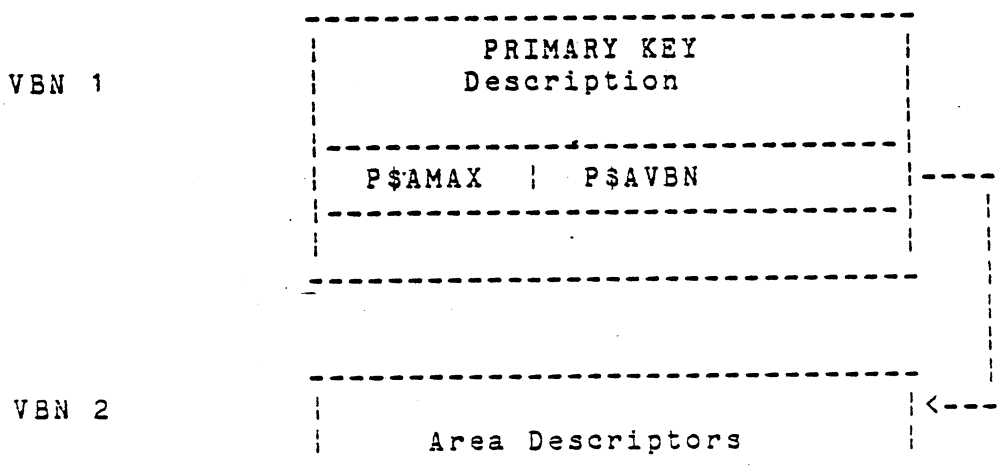


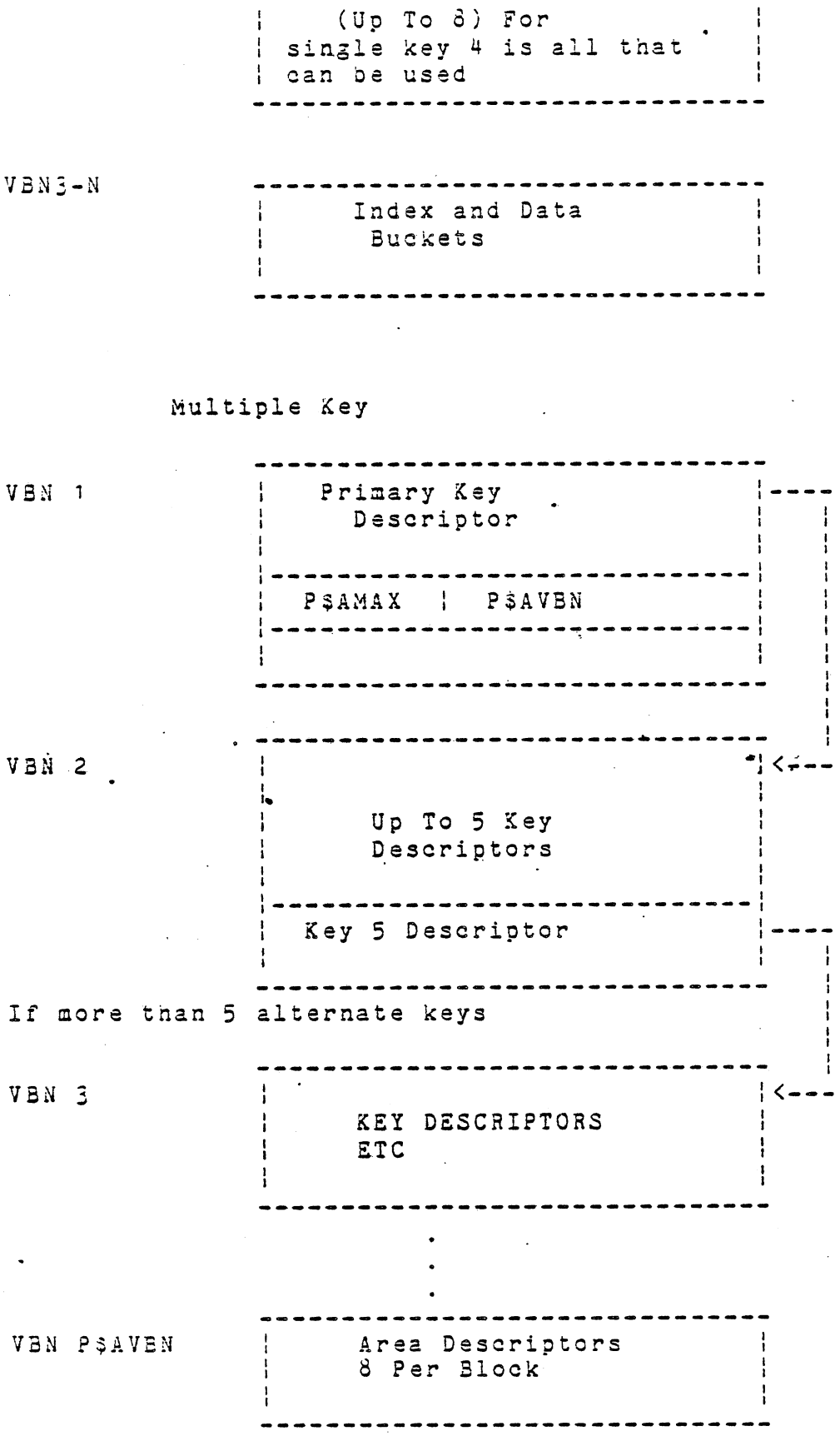
cell size = MRS+FIXED CTRL SIZE+3

7.5 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




(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 overhead 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.

7.5.0 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.

7.5.0.1 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 7.5.0.3 and 7.5.2.3 for details on RRV records.

7.5.0.2 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.

7.5.0.3 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.

7.5.1 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.

7.5.1.1 Header Area

Tne 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).

7.5.1.1.1 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.).

7.5.1.1.2 B$TAA 1 Byte This Allocation Area

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

7.5.1.1.3 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.

7.5.1.1.4 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.

7.5.1.1.5 B$NID 1 Byte Next Available ID

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

7.5.1.1.6 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.

7.5.1.1.7 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.

7.5.1.1.8 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.

7.5.1.1.9 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

7.5.1.2 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.

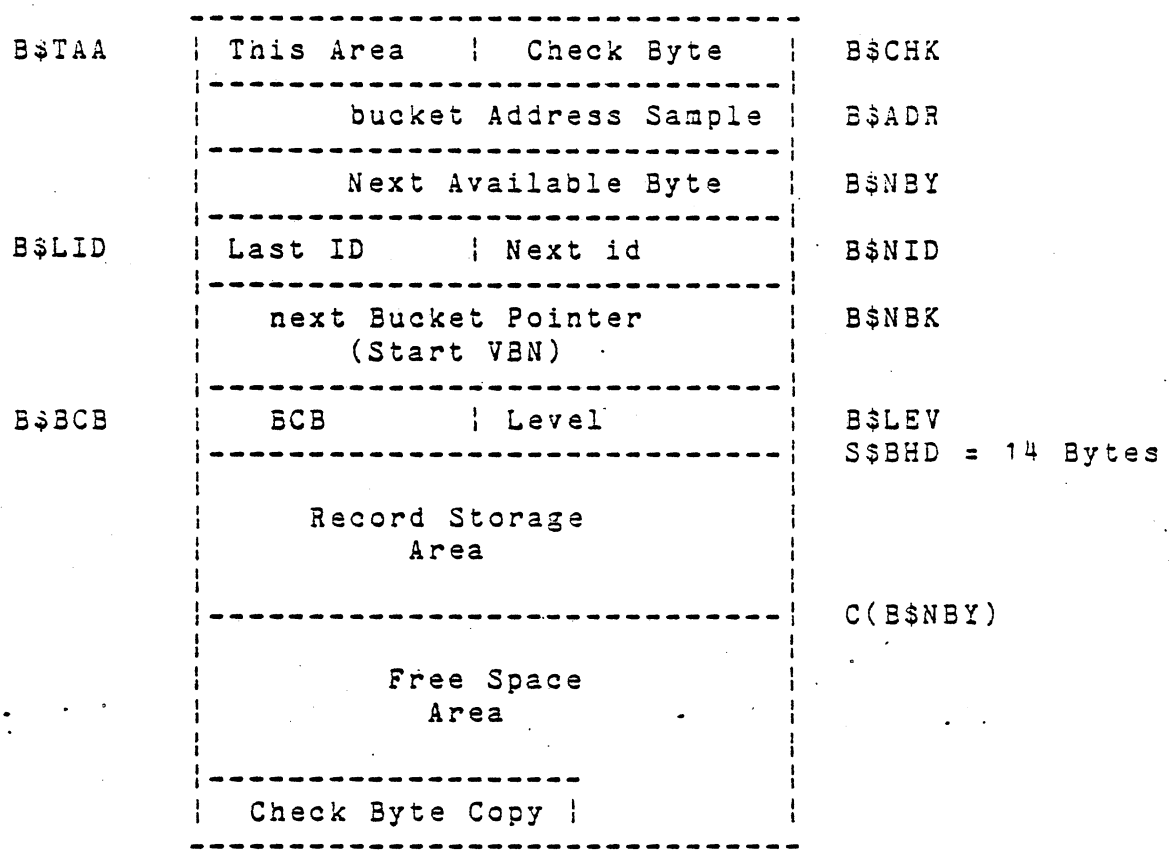
7.5.1.3 Free Storage Area

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.

7.5.1.4 S$BHD 14 Bytes Size of Header Area

This symbol represents the size of the bucket header area.

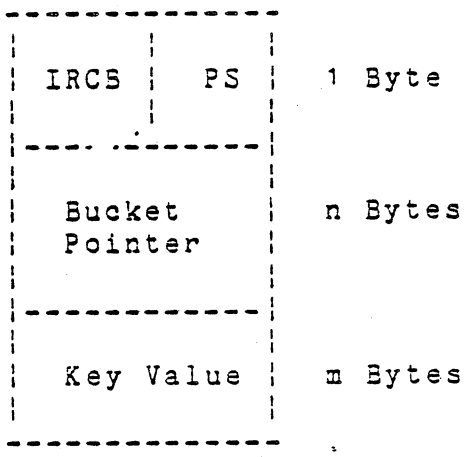
7.5.1.A Bucket Format Layout



7.5.2 Record Structures

The following record structures apply to the Indexed file organization.

7.5.2.1 Index bucket record



IRCB contains Index Record Control Bits

The following bits are defined in the IRCB byte:

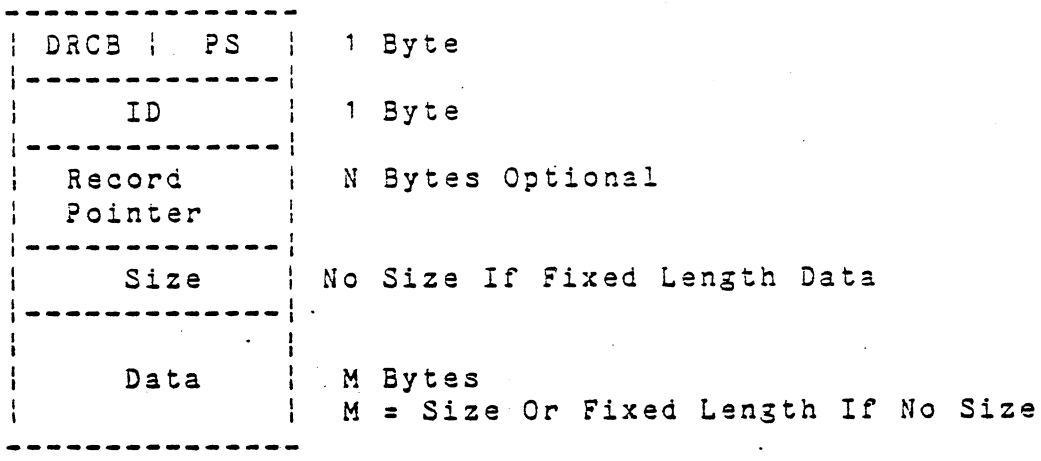
IC(IRCB) Compress key value (not currently defined).

EC(IRCB) Pointers are empty (not used).

PS is the pointer size as follows:

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

7.5.2.2 General Data Bucket Record



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.

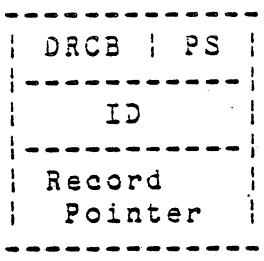
PS is the pointer size for the Record pointer as follows:

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

7.5.2.3 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:



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

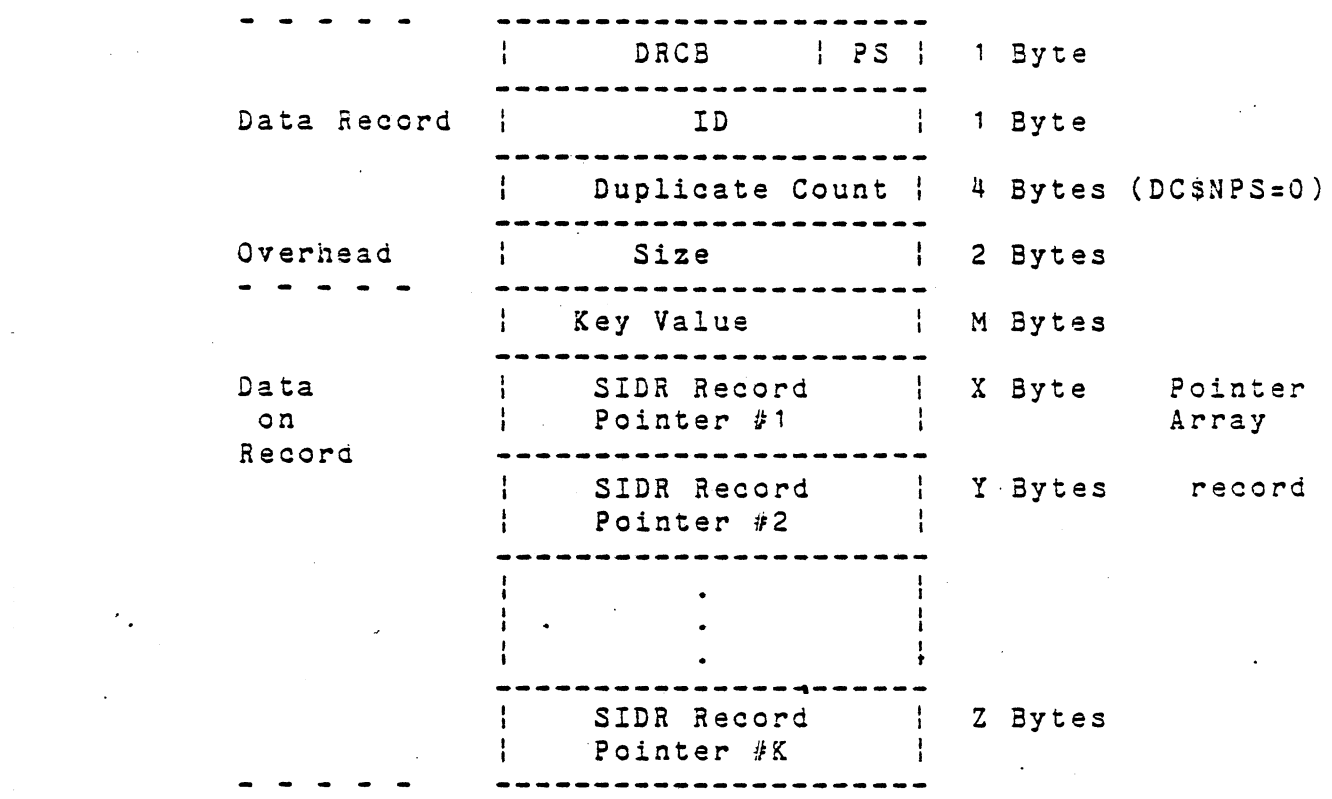
7.5.2.3.1 DELETED RRV RECORDS

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

7.5.2.4 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:



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.

7.5.2.5 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. 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.

7.5.2.6 SIDR Record Pointers

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

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

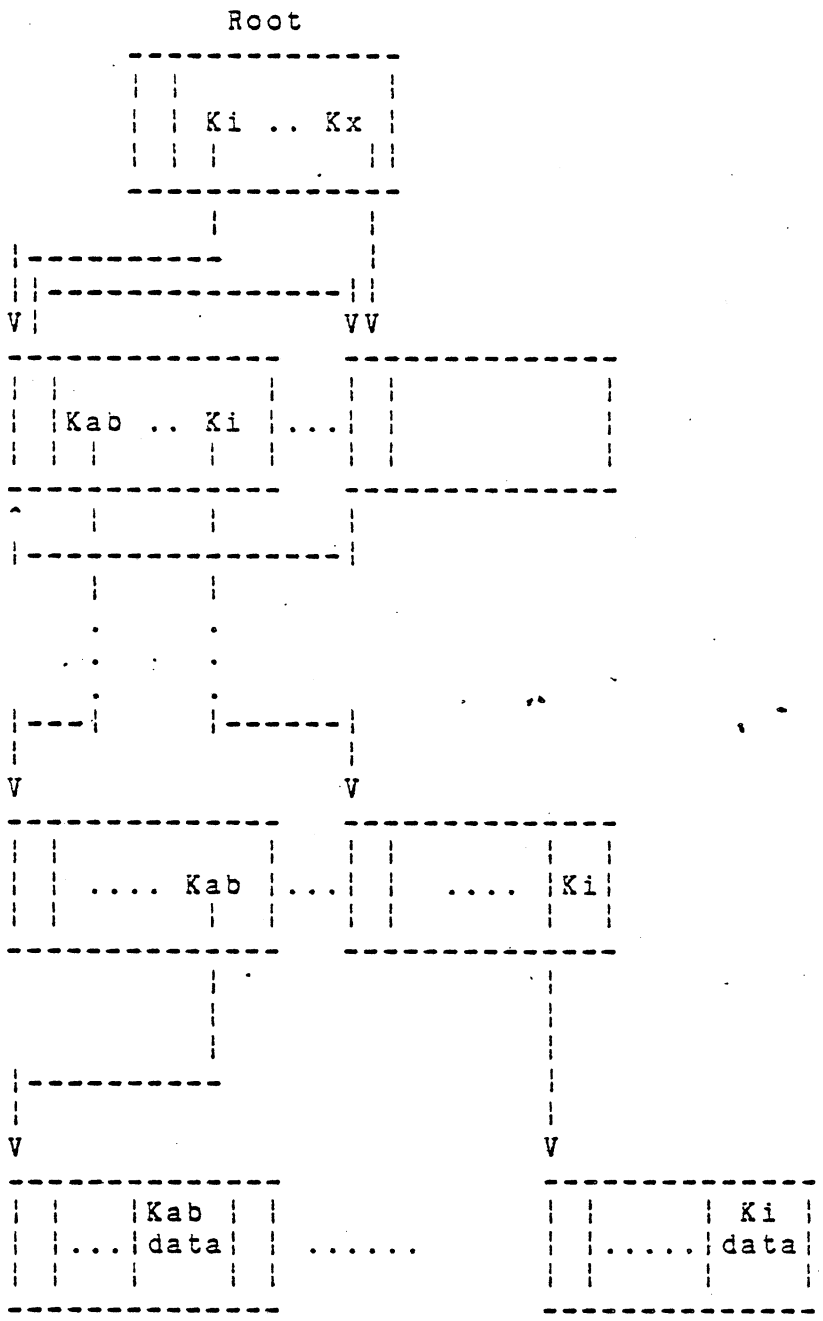
DRCB bits used for SIDR record pointers

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

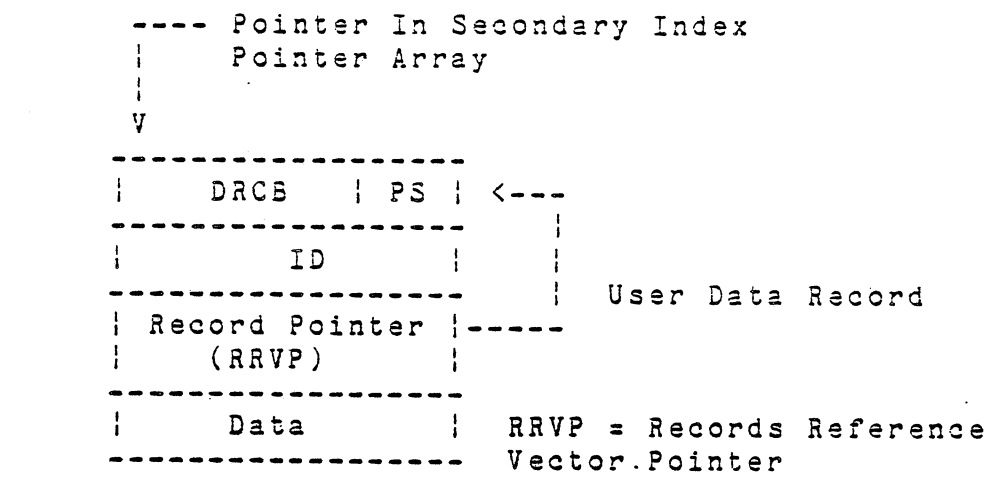
Index Structure



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



CASE 2: Record Has Moved

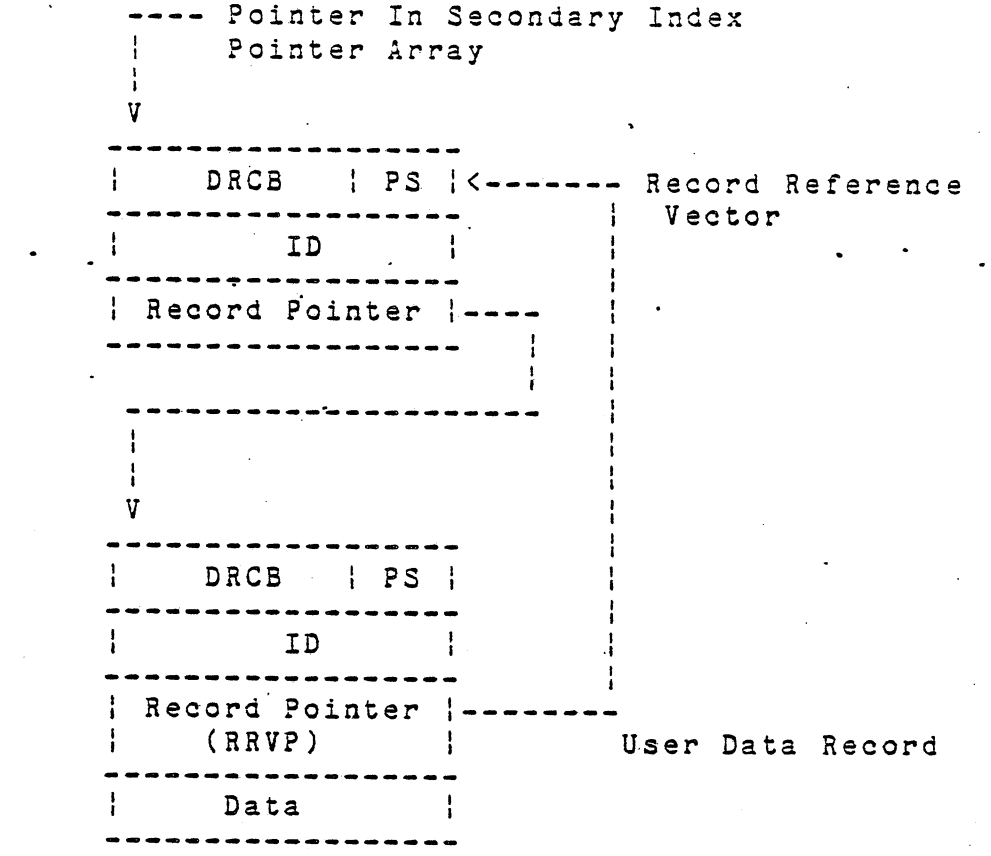


Figure 7-3

RRV Usage