

**iRMX 86™
DISK VERIFICATION UTILITY
REFERENCE MANUAL**

Order Number: 144133-001

REV.	REVISION HISTORY	PRINT DATE
-001	Original Issue	11/81

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BXP	Intel	Library Manager	Plug-A-Bugle
CREDIT	Intel	MCS	PROMPT
i	Intelevison	Megachassis	RMX/80
ICE	Intellec	Micromainframe	System 2000
iCS	iRMX	Micromap	
im	iSBC	Multibus	
Insite	iSBX	Multimodule	

PREFACE

This manual documents the Disk Verification Utility, a software tool that runs as a Human Interface command, verifying and modifying the data structures of iRMX 86 named and physical volumes. The manual describes the utility invocation and contains detailed descriptions of all utility commands. Also, because users must be familiar with the structure of iRMX 86 volumes in order to use the Disk Verification Utility intelligently, the manual contains an appendix that describes the structure of iRMX 86 named volumes.

READER LEVEL

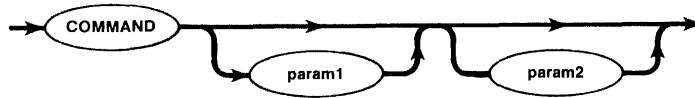
This manual is intended for system programmers who have had experience in examining actual volume information. It does not attempt to teach the user the proper procedures for examining and editing volume information.

NOTATIONAL CONVENTIONS

This manual uses the following conventions to illustrate syntax.

UPPERCASE	Uppercase information must be entered exactly as shown. You can, however, enter this information in uppercase or lowercase.
lowercase	Lowercase fields contain variable information. You must enter the appropriate value or symbol for variable fields.
<u>underscore</u>	In examples of dialog at the terminal, user input is underscored to distinguish it from system output.

Also, this manual uses the "railroad track" schematic to illustrate the syntax of the disk verification commands. This syntax consists of what looks like an aerial view of a model railroad setup, with syntactic elements scattered along the track. To interpret the command syntax, you start at the left side of the schematic, follow the track through all the syntactic elements you desire (sharp turns and backing up are not allowed), and exit at the right side of the schematic. The syntactic elements that you encounter, separated by spaces, comprise a valid command. For example, a command that consists of a command name and two optional parameters would have the following schematic representation:



You could enter this command in any of the following forms:

```

COMMAND
COMMAND param1
COMMAND param2
COMMAND param1 param2

```

The arrows indicate the possible flow through the tracks; they are omitted in the remainder of the manual.

RELATED PUBLICATIONS

The following manuals provide additional information that may be helpful to users of this manual.

<u>Manual</u>	<u>Number</u>
iRMX™ 86 Human Interface Reference Manual	9803202
iRMX™ 86 Nucleus Reference Manual	9803122
iRMX™ 86 Basic I/O System Reference Manual	9803123
iRMX™ 86 Loader Reference Manual	143318
iRMX™ 86 Configuration Guide	9803126
iRMX™ 86 Installation Guide	9803125

CONTENTS

	PAGE
CHAPTER 1	
INVOKING THE DISK VERIFICATION UTILITY	
Invocation.....	1-2
Output.....	1-4
Invocation Error Messages.....	1-4
CHAPTER 2	
DISKVERIFY COMMANDS	
Command Names.....	2-1
Parameters.....	2-1
Input Radices.....	2-2
Command Error Messages.....	2-2
Command Dictionary.....	2-3
ALLOCATE Command.....	2-5
DISK Command.....	2-7
DISPLAYBYTE Command.....	2-9
DISPLAYDIRECTORY Command.....	2-12
DISPLAYFNODE Command.....	2-14
DISPLAYWORD Command.....	2-18
EXIT Command.....	2-21
FREE Command.....	2-22
HELP Command.....	2-24
Miscellaneous Commands.....	2-25
ADD.....	2-25
ADDRESS.....	2-25
BLOCK.....	2-26
DEC.....	2-27
DIV.....	2-27
HEX.....	2-27
MOD.....	2-28
MUL.....	2-28
SUB.....	2-29
QUIT Command.....	2-30
READ Command.....	2-31
SAVE Command.....	2-32
SUBSTITUTEBYTE Command.....	2-34
SUBSTITUTEWORD Command.....	2-37
VERIFY Command.....	2-40
WRITE Command.....	2-48
APPENDIX A	
STRUCTURE OF iRMX 86 NAMED VOLUMES	
Introduction.....	A-1
Volume Labels.....	A-2
ISO Volume Label.....	A-2
iRMX 86 Volume Label.....	A-4
Initial Files.....	A-8
Fnode File.....	A-8

CONTENTS (continued)

	PAGE
APPENDIX A (continued)	
Fnode 0 (Fnode File).....	A-14
Fnode 1 (Volume Free Space Map File).....	A-14
Fnode 2 (Free Fnodes Map File).....	A-15
Fnode 4 (Bad Blocks File).....	A-15
Root Directory.....	A-16
Other Fnodes.....	A-16
Long and Short Files.....	A-16
Short Files.....	A-17
Long Files.....	A-18
Flexible Diskette Formats.....	A-20
Example Volume.....	A-22
ISO Volume Label.....	A-22
iRMX 86 Volume Label.....	A-23
Fnode File.....	A-24
Fnode 0 (Fnode File).....	A-24
Fnode 1 (Free Space Map).....	A-25
Fnode 2 (Free Fnode Map).....	A-26
Fnode 3 (Accounting File).....	A-27
Fnode 4 (Bad Blocks File).....	A-28
Fnode 5 (Root Directory).....	A-29
Fnode 6 (Example File).....	A-31
Free Space Map File.....	A-32
Free Fnodes Map File.....	A-33
Root Directory.....	A-33

FIGURES

2-1.	DISPLAYBYTE Format.....	2-10
2-2.	DISPLAYWORD Format.....	2-19
2-3.	NAMED1 Verification Output.....	2-41
2-4.	NAMED2 Verification Output.....	2-42
2-5.	PHYSICAL Verification Output.....	2-43
A-1.	General Structure of Named Volumes.....	A-1
A-2.	Short File Fnode.....	A-17
A-3.	Long File Fnode.....	A-19

TABLES

A-1.	Eight-Inch Diskette Characteristics.....	A-20
A-2.	5 1/4-Inch Diskette Characteristics.....	A-21

CHAPTER 1. INVOKING THE DISK VERIFICATION UTILITY

In the process of using an iRMX 86 application system, you may have occasion to store data on secondary storage devices, sometimes large amounts of data. Due to the nature of secondary storage devices, unforeseen circumstances such as power irregularities may destroy information on these devices, causing them to be inaccessible to your iRMX 86 system. In some cases, the loss of only a small amount of data can render an entire volume, such as a disk, useless.

In such cases, it is desirable to have a mechanism to examine and modify the damaged volume. This mechanism would allow you to determine how much of the information on the volume was damaged. It would also allow you to recreate file structures on the damaged volume so that you could salvage some of the valid data. The iRMX 86 disk verification utility is a tool that allows you to perform these functions.

The disk verification utility verifies the data structures of iRMX 86 physical and named volumes. It can also be used to reconstruct the free fnodes map and the volume free space map of the volume and perform absolute editing.

You can use the disk verification utility in one of two ways:

- As a single command which verifies the structures of a volume and returns control to the Human Interface.
- As an interactive program which allows you to check and modify information on the volume by entering individual disk verification commands.

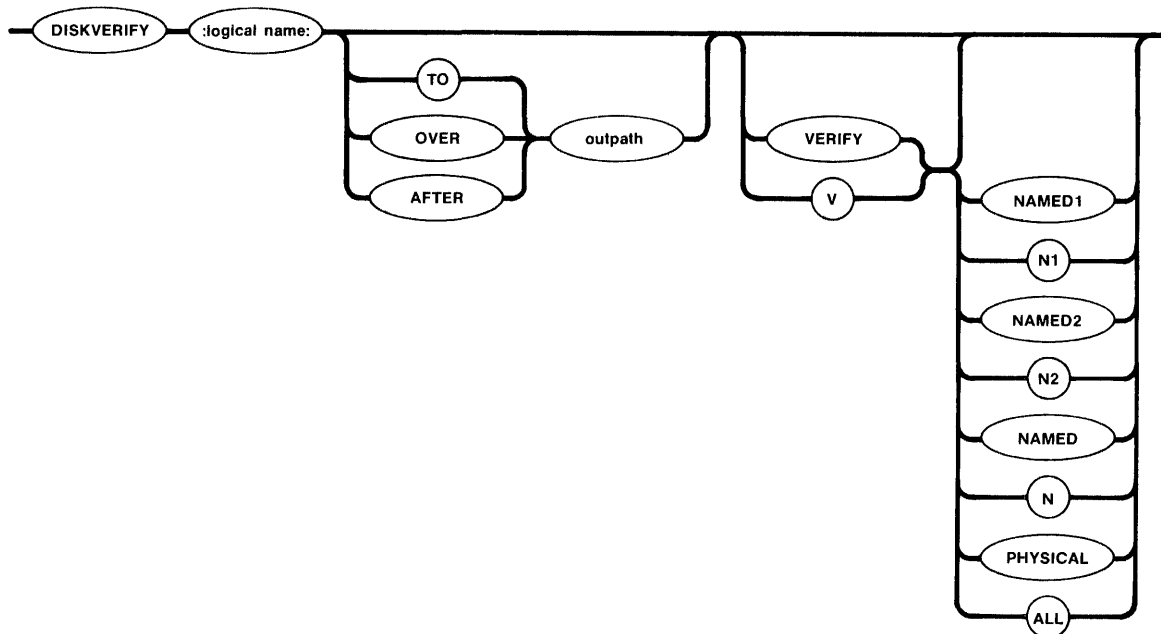
To take full advantage of the capabilities of the disk verification utility, you must be familiar with the structure of iRMX 86 named volumes. Appendix A contains detailed information about the volume structure. If you are unfamiliar with the iRMX 86 volume structure, you should avoid using the individual disk verification commands. When used carelessly, these commands can make your volumes unusable.

However, even if you know nothing about iRMX 86 volume structures, you can still use the utility as a single command to verify that the data structures on an iRMX 86 volume are valid.

INVOKING THE DISK VERIFICATION UTILITY

INVOCATION

The format of the Human Interface command used to invoke the disk verification utility is as follows:



where:

- :logical-name:** Logical name of the secondary storage device containing the volume.
- TO** Copies the output from the disk verification utility to the specified file. If no preposition is specified, TO :CO: is the default.
- OVER** Copies the output from the disk verification utility over the specified file.
- AFTER** Appends the output from the disk verification utility to the end of the specified file.
- outpath** Pathname of the file to receive the output from the disk verification utility. If you omit this parameter and the TO/OVER/AFTER preposition, the utility copies the output to the console screen (TO :CO:). You cannot direct the output to a file on the volume being verified. If you attempt this, the utility returns an E\$NOT_CONNECTION error message.

INVOKING THE DISK VERIFICATION UTILITY

- VERIFY or V** Performs a verification of the volume. This verification function and the associated options are described in detail in the "VERIFY Command" section of Chapter 2. If you specify this parameter and omit the options, the utility performs the NAMED verification.
- If you specify this parameter, the utility performs the verification function and returns control to you at the Human Interface level. You can then enter any Human Interface command.
- If you omit this parameter, the utility displays a header message and the utility prompt (*). You can then enter any of the disk verification commands listed in Chapter 2.
- NAMED1 or N1** VERIFY option that applies to named volumes only. This option checks the fnodes of the volume to ensure that they match the directories in terms of file type and file heirarchy. This option also checks the information in each fnode to ensure that it is consistent. Refer to the description of the VERIFY command in Chapter 2 for more information.
- NAMED2 or N2** VERIFY option that applies to named volumes only. This option checks the allocation of fnodes on the volume, checks the allocation of space on the volume, and verifies that the fnodes point to the correct locations on the volume. Refer to the description of the VERIFY command in Chapter 2 for more information.
- NAMED or N** VERIFY option that performs both the NAMED1 and NAMED2 verification functions on a named volume. If you omit the VERIFY option, NAMED is the default option.
- PHYSICAL** VERIFY option that applies to both named and physical volumes. This option reads all blocks on the volume and checks for I/O errors.
- ALL** VERIFY option that applies to both named and physical volumes. For named volumes, this option performs both the NAMED and PHYSICAL verification functions. For physical volumes, this option performs the PHYSICAL verification function.

INVOKING THE DISK VERIFICATION UTILITY

OUTPUT

When you enter the DISKVERIFY command, the utility responds by displaying the following line:

```
iRMX 86 DISK VERIFY UTILITY, Vx.x
```

where Vx.x is the version number of the utility. If you specify the VERIFY or V parameter in the DISKVERIFY command, the utility performs a verification of the volume and copies the verification information to the console (or to the file specified by the outpath parameter). The verification information is the same as that produced by the VERIFY utility command. Refer to the description of the VERIFY command in Chapter 2 for a description of the verification output. After generating the verification output, the utility returns control to the Human Interface, which prompts you for more Human Interface commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1: VERIFY NAMED2
iRMX 86 DISK VERIFY UTILITY , Vx.x
DEVICE NAME = F1          : DEVICE SIZE = 0003E900 : BLOCK SIZE = 0080

'NAMED2' VERIFICATION

      BIT MAPS O.K.
```

-

However, if you omit the VERIFY (or V) parameter from the DISKVERIFY command, the utility does not return control to the Human Interface. Instead, it issues an asterisk (*) as a prompt and waits for you to enter individual DISKVERIFY commands. The following is an example of such a DISKVERIFY command:

```
-DISKVERIFY :F1:
iRMX 86 DISK VERIFY UTILITY , Vx.x
*
```

After you receive the asterisk prompt, you can enter any of the DISKVERIFY commands listed in the next section. If you enter anything else, the utility will display an error message.

INVOCATION ERROR MESSAGES

```
logical name, 0045 : E$LOG_NAME_NEXIST
```

You specified a nonexistent logical name in either the :logical name: parameter or the outpath parameter.

```
8042 : E$NOT_CONNECTION
```

You attempted to direct output to a file on the volume being verified.

INVOKING THE DISK VERIFICATION UTILITY

command line error

You made a syntax error when entering the command.

device size inconsistent

size in volume label = value1 : computed size = value2

When the disk verification utility computed the size of the volume, the size it computed did not match the information recorded in the iRMX 86 volume label. It is likely that the volume label contains invalid or corrupted information. This error is not a fatal error, but it is an indication that further error conditions may result during the verification session. You may have to reformat the volume or use the disk verification utility to modify the volume label.

logical name, illegal logical name

The logical name you specified was not surrounded by colons (:).

not a named disk

You tried to perform a NAMED, NAMED1, or NAMED2 verification on a physical volume.

verify-function argument error

The VERIFY option you specified is not valid.

CHAPTER 2. DISKVERIFY COMMANDS

When the disk verification utility issues the asterisk prompt, you can enter individual DISKVERIFY commands to examine or change the information on the volume. This process usually involves reading a portion of the volume into a buffer, modifying that buffer, and writing the information back to the volume. This chapter describes the commands that allow you to perform these operations.

The commands in this chapter are presented in alphabetical order, without regard to function. Before describing the individual commands, this chapter discusses command names, parameters, input radices, and error messages. It also provides a command dictionary.

COMMAND NAMES

When you enter a DISKVERIFY command, you can enter the command name or command name abbreviation as listed in this chapter, or you can enter any portion of the command name that uniquely identifies the command from all other DISKVERIFY commands.

For example, when specifying the DISPLAYFNODE command, you can enter the command name as:

```
DISPLAYFNODE
DF
DISPLAYF
```

or any other partial form of the word DISPLAYFNODE that contains at least the characters DISPLAYF.

PARAMETERS

Several DISKVERIFY commands have parameters which this chapter describes as being in the form:

```
keyword = value
```

Even though the individual command descriptions do not mention this, you can also enter these parameters in the form:

```
keyword (value)
```

For example, the following are two acceptable ways of specifying a FREE command:

```
FREE FNODE = 10
```

```
FREE FNODE (10)
```

INPUT RADICES

DISKVERIFY always produces numerical output in hexadecimal format. However, when you provide input to DISKVERIFY, you can specify the radix of numerical quantities by including a radix character immediately after the number. The valid radix characters include:

<u>radix</u>	<u>character</u>	<u>example</u>
hexadecimal	h or H	16h, 7CH
decimal	t or T	23t, 100T
octal	o, O, q, or Q	27o, 33Q

If you omit the radix character, DISKVERIFY assumes the number is hexadecimal.

COMMAND ERROR MESSAGES

Each DISKVERIFY command can generate a number of error messages which indicate errors in the way you specified the command or problems with the volume itself. The messages for each command are listed with the command itself. However, the following messages can also occur with many of the commands (generally, whenever the utility reads from or writes to the volume):

seek error

The utility unsuccessfully attempted to seek to a location on the volume. This error normally results from invalid information in the IRMX 86 volume label or in the fnodes.

block I/O error

The utility attempted to read or write a block on the volume and found that the block was physically flawed. Thus it cannot complete the requested command.

Also, when the disk verification utility begins processing, it obtains some information from the iRMX 86 volume label. If the label contains invalid information, the utility, in some cases, can assume that a named volume is a physical volume. If this occurs, the commands that apply to named volumes only (such as DISPLAYFNODE, DISPLAYDIRECTORY, and VERIFY NAMED) will issue the following message:

not a named disk

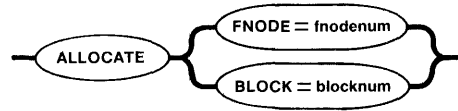
If you are convinced that your volume is indeed a named volume, this message may indicate that the iRMX 86 volume label is corrupted.

COMMAND DICTIONARY

Command	Synopsis	Page
READ	Reads a volume block into the working buffer	2-31
DISPLAYBYTE	Displays the working buffer in byte format	2-9
DISPLAYWORD	Displays the working buffer in word format	2-18
SUBSTITUTEBYTE	Modifies the contents of the working buffer in byte format	2-34
SUBSTITUTEWORD	Modifies the contents of the working buffer in word format	2-37
WRITE	Writes the working buffer to the volume	2-48
DISK	Lists the attributes of the volume	2-7
DISPLAYFNODE	Displays fnode information	2-14
DISPLAYDIRECTORY	Displays directory contents	2-12
ALLOCATE	Marks a particular fnode or volume block as allocated	2-5
FREE	Marks a particular fnode or volume block as free	2-22
SAVE	Writes the updated fnode and free space maps to the volume	2-32
VERIFY	Verifies the volume	2-40
HELP	Lists the DISKVERIFY commands	2-24
miscellaneous commands	Perform useful arithmetic and conversion functions; the commands include ADD, SUB, MUL, DIV, MOD, HEX, DEC, ADDRESS, and BLOCK.	2-25
EXIT	Exits the disk verification utility.	2-21
QUIT	Exits the disk verification utility.	2-30

ALLOCATE COMMAND

This command designates file descriptor nodes (fnodes) and volume blocks as allocated. The format of the ALLOCATE command is as follows:



INPUT PARAMETERS

- fnum** Number of the fnode to allocate. This number can range from 0 through (max fnodes - 1), where max fnodes is the number of fnodes defined when the volume was originally formatted.
- blocknum** Number of the volume block to allocate. This number can range from 0 through (max blocks - 1), where max blocks is the number of volume blocks in the volume.

OUTPUT

If you are using ALLOCATE to allocate fnodes, ALLOCATE displays the following message:

fnum, fnode marked allocated

where fnum is the number of the fnode that the utility designated as allocated.

If you are using ALLOCATE to allocate volume blocks, ALLOCATE displays the following message:

blocknum, block marked allocated

where blocknum is the number of the volume block that the utility designated as allocated.

ALLOCATE does not check the allocation status of fnodes or blocks before allocating them. Therefore, if you specify ALLOCATE for a block or fnode that is already allocated, ALLOCATE returns the same message as it would for a previously unallocated block or fnode.

ALLOCATE

DESCRIPTION

Fnodes are data structures on the volume that describe the files on the volume. They are created when the volume is formatted. An allocated fnode is one that represents an actual file. ALLOCATE designates fnodes as allocated by updating the FLAGS field of the fnode and free fnodes map file with this information.

An allocated volume block is a block of data storage that is part of a file; it is not available to be assigned to a new file. ALLOCATE designates volume blocks as allocated by updating the volume free space map with this information.

ERROR MESSAGES

argument error

You made a syntax error in the command or specified a nonnumeric character in the blocknum or fnum parameter.

blocknum, block out of range

The block number that you specified was larger than the largest block number in the volume.

fnum, fnode out of range

The fnode number that you specified was larger than the largest fnode number in the volume.

DISK COMMAND

This command displays the attributes of the volume being verified. The format of this command is as follows:



OUTPUT

The output of the DISK command depends on whether the volume is formatted as a physical or named volume. For a physical volume, the DISK command displays the following information:

```
Device name = devname
  Physical disk
    Device gran = devgran
    Block size = devgran
    No of blocks = numblocks
    Volume size = size
```

where:

devname	Name of the device containing the volume. This is the physical name of the device, as specified in the ATTACHDEVICE Human Interface command.
devgran	Granularity of the device, as defined in the device unit information block (DUIB) for the device. Refer to the iRMX 86 CONFIGURATION GUIDE for more information about DUIBs. For physical devices, this is also the volume block size.
numblocks	Number of volume blocks in the volume.
size	Size of the volume, in bytes.

For a named volume, the DISK command displays the following information:

```
Device name = devname
  Named disk, Volume name = volname
    Device gran = devgran
    Block size = volgran
    No of blocks = numblocks
    Volume size = size
    No of fnodes = numfnodes
```

DISK

OUTPUT (continued)

The devname, devgran, numblocks, and size fields are the same as for physical files. The remaining fields are as follows:

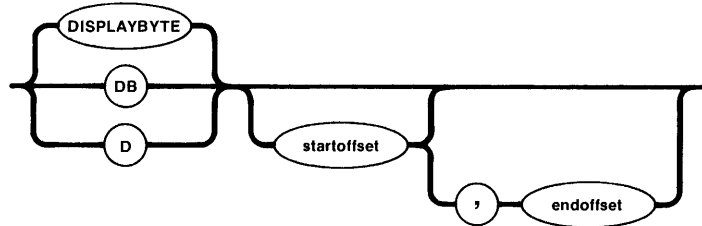
volname	Name of the volume, as specified when the volume was formatted. Refer to Appendix A or to the description of the FORMAT command in the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for more information.
volgran	Volume granularity, as specified when the volume was formatted. Refer to Appendix A or to the description of the FORMAT command in the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for more information.
numfnodes	Number of fnodes in the volume. The fnodes were created when the volume was formatted. Refer to Appendix A or to the description of the FORMAT command in the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for more information.

DESCRIPTION

The DISK command displays the attributes of the volume. The format of the output from DISK depends on whether the volume is formatted as a named or physical volume.

DISPLAYBYTE COMMAND

This command displays the specified portion of the working buffer in byte format. It displays the buffer in 16-byte rows. The format of this command is as follows:



INPUT PARAMETERS

- startoffset Number of the byte, relative to the start of the buffer, which begins the display. DISPLAYBYTE starts the display with the row containing the specified offset. If you omit this parameter, DISPLAYBYTE starts displaying from the beginning of the working buffer.
- endoffset Number of the byte, relative to the start of the buffer, which ends the display. If you omit this parameter, DISPLAYBYTE displays only the row indicated by startoffset. However, if you omit both startoffset and endoffset, DISPLAYBYTE displays the entire working buffer.

OUTPUT

In response to the command, DISPLAYBYTE displays the specified portion of the working buffer in rows, with 16 bytes displayed in each row. Figure 2-1 illustrates the format of the display.

As Figure 2-1 shows, DISPLAYBYTE begins by listing the block number of the data being displayed. It then lists the specified portion of the buffer, providing the column numbers as a header and beginning each row with the relative address of the first byte in the row. It also includes, at the right of the listing, the ASCII equivalents of the bytes, if the ASCII equivalents are printable characters. (If a byte is not a printable character, DISPLAYBYTE displays a period in the corresponding position.)

DISPLAYBYTE

OUTPUT (continued)

BLOCK NUMBER = blocknum

offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII STRING
0000	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
.					.						.						.
.					.						.						.
.					.						.						.

Figure 2-1. DISPLAYBYTE Format

DESCRIPTION

After you read a volume block of memory into the working buffer with the READ command, you can display part or all of that memory, in byte format, by entering the DISPLAYBYTE command. DISPLAYBYTE displays the hexadecimal value for each byte in the specified portion of the buffer.

If you omit all parameters, DISPLAYBYTE displays the entire block stored in the working buffer.

ERROR MESSAGES

argument error

You made a syntax error in the command or specified a nonnumeric character in one of the offset parameters.

invalid offset

You either specified a larger value for startoffset than for endoffset or you specified an offset value that was larger than the number of bytes in the block.

EXAMPLES

Assuming that you have read block 20h into the working buffer with the READ command, the following command displays that block.

*DISPLAYBYTE

BLOCK NUMBER = 20

offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII STRING
0000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0010	00	00	08	00	00	00	00	00	00	00	01	00	0F	FF	FF	00
0020	00	00	00	00	00	05	00	00	00	00	25	00	08	01	FF	FF%
0030	25	1F	00	00	2E	00	00	00	25	1F	00	00	2B	00	00	00	%.....%...+...
0040	01	00	00	00	01	00	80	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00
0070	00	00	00	00	01	00	0F	FF	FF	00	00	00	00	00	00	05

*

The following command displays the portion of the block containing the offsets 31h through 45h.

*D 31, 45

BLOCK NUMBER = 20

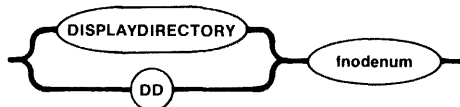
offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII STRING
0030	25	1F	00	00	2E	00	00	00	25	1F	00	00	2B	00	00	00	%.....%...+...
0040	01	00	00	00	01	00	80	00	00	00	00	00	00	00	00	00

*

DISPLAYDIRECTORY

DISPLAYDIRECTORY COMMAND

This command lists all the files contained in a directory. The format of the DISPLAYDIRECTORY command is as follows:



INPUT PARAMETER

fnodenum Number of the fnode that corresponds to a directory file. This number can range from 0 through (max fnodes - 1), where max fnodes is the number of fnodes defined when the volume was originally formatted. DISPLAYDIRECTORY lists all files contained in this directory.

OUTPUT

In response to the command, DISPLAYDIRECTORY lists information about all files contained in the specified directory. The format of this display is as follows:

FILE NAME	FNODE	TYPE	FILE NAME	FNODE	TYPE	FILE NAME	FNODE	TYPE
filenam	fnode	type	filenam	fnode	type	filenam	fnode	type
filenam	fnode	type	filenam	fnode	type	filenam	fnode	type
.
.
.

where:

filenam Name of the file contained in the directory.

fnode Number of the fnode that describes the file.

type Type of the file, either DATA (for data files) or DIR (for directory files).

DESCRIPTION

DISPLAYDIRECTORY displays a list of files contained in the specified directory, along with their fnode numbers and types. With this information you can use other disk verification commands to examine the individual files.

ERROR MESSAGES

argument error

You specified a nonnumeric character in the fnodenum parameter.

not a named disk

The volume you are verifying is not formatted as a named volume. Thus there are no directories to display.

fnodenum, fnode not a directory

The number you specified for the fnodenum parameter is not an fnode for a directory file.

fnodenum, fnode out of range

The number you specified for the fnodenum parameter is larger than the largest fnode number on the volume.

EXAMPLE

The following command lists the files contained in the directory whose fnode is fnode 5.

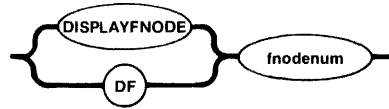
*DISPLAYDIRECTORY 5

FILE NAME	FNODE	TYPE	FILE NAME	FNODE	TYPE	FILE NAME	FNODE	TYPE
change.p86	0006	DATA	samp.txt	0007	DATA	NAMES	0008	DIR
PLACES	0009	DIR	change.plm	000A	DATA			

*

DISPLAYFNODE COMMAND

This command displays the fields associated with an fnode. The format of the DISPLAYFNODE command is as follows:



INPUT PARAMETER

fnodenum Number of the fnode to be displayed. This number can range from 0 through (max fnodes - 1), where max fnodes is the number of fnodes defined when the volume was originally formatted.

OUTPUT

In response to this command, DISPLAYFNODE displays the fields of the specified fnode. The format of the display is as follows:

```

Fnode number = fnodenum
                flags : flgs
                type  : typ
file gran/vol gran : gran
                owner : own
create,access,mod times : crtime, acctime, modtime
                total size : totsize
                total blks : totblks
                pointer(1) : blks, blkptr
                pointer(2) : blks, blkptr
                pointer(3) : blks, blkptr
                pointer(4) : blks, blkptr
                pointer(5) : blks, blkptr
                pointer(6) : blks, blkptr
                pointer(7) : blks, blkptr
                pointer(8) : blks, blkptr
                this size : thissize
                id count : count
accessor(1) : access, id
accessor(2) : access, id
accessor(3) : access, id
                parent  : prnt
                aux(*)  : auxbytes
  
```

OUTPUT (continued)

where:

fnodenum Number of the fnode being displayed. If the fnode does not describe an actual file (that is, if it is not allocated), the following message appears next to this field:

*** ALLOCATION STATUS BIT IN THIS FNODE NOT SET ***

In this case, the fnode fields are normally set to zero.

flgs A word defining the attributes of the file. Significant bits of this word are:

<u>bit</u>	<u>Meaning</u>
0	Allocation status. This bit is set to 1 for allocated fnodes and set to 0 for free fnodes.
1	Long or short file attribute. This bit is set to 1 for long files and set to 0 for short files.
5	Modification attribute. This bit is set to 1 whenever a file is modified.
6	Deletion attribute. This bit is set to 1 to indicate a temporary file or a file that is going to be deleted.

The DISPLAYFNODE command displays a message next to this field to indicate whether the file is a long or short file.

typ Type of file. This field contains a value and a message. The possible values and messages are:

<u>value</u>	<u>message</u>
00	fnode file
01	volume map file
02	fnode map file
03	account file
04	bad block file
06	directory file
08	data file

gran File granularity, specified as a multiple of the volume granularity.

own User ID of the owner of the file.

DISPLAYFNODE

OUTPUT (continued)

crtime Time and date of file creation, last access, and last
acctime modification. These values are expressed as the
modtime number of seconds since January 1, 1978.

totsize Total size, in bytes, of the actual data in the file.

totblks Total number of volume blocks used by the file,
including indirect block overhead.

blks, blkptr Values which identify the data blocks of the file. For
short files, each blks parameter indicates the number
of volume blocks in the data block and each blkptr is
the number of the first such volume block. For long
files, each blks parameter indicates the number of
volume blocks pointed to by an indirect block and each
blkptr is the block number of the indirect block.

thisize Size in bytes of the total data space allocated to the
file, minus any space used for indirect blocks.

count Number of user IDs associated with the file.

access, id Each pair of fields indicate the access rights for the
file (access) and the ID of the user who gains access
(id). Bits in the access field are set to indicate the
following access rights:

<u>bit</u>	<u>data file</u> <u>operation</u>	<u>directory</u> <u>operation</u>
0	delete	delete
1	read	display
2	append	add entry
3	update	change entry

The first ID listed is the owning user's ID.

prnt Fnode number of the directory file which contains the
file.

auxbytes Auxiliary bytes associated with the file.

Appendix A contains a more detailed description of the fnode fields.

DESCRIPTION

Fnodes are data structures on the volume that describe the files on the volume. The fnode structures are created when the volume is formatted. Each time a file is created on the volume, the iRMX 86 Operating System allocates an fnode for the file and fills in the fnode fields to describe the file. The DISPLAYFNODE command allows you to examine these fnodes and determine where the data for each file resides.

ERROR MESSAGES

argument error

You entered a value for the fnodeum parameter that was not a legitimate fnode number.

not a named disk

The volume you are verifying is not formatted as a named volume. Thus there are no fnodes to display.

fnodeum, fnode out of range

The number you specified for the fnodeum parameter is larger than the largest fnode number on the volume.

EXAMPLE

The following example displays fnode 6 of a volume.

*DISPLAYFNODE 6

Fnode number = 6

```

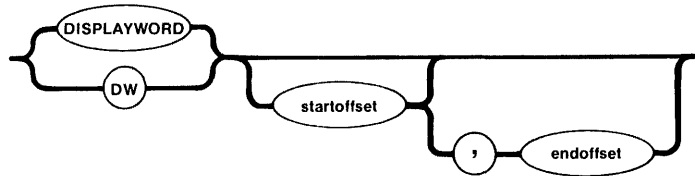
                flags : 0025 => short file
                  type : 08 => data file
    file gran/vol gran : 01
                  owner : FFFF => world
create,access,mod times : 00000017, 00000158, 00000018
                total size : 000003C1
                total blks : 00000008
                pointer(1) : 0008, 000050
                pointer(2) : 0000, 000000
                pointer(3) : 0000, 000000
                pointer(4) : 0000, 000000
                pointer(5) : 0000, 000000
                pointer(6) : 0000, 000000
                pointer(7) : 0000, 000000
                pointer(8) : 0000, 000000
                this size : 00000400
                id count : 0001
                accessor(1) : 0F, FFFF
                accessor(2) : 00, 0000
                accessor(3) : 00, 0000
                parent : 0005
                aux(*) : 000000

```

*

DISPLAYWORD COMMAND

This command displays the specified portion of the working buffer in word format. It displays the buffer in 8-word rows. The format of this command is as follows:



INPUT PARAMETERS

- startoffset** Number of the byte, relative to the start of the buffer, which begins the display. DISPLAYWORD starts the display with the row containing the specified offset. If you omit this parameter, DISPLAYWORD starts displaying from the beginning of the working buffer.
- endoffset** Number of the byte, relative to the start of the buffer, which ends the display. If you omit this parameter, DISPLAYWORD displays only the row indicated by startoffset. However, if you omit both startoffset and endoffset, DISPLAYWORD displays the entire working buffer.

OUTPUT

In response to the command, DISPLAYWORD displays the specified portion of the working buffer in rows, with 8 words displayed in each row. Figure 2-2 illustrates the format of the display.

As Figure 2-2 shows, DISPLAYWORD begins by listing the block number of the data being displayed. It then lists the specified portion of the buffer, providing the column numbers as a header and beginning each row with the relative address of the first word in the row.

OUTPUT (continued)

BLOCK NUMBER = blocknum

offset	0	2	4	6	8	A	C	E
0000	0100	0302	0504	0706	0908	0B0A	0DOC	0FOE
0010	0000	0000	0000	0000	0000	0000	0000	0000
0020	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFF
.		.				.		
.		.				.		
.		.				.		

Figure 2-2. DISPLAYWORD Format

DESCRIPTION

After you read a block of memory into the working buffer with the READ command, you can display part or all of that memory, in word format, by entering the DISPLAYWORD command. DISPLAYWORD displays the hexadecimal value for each word in the specified portion of the buffer.

If you omit all parameters, DISPLAYWORD displays the entire block stored in the working buffer.

ERROR MESSAGES

argument error

You made a syntax error in the command or specified a nonnumeric character in one of the offset parameters.

invalid offset

You either specified a larger value for startoffset than for endoffset or you specified an offset value that was larger than the number of bytes in the block.

DISPLAYWORD

EXAMPLES

Assuming that you have read block 20h into the working buffer with the READ command, the following command displays that block in word format.

*DISPLAYWORD

BLOCK NUMBER = 20

offset	0	2	4	6	8	A	C	E
0000	0000	0000	0000	0000	0000	0000	0000	0000
0010	0000	0080	0000	0000	0000	0001	FF0F	00FF
0020	0000	0000	0500	0000	0000	0025	0108	FFFF
0030	1F25	0000	002E	0000	1F25	0000	002B	0000
0040	0001	0000	0001	0080	0000	0000	0000	0000
0050	0000	0000	0000	0000	0000	0000	0000	0000
0060	0000	0000	0000	0000	0000	0000	0080	0000
0070	0000	0000	0001	FF0F	00FF	0000	0000	0500

*

The following command displays the portion of the block that contains the offsets 31h through 45h (words beginning at odd addresses).

*DW 31, 45

BLOCK NUMBER = 20

offset	0	2	4	6	8	A	C	E
0031	001F	2E00	0000	2500	001F	2B00	0000	0100
0041	0000	0100	8000	0000	0000	0000	0000	0000

*

The following command displays the portion of the block that contains the offsets 30h through 45h (words beginning at even addresses).

*DISPLAYWORD 30, 45

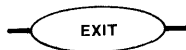
BLOCK NUMBER = 20

offset	0	2	4	6	8	A	C	E
0030	1F25	0000	002E	0000	1F25	0000	002B	0000
0040	0001	0000	0001	0080	0000	0000	0000	0000

*

EXIT COMMAND

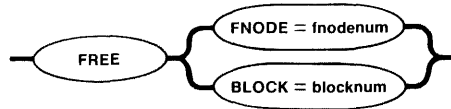
This command exits the disk verification utility and returns control to the Human Interface command level. The format of the EXIT command is as follows:



This command is identical to the QUIT command.

FREE COMMAND

This command designates fnodes and volume blocks as free (unallocated). The format of the FREE command is as follows:

**INPUT PARAMETERS**

- | | |
|----------|--|
| fnodenum | Number of the fnode to free. This number can range from 0 through (max fnodes - 1), where max fnodes is the number of fnodes defined when the volume was originally formatted. |
| blocknum | Number of the volume block to free. This number can range from 0 through (max blocks - 1), where max blocks is the number of volume blocks in the volume. |

OUTPUT

If you are using FREE to deallocate fnodes, FREE displays the following message:

fnodenum, fnode marked free

where fnodenum is the number of the fnode that the utility designated as free.

If you are using FREE to deallocate volume blocks, FREE displays the following message:

blocknum, block marked free

where blocknum is the number of the volume block that the utility designated as free.

FREE does not check the allocation status of fnodes or blocks before freeing them. Therefore, if you specify FREE for a block or fnode that is not allocated, FREE returns the same message as it would for a previously allocated block or fnode.

DESCRIPTION

Free fnodes are fnodes for which no actual files exist. FREE designates fnodes as free by updating both the FLAGS field of the fnode and the free fnodes map file.

Free volume blocks are blocks that are not part of any file; they are available to be assigned to any new or current file. FREE designates volume blocks as free by updating the volume free space map.

ERROR MESSAGES

argument error

You made a syntax error in the command or specified a nonnumeric character in the blocknum or fnodenum parameter.

blocknum, block out of range

The block number that you specified was larger than the largest block number in the volume.

fnodenum, fnode out of range

The fnode number that you specified was larger than the largest fnode number in the volume.

HELP COMMAND

This command lists all available DISKVERIFY commands and provides a short description of each command. The format of this command is:



OUTPUT

In response to this command, HELP displays the following information:

```

        read : read a disk block into the buffer
    display byte : display the buffer (byte format)
    display word : display the buffer (word format)
    substitute byte : modify the buffer (byte format)
    substitute word : modify the buffer (word format)
        write : write to the disk block from the buffer
    verify : verify the disk
        save : save free fnodes and space maps
    allocate/free : allocate/free an fnode or disk block
        disk : display disk attributes
    exit/quit : quit disk verify
    display directory : display the directory contents
    display fnode : display fnode information
miscellaneous commands :
    i. address : convert block number to absolute address
    ii. block : convert absolute address to block number
    iii. hex : display number as hexadecimal number
    iv. dec : display number as decimal number
    v. add : add two 16-bit numbers
    vi. sub : subtract a 16-bit number from a 32-bit number
    vii. mul : multiply two 16-bit numbers
    viii. div : divide a 32-bit number by a 16-bit number
    ix. mod : 32-bit number MODULO 16-bit number

```

MISCELLANEOUS COMMANDS

The following commands provide you with the ability to perform arithmetic and conversion operations within the disk verification utility. The commands perform the operations on unsigned numbers only and do not report any overflow conditions.

ADD

This command adds two numbers together. Its format is:



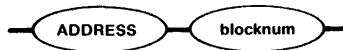
where:

arg1 and arg2 Numbers which the command adds together.

In response, the command displays the unsigned sum of the two numbers in both hexadecimal and decimal format.

ADDRESS

All memory in a volume is divided into volume blocks, which are areas of memory the same size as the volume granularity. Volume blocks are numbered sequentially in the volume, starting with the block containing the smallest addresses (block 0). The ADDRESS command converts a block number into an absolute address on the volume, so that you don't have to perform this conversion by hand. The format of this command is:



where:

blocknum Volume block number which ADDRESS converts into an absolute address. This parameter can range from 0 through (max blocks - 1), where max blocks is the number of volume blocks in the volume.

MISCELLANEOUS COMMANDS

ADDRESS (continued)

In response, ADDRESS displays the following information:

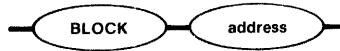
absolute address = addr

where:

addr Absolute address (in hexadecimal) that corresponds to the specified block number. This address represents the number of the byte that begins the block and can range from 0 through (volume size - 1), where volume size is the size, in bytes, of the volume.

BLOCK

The BLOCK command is the inverse of the address command. It converts a 32-bit absolute address into a volume block number, so that you don't have to perform this conversion by hand. The format of this command is:



where:

address Absolute address, which BLOCK converts into a block number. This parameter can range from 0 through (volume size - 1), where volume size is the size, in bytes, of the volume.

In response, BLOCK displays the following information:

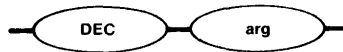
block number = blocknum

where:

blocknum Number of the volume block that contains the specified absolute address. The BLOCK command determines this value by dividing the absolute address by the volume block size and truncating the result.

DEC

This command finds the decimal equivalent of a number. Its format is:



where:

arg Number for which the command finds the decimal equivalent.

In response, the command displays the decimal equivalent of the specified number.

DIV

This command divides one number by another. Its format is:



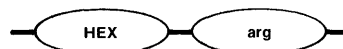
where:

arg1 and Numbers on which the command operates. It divides
arg2 arg1 by arg2.

In response, the command displays the unsigned, integer quotient in both hexadecimal and decimal format.

HEX

This command finds the hexadecimal equivalent of a number. Its format is:



MISCELLANEOUS COMMANDS

HEX (continued)

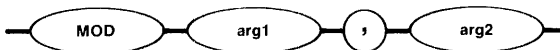
where:

arg Number for which the command finds the hexadecimal equivalent.

In response, the command displays equivalent of the specified number.

MOD

This command finds the remainder of one number divided by another. Its format is:



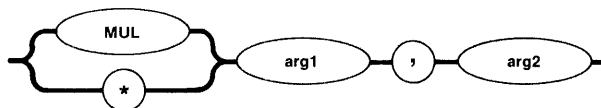
where:

arg1 and Numbers on which the command operates. It performs
arg2 the operation arg1 modulo arg2.

In response, the command displays the value arg1 modulo arg2 in both hexadecimal and decimal format.

MUL

This command multiplies two numbers together. Its format is:



where:

arg1 and Numbers which the command multiplies together.
arg2

In response, the command displays the unsigned product of the two numbers in both hexadecimal and decimal format.

SUB

This command subtracts one number from another. Its format is:



where:

arg1 and arg2 Numbers on which the command operates. The command subtracts arg2 from arg1.

In response, the command displays the unsigned difference in both hexadecimal and decimal format.

ERROR MESSAGES

argument error

You made a syntax error in the command, specified a nonnumeric value for one of the arguments, or specified a value for a block number parameter that was not a valid block number.

block number out of range

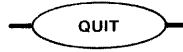
If the command was an ADDRESS command, the block number you entered was greater than the number of blocks in the volume. If the command was a BLOCK command, BLOCK converted the address to a volume block number, but the block number was greater than the number of blocks in the volume.

EXAMPLES

```
*MUL 134T, 13T
 6CE ( 1742T)
*+ 8, 4
 0C ( 12T)
*SUB 8884, 256
862E (34350T)
*MOD 1225, 256T
 25 ( 37T)
*HEX 155T
 9B
*ADDRESS 15
absolute address = 0A80
*
*BLOCK 2236
  block number = 44
```

QUIT COMMAND

This command exits the disk verification utility and returns control to the Human Interface command level. The format of the QUIT command is as follows:

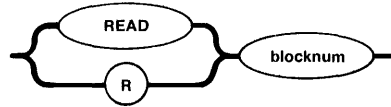


QUIT

This command is identical to the EXIT command.

READ COMMAND

This command reads a volume block from the disk into the working buffer. The format of the READ command is:



INPUT PARAMETER

blocknum	Number of the volume block to read. This number can range from 0 through (max blocks - 1), where max blocks is the number of volume blocks in the volume.
----------	---

OUTPUT

In response to the command, READ reads the block into the working buffer and displays the following message:

```
read block number:  blocknum
```

where blocknum is the number of the block.

DESCRIPTION

The READ command copies a specified volume block from the volume to the working buffer. It destroys any data currently in the working buffer. Once the block is in the working buffer, you can use DISPLAYBYTE and DISPLAYWORD to display the block and you can use SUBSTITUTEBYTE and SUBSTITUTEWORD to change the data in the block. Finally, you can use the WRITE command to write the modified block back out to the volume.

ERROR MESSAGES

argument error

You specified a nonnumeric character in the blocknum parameter.

blocknum, block out of range

The block number that you specified was larger than the largest block number in the volume.

SAVE COMMAND

This command writes the reconstructed free fnodes map and volume free space map to the volume being verified. (The NAMED2 option of the VERIFY command originally created the reconstructed maps.) The format of the SAVE command is:

**OUTPUT**

In response to this command, SAVE displays the following message:

save fnode map?

If you want to write the reconstructed free fnodes map to the volume, enter Y or YES. Otherwise, enter any other character or a carriage return alone. If you enter YES, SAVE writes the free fnodes map to the volume and displays the following message:

free fnode map saved

In any case, SAVE next displays the following message:

save space map?

If you want to write the reconstructed free space map to the volume, enter Y or YES. Otherwise, enter any other character or a carriage return alone. If you enter YES, SAVE writes the volume free space map to the volume and displays the following message:

free space map saved

DESCRIPTION

Whenever you perform a VERIFY function with the NAMED2 option (refer to the description of the VERIFY command for more information), VERIFY creates its own free fnodes map and volume free space map. It does this by examining all directories and fnodes on the volume, not by copying the maps that exist on the volume. To create the free fnodes map, it examines every directory on the volume to determine which fnodes represent actual files. To create the volume free space map, it examines the POINTER(n) fields of the fnodes to determine which volume blocks the files use.

DESCRIPTION (continued)

VERIFY then compares the newly created maps with the maps that exist on the volume. If a discrepancy exists, VERIFY displays a message to indicate the discrepancy.

The SAVE command takes the free fnodes map and the volume free space map created during the VERIFY operation and writes them to the volume, replacing the maps that currently exist.

ERROR MESSAGES

not a named disk

The volume being verified is not a named volume. Free fnode maps and volume free space maps exist only on named volumes.

nothing to save

You did not enter the VERIFY command with the NAMED2, NAMED, or ALL options prior to entering the SAVE command. Thus SAVE has no free fnode map and volume free space map with which to replace those that exist on the volume.

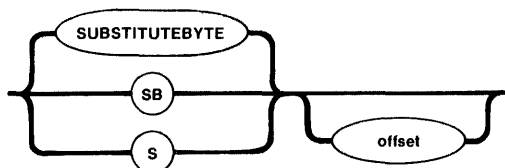
EXAMPLE

The following example illustrates the format of the SAVE command.

```
*SAVE
save fnode map? no
save space map? y
      free space map saved
*
```

SUBSTITUTEBYTE COMMAND

This command allows you to interactively change the contents of the working buffer (in byte format). The format of the SUBSTITUTEBYTE command is:



INPUT PARAMETER

offset	Number of the byte, relative to the start of the working buffer, which the command can change in response to user input. This number can range from 0 to (block size - 1), where block size is the size of a volume block (and thus the size of the working buffer). If you omit this parameter, the command assumes a value of 0.
--------	--

OUTPUT

In response to the command, SUBSTITUTEBYTE displays the specified byte and waits for you to enter a new value. This display appears as:

```
offset: val -
```

where offset is the number of the byte, relative to the start of the buffer, and val is the current value of the byte. At this point, you can enter one of the following:

- A value followed by a carriage return. This causes SUBSTITUTEBYTE to substitute the new value for the current byte. If the value you enter requires more than one byte of storage, SUBSTITUTEBYTE uses only the low-order byte of the value. SUBSTITUTEBYTE then displays the next byte in the buffer and waits for your further response.
- A carriage return alone. This causes SUBSTITUTEBYTE to leave the current value as is and display the next byte in the buffer. It then waits for your response.

OUTPUT (continued)

- A value followed by a period (.) and a carriage return. This causes SUBSTITUTEBYTE to substitute the new value for the current byte. It then exits from the SUBSTITUTEBYTE command and gives you the asterisk (*) prompt, permitting you to enter any DISKVERIFY command.
- A period (.) followed by a carriage return. This exits the SUBSTITUTEBYTE command and gives you the asterisk (*) prompt, permitting you to enter any DISKVERIFY command.

DESCRIPTION

The SUBSTITUTEBYTE command gives you the ability to interactively change bytes in the working buffer. Once you enter the command, SUBSTITUTEBYTE displays the offset and the value of the first byte. You can change the byte by entering a new byte value, or you can leave the byte as is by entering a carriage return only. The command then displays the next byte in the buffer. In this manner, you can consecutively step through the buffer, changing whatever bytes are appropriate. When you finish changing the buffer, you can enter a period followed by a carriage return to exit the command.

The SUBSTITUTEBYTE command considers the working buffer to be a circular buffer. That is, entering a carriage return when you are positioned at the last byte of the buffer causes SUBSTITUTEBYTE to display the first byte of the buffer.

The SUBSTITUTEBYTE command changes only the values in the working buffer. To make the changes in the volume, you must enter the WRITE command to write the working buffer back to the volume.

ERROR MESSAGES

argument error

You specified a nonnumeric character in the offset parameter.

invalid offset

You specified an offset value that was larger than the number of bytes in the block.

SUBSTITUTEBYTE

EXAMPLE

This example changes several bytes in two portions of the working buffer. Two SUBSTITUTEBYTE commands are used. Carriage returns are denoted by a <cr> to aid your understanding of this example.

*SUBSTITUTEBYTE<cr>

0000: A0 - 00<cr>
0001: 80 - <cr>
0002: E5 - <cr>
0003: FF - 31<cr>
0004: FF - .<cr>

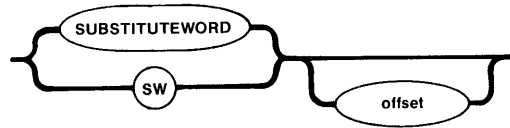
*SUBSTITUTEBYTE 40<cr>

0040: 00 - E6<cr>
0041: 00 - E6<cr>
0042: 00 - .<cr>

*

SUBSTITUTEWORD COMMAND

This command allows you to interactively change the contents of the working buffer (in word format). The format of the SUBSTITUTEWORD command is:



INPUT PARAMETER

offset Number of the byte, relative to the start of the working buffer, which the command can change in response to user input. This number can range from 0 to (block size - 1), where block size is the size of a volume block (and thus the size of the working buffer). If you omit this parameter, the command assumes a value of 0.

OUTPUT

In response to the command, SUBSTITUTEWORD displays the word beginning at the specified byte and waits for you to enter a new value. This display appears as:

offset: val -

where offset is the number of the byte which begins the word, relative to the start of the buffer, and val is the current value of the word. At this point, you can enter one of the following:

- A value followed by a carriage return. This causes SUBSTITUTEWORD to substitute the new value for the current word. If the value you enter requires more than one word of storage, SUBSTITUTEWORD uses only the low-order word of the value. SUBSTITUTEWORD then displays the next word in the buffer and waits for your further response.
- A carriage return alone. This causes SUBSTITUTEWORD to leave the current value as is and display the next word in the buffer. It then waits for your response.
- A value followed by a period (.) and a carriage return. This causes SUBSTITUTEWORD to substitute the new value for the current byte. It then exits from the SUBSTITUTEWORD command and gives you the asterisk (*) prompt, permitting you to enter any DISKVERIFY command.

SUBSTITUTEWORD

OUTPUT (continued)

- A period (.) followed by a carriage return. This exits the SUBSTITUTEWORD command and gives you the asterisk (*) prompt, permitting you to enter any DISKVERIFY command.

DESCRIPTION

The SUBSTITUTEWORD command is exactly like the SUBSTITUTEBYTE command except that it allows you to interactively modify words instead of bytes. Once you enter the command, SUBSTITUTEWORD displays the offset and the value of the first word. You can change the word by entering a new word value, or you can leave the word as is by entering a carriage return only. The command then displays the next word in the buffer. In this manner, you can consecutively step through the buffer, changing whatever words are appropriate. When you finish changing the buffer, you can enter a period followed by a carriage return to exit the command.

The SUBSTITUTEWORD command considers the working buffer to be a circular buffer. That is, entering a carriage return when you are positioned at the last byte of the buffer causes SUBSTITUTEWORD to display the first byte of the buffer.

The SUBSTITUTEWORD command changes only the values in the working buffer. To make the changes in the volume, you must enter the WRITE command to write the working buffer back to the volume.

ERROR MESSAGES

argument error

You specified a nonnumeric character in the offset parameter.

invalid offset

You specified an offset value that was larger than the number of bytes in the block.

EXAMPLE

This example changes several bytes in two areas of the working buffer. Two SUBSTITUTEWORD commands are used. Carriage returns are denoted by a <cr> to aid your understanding of this example.

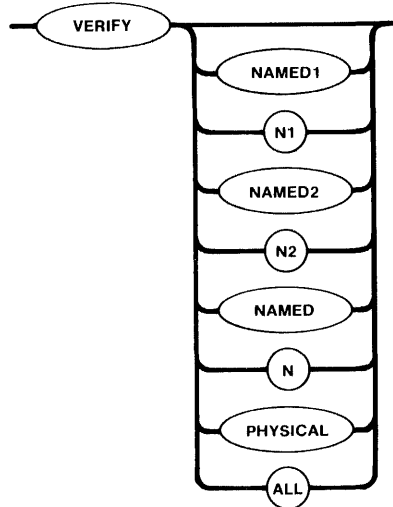
EXAMPLE (continued)

*SUBSTITUTEWORD<cr>0000: A0B0 - 0000<cr>0002: 8070 - <cr>0004: E511 - <cr>0006: FFFF - 3111<cr>0008: FFFF - .<cr>*SUBSTITUTEWORD 35<cr>0035: 0000 - E6FF<cr>0037: 0000 - E6AB<cr>0039: 0000 - .<cr>

*

VERIFY COMMAND

This command checks the structures on the volume to determine whether the volume is properly formatted. The format of the VERIFY command is:



INPUT PARAMETERS

NAMED1 or N1 Checks named volumes to ensure that the information recorded in the fnodes is consistent and matches the information obtained from the directories themselves. VERIFY performs the following operations during a NAMED1 verification:

- Checks fnode numbers in the directories to see if they correspond to allocated fnodes.
- Checks the parent fnode numbers recorded in the fnodes to see if they match with the information recorded in the directories.
- Checks the fnodes against the files to determine if the fnodes specify the proper file type.
- Checks the POINTER(n) structures of long files to see if the indirect blocks accurately reflect the number of blocks used by the file.
- Checks each fnode to see if the TOTAL SIZE, TOTAL BLKS, and THIS SIZE fields are consistent.

NAMED2 or N2 Checks named volumes to ensure that the information recorded in the free fnodes map and the volume free space map matches the actual files and fnodes. VERIFY performs the following operations during a NAMED2 verification:

INPUT PARAMETERS (continued)

- Creates a free fnodes map by examining every directory in the volume. It then compares that free fnodes map with the one already on the volume.
 - Creates a free space map by examining the information in the fnodes. It then compares that free space map with the one already on the volume.
 - Checks to see if the block numbers recorded in the fnodes and the indirect blocks actually exist.
 - Checks to see if two or more files use the same volume block.
 - Checks to see if two or more files use the same fnode.
- NAMED or N Performs both the NAMED1 and NAMED2 operations on a named volume. If you omit the parameter from the VERIFY command, NAMED is the default parameter.
- PHYSICAL Reads all blocks on the volume and checks for I/O errors. This parameter applies to both named and physical volumes.
- ALL Performs all operations appropriate to the volume. For named volumes, this option performs both the NAMED and PHYSICAL operations. For physical volumes, this option performs the PHYSICAL operations.

OUTPUT

VERIFY produces a different kind of output for each of the NAMED1, NAMED2, and PHYSICAL options. The NAMED and ALL options produce combinations of the first three kinds of output.

Figure 2-3 illustrates the format of the NAMED1 output.

```

DEVICE NAME = devname      : DEVICE SIZE = devsize : BLK SIZE = blksize

'NAMED1' VERIFICATION

FILE = (filename, fnodenum) : LEVEL = lev : PARENT = parnt : TYPE = typ
error messages
FILE = (filename, fnodenum) : LEVEL = lev : PARENT = parnt : TYPE = typ
error messages
      :                :                :                :
FILE = (filename, fnodenum) : LEVEL = lev : PARENT = parnt : TYPE = typ
error messages

```

Figure 2-3. NAMED1 Verification Output

VERIFY

OUTPUT (continued)

The following paragraphs identify the fields listed in Figure 2-3.

devname	Physical name of the device, as specified in the ATTACHDEVICE Human Interface command.
devsize	Hexadecimal size of the volume, in bytes.
blksize	Hexadecimal volume granularity. This number is the size of a volume block.
filename	Name of the file (1 to 14 characters).
fnodenum	Hexadecimal number of the file's fnode.
lev	Hexadecimal level of the file in the file heirarchy. The root directory of the volume is the only level 0 file. Files contained in the root directory are level 1 files. Files contained in level 1 directories are level 2 files. This numbering continues for all levels of files in the volume.
parnt	Fnode number of the directory which contains this file, in hexadecimal.
typ	File type, either DATA (for data files) or DIR (for directory files). If VERIFY cannot ascertain that the file is a directory or data file, it displays the characters "****" in this field.
error messages	Messages, which indicate the errors associated with the previously-listed file. The error messages which can occur are listed later in this section.

As Figure 2-3 shows, the NAMED1 option displays information about each file that is in error. The NAMED1 display also contains error messages which immediately follow the listing of the affected files.

Figure 2-4 illustrates the format of the NAMED2 output.

```
DEVICE NAME = devname      : DEVICE SIZE = devsize : BLK SIZE = blksize
'NAMED2' VERIFICATION
      BIT MAPS O.K.
```

Figure 2-4. NAMED2 Verification Output

The fields in Figure 2-4 are exactly the same as the corresponding fields in Figure 2-3.

OUTPUT (continued)

If VERIFY detects an error during NAMED2 verification, it displays one or more error message in place of the "BIT MAPS O.K." message.

Figure 2-5 illustrates the format of the PHYSICAL output.

```

DEVICE NAME = devname      : DEVICE SIZE = devsize : BLK SIZE = blksize
'PHYSICAL' VERIFICATION
      NO ERRORS
  
```

Figure 2-5. PHYSICAL Verification Output

The fields in Figure 2-5 are exactly the same as the corresponding fields in Figure 2-3.

If VERIFY detects an error during PHYSICAL verification, it displays one or more error message in place of the "NO ERRORS" message.

If you specify NAMED verification, VERIFY displays both the NAMED1 and NAMED2 output. If you specify the ALL verification for a named volume, VERIFY displays the NAMED1, NAMED2, and PHYSICAL output. If you specify the ALL verification for a physical volume, VERIFY displays the PHYSICAL output.

DESCRIPTION

The VERIFY command checks physical and named volumes to ensure that the volumes contain valid file structures and data areas. VERIFY can perform three kinds of verification: NAMED1, NAMED2, and PHYSICAL. NAMED1 and NAMED2 verifications check the file structures of named volumes. They do not apply to physical volumes. PHYSICAL verification checks each data block of the volume for I/O errors. PHYSICAL verification applies to both named and physical volumes.

As part of the NAMED2 verification, VERIFY creates a free fnodes map and a volume free space map which it compares with the corresponding maps on the volume. You can use the SAVE command to write the maps produced during NAMED2 verification to the volume, overwriting the maps on the volume.

VERIFY

ERROR MESSAGES

Four kinds of error messages can occur as a result of entering the VERIFY command: VERIFY command errors, NAMED1 errors, NAMED2 errors, and PHYSICAL errors.

VERIFY command errors

verify-function argument error

The parameter you specified is not a valid VERIFY parameter.

not a named disk

You tried to perform a NAMED, NAMED1, or NAMED2 verification on a physical volume.

NAMED1 errors

The following messages can appear in a NAMED1 display, immediately after the file to which they refer.

fnodenum, allocation status in this fnode not set

The file is listed in a directory but the flags field of its fnode indicates that fnode is free. The free fnodes map may or may not list the fnode as allocated.

numblocks, blocks in indirect block do not match #blks in the fnode

The file is a long file, and the number of blocks listed in a POINTER(n) field of the fnode does not agree with the number of blocks listed in the indirect block.

directory stack overflow

This message can indicate an internal error in the disk verification utility. However, it can also indicate that a directory on the volume lists, as one of its entries, itself or one of the parent directories in its pathname. If this happens, the utility, when it searches through the directory tree, continually loops through a portion of the tree, overflowing an internal buffer area.

file size inconsistent

total size = tosize : this size = thsize : data blocks = numblks

The TOTAL SIZE, THIS SIZE, and TOTAL BLKS fields of the fnode are inconsistent.

ERROR MESSAGES (continued)

fnode not on the disk

The fnode number of the file, as recorded in the directory file, is larger than the largest fnode number on the volume.

illegal file type

The file type of a user file, as recorded in TYPE field of the fnode, is other than directory (06) or data (08).

insufficient memory to create directory stack

There is not enough dynamic memory in the system for the utility to perform the verification.

invalid block# recorded in the fnode

One of the POINTER(n) fields in the fnode specifies a block number that is larger than the largest block number in the volume.

invalid block# recorded in the indirect block

The file is a long file and one of the indirect blocks specifies a block number that is larger than the largest block number in the volume.

parent fnode number does not match

The PARENT field of the fnode does not agree with the fnode number of the directory that contains the file. VERIFY displays the fnode number of the directory that contains the file, not the fnode number recorded in the PARENT field of the file's fnode.

total blocks does not reflect the data blocks correctly

The TOTAL BLKS field of the fnode and the number of blocks recorded in the POINTER(n) fields are inconsistent.

NAMED2 errors

The following messages can appear in a NAMED2 display.

blocknum, block allocated but not referenced

The volume free space map lists the specified volume block as allocated, but no fnode specifies the block as part of a file.

VERIFY

ERROR MESSAGES (continued)

blocknum, block referenced but not allocated

An fnode indicates that the specified volume block is part of a file, but the volume free space map lists the block as free.

directory stack overflow

This message can indicate an internal error in the disk verification utility. However, it can also indicate that a directory on the volume lists, as one of its entries, itself or one of the parent directories in its pathname. If this happens, the utility, when it searches through the directory tree, continually loops through a portion of the tree, overflowing an internal buffer area.

fnodenum, fnode map bit marked allocated but not referenced

The free fnodes map lists the specified fnode as allocated, but no directory contains a file with the fnode number.

fnodenum, fnode referenced but fnode map bit marked free

The specified fnode number is listed in a directory, but the free fnodes map lists the fnode as free.

insufficient memory to create directory stack

There is not enough dynamic memory in the system for the utility to perform the verification.

insufficient memory to create fnode and space maps

During a NAMED2 verification, the utility tried to create a free fnodes map and a volume free space map. However, there is not enough dynamic memory available in the system to create these maps.

blocknum, multiple reference to this block

More than one fnode specifies this block as part of a file.

fnodenum, multiple reference to this fnode

The directories on the volume list more than one file associated with this fnode number.

ERROR MESSAGES (continued)

PHYSICAL errors

blocknum, error

An I/O error occurred when VERIFY tried to access the specified volume block. The volume is probably flawed.

other errors

The following error messages indicate internal errors in the disk verification utility. Under normal conditions these messages should never appear. However, if these messages (or other undocumented messages that also appear to indicate internal problems) do appear during a NAMED1 or NAMED2 verification, you should exit the disk verification utility and re-enter the DISKVERIFY command.

directory stack underflow
 directory stack empty
 directory stack error

EXAMPLE

The following command performs both named and physical verification on a named volume.

*VERIFY ALL

DEVICE NAME = F1 : DEVICE SIZE = 0003E900 : BLK SIZE = 0080

'NAMED1' VERIFICATION

'NAMED2' VERIFICATION

BIT MAPS O.K.

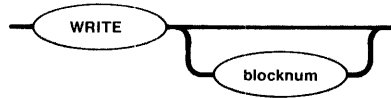
'PHYSICAL' VERIFICATION

NO ERRORS

*

WRITE COMMAND

This command writes the contents of the working buffer to the volume. The format of this command is:



INPUT PARAMETER

blocknum	Number of the volume block to which the command writes the working buffer. If you omit this parameter, WRITE writes the buffer back to the block most recently accessed.
----------	--

OUTPUT

In response to the command, WRITE displays the following message:

write to (blocknum)?

where blocknum is the number of the volume block to which WRITE intends to write the working buffer. If you respond by entering Y or any character string beginning with Y, WRITE copies the working buffer to the specified block on the volume. Any other response aborts the write process.

DESCRIPTION

The WRITE command is used in conjunction with the READ, DISPLAYBYTE, DISPLAYWORD, SUBSTITUTEBYTE, and SUBSTITUTEDWORD commands to modify information on the volume. Initially you use READ to copy a volume block from the volume to a working buffer. Then you can use DISPLAYBYTE and DISPLAYWORD to view the buffer and SUBSTITUTEBYTE and SUBSTITUTEDWORD to change the buffer. Finally, you can use WRITE to write the modified buffer back to the volume. By default, WRITE copies the buffer to the block most recently accessed by a READ or WRITE command.

A WRITE command does not destroy the data in the working buffer. The data remains the same until the next SUBSTITUTEBYTE, SUBSTITUTEDWORD, or READ command modifies the buffer.

ERROR MESSAGES

argument error

You made a syntax error or specified nonnumeric characters in the blocknum parameter.

blocknum, block out of range

The block number you specified was larger than the largest block number in the volume.

EXAMPLE

The following command copies the working buffer to the block from which it was read.

```
*WRITE  
write to (4B)? y  
*
```


APPENDIX A. STRUCTURE OF iRMX™ 86 NAMED VOLUMES

This appendix describes the structure of an iRMX 86 volume that contains named files. Those users who wish to examine named file volumes or create their own formatting utility programs can use this information.

This appendix is intended for system programmers who have had experience in reading and writing actual volume information. It does not attempt to teach the reader these functions.

INTRODUCTION

Each iRMX 86 named volume contains ISO (International Organization for Standardization) label information as well as iRMX 86 label information and files. Figure A-1 illustrates the general structure of a named file volume.

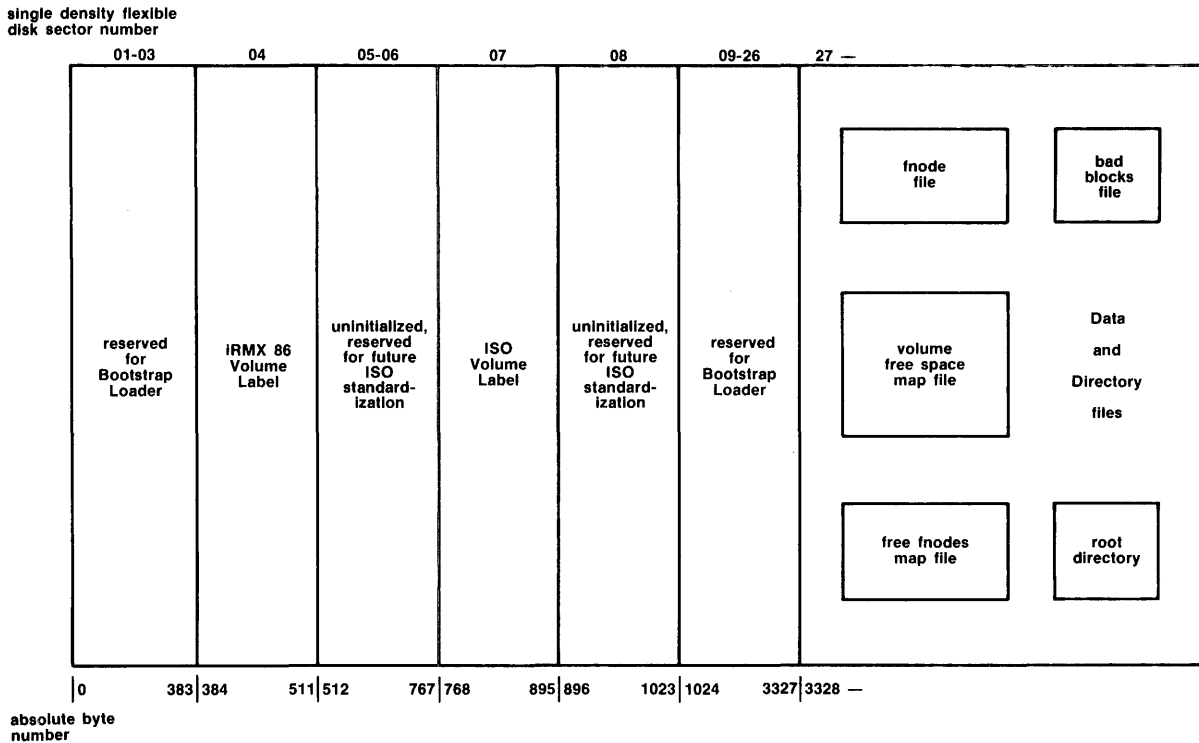


Figure A-1. General Structure of Named Volumes

STRUCTURE OF iRMX 86™ NAMED VOLUMES

This appendix discusses the structure in more detail. It includes information concerning the following:

- ISO Volume Label
- iRMX 86 Volume Label
- fnode file
- volume free space map file
- free fnodes map file
- bad blocks file
- root directory

It also discusses the structure of directory files and the concepts of long and short files.

The blocks in Figure A-1 that are reserved for the Bootstrap Loader are not discussed. To include these blocks on a new volume that you are formatting, you should copy them from an already formatted volume.

NOTE

The following sections of this appendix refer to a data type called DWORD. DWORD must be declared literally as POINTER. This results in a 32-bit variable for the PLM/86 models COMPACT, MEDIUM, and LARGE.

VOLUME LABELS

This section describes the structure of the volume labels that must be present on a named volume. These labels are the ISO volume label and the iRMX 86 volume label.

ISO VOLUME LABEL

The ISO (International Organization for Standardization) volume label is recorded in absolute byte positions 768 through 895 of the volume (for example, sector 07 of a single density flexible diskette). The structure of this volume label is as follows:

STRUCTURE OF iRMX 86™ NAMED VOLUMES

DECLARE

```

ISO$VOL$LABEL STRUCTURE(
    LABEL$ID(3)          BYTE,
    RESERVED$A          BYTE,
    VOL$NAME(6)         BYTE,
    VOL$STRUC           BYTE,
    RESERVED$B(60)     BYTE,
    REC$SIDE            BYTE,
    RESERVED$C(4)      BYTE,
    ILEAVE(2)           BYTE,
    RESERVED$D          BYTE,
    ISO$VERSION         BYTE,
    RESERVED$E(48)     BYTE);

```

where:

LABEL\$ID(3)	Label identifier. For named file volumes, this field contains the ASCII characters "VOL".
RESERVED\$A	Reserved field containing the ASCII character "1".
VOL\$NAME(6)	Volume name. This field can contain up to six printable ASCII characters, left justified and space filled. A value of all spaces implies that the volume name is recorded in the iRMX 86 Volume Label (absolute byte positions 384-393).
VOL\$STRUC	For named file volumes, this field contains the ASCII character "N", indicating that this volume has a non-ISO file structure.
RESERVED\$B(60)	This is a reserved field containing 60 bytes of ASCII spaces.
REC\$SIDE	For named file volumes, this field contains the ASCII character "1" to indicate that only one side of the volume is to be recorded.
RESERVED\$C(4)	This is a reserved field containing four bytes of ASCII spaces.
ILEAVE(2)	Two ASCII digits indicating the interleave factor for the volume, in decimal. ASCII digits consist of the numbers 0 through 9. When formatting named volumes, you should set this field to the same interleave factor that you use when physically formatting the volume.
RESERVED\$D	This is a reserved field containing an ASCII space.
ISO\$VERSION	For named file volumes, this field contains the ASCII character "1", which indicates ISO version number one.
RESERVED\$D(48)	This is a reserved field containing 48 ASCII spaces.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

iRMX 86 VOLUME LABEL

The iRMX 86 Volume Label is recorded in absolute byte positions 384 through 511 of the volume (sector 04 of a single density flexible diskette). The structure of this volume label is as follows:

```

DECLARE
  RMX$VOLUME$INFORMATION  STRUCTURE(
    VOL$NAME(10)           BYTE,
    FLAGS                  BYTE,
    FILE$DRIVER            BYTE,
    VOL$GRAN                WORD,
    VOL$SIZE                DWORD,
    MAX$FNODE              WORD,
    FNODE$START            DWORD,
    FNODE$SIZE             WORD,
    ROOT$FNODE             WORD,
    DEV$GRAN               WORD,
    INTERLEAVE             WORD,
    TRACK$SKEW             WORD,
    SYSTEM$ID              WORD,
    SYSTEM$NAME(12)        BYTE,
    DEVICE$SPECIAL(8)     BYTE);
  
```

where:

VOL\$NAME(10) Volume name in printable ASCII characters, left justified and zero filled.

FLAGS BYTE which lists the device characteristics for automatic device recognition. The individual bits in this byte indicate the following characteristics (bit 0 is rightmost bit):

<u>Bit</u>	<u>Meaning</u>
0	VF\$AUTO flag. When set to one, this bit indicates that the FLAGS byte contains valid data for automatic device recognition. When set to zero, it indicates that the remaining flags contain meaningless data.
1	VF\$DENSITY flag. This bit indicates the recording density of the volume. When set to one, it indicates modified frequency modulation (MFM) or double-density recording. When set to zero, it indicates frequency modulation (FM) or single-density recording.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

	<u>Bit</u>	<u>Meaning</u>
	2	VF\$SIDES flag. This bit indicates the number of recording sides on the volume. When set to one, it indicates a double-sided volume. When set to zero, it indicates a single-sided volume.
	3	VF\$MINI flag. This bit indicates the size of the recording media. When set to one, it indicates a 5 1/4-inch volume. When set to zero, it indicates an 8-inch volume.
FILE\$DRIVER		Number of the file driver used with this volume. For named file volumes, this field is set to four.
VOL\$GRAN		Volume granularity, specified in bytes. This value must be a multiple of the device granularity. It sets the size of a logical device block, also called a volume block.
VOL\$SIZE		Size of the entire volume, in bytes.
MAX\$FNODE		Number of fnodes in the fnode file. Refer to the next section for a description of fnodes.
FNODE\$START		A 32-bit value which represents the number of the first byte in the fnode file (byte 0 is the first byte of the volume).
FNODE\$SIZE		Size of an fnode, in bytes.
ROOT\$FNODE		Number of the fnode describing the root directory. Refer to the next section for further information.
DEV\$GRAN		Device granularity of all tracks except track zero (which contains the volume label). This field is important only when the system requires automatic device recognition.
INTERLEAVE		Block interleave factor for this volume. This value indicates the physical distance, in blocks, between consecutively-numbered blocks on the volume. A value of one indicates that consecutively-numbered blocks are adjacent. A value of zero indicates an unknown or undefined interleave factor.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

TRACK\$SKEW Offset, in bytes, between the first block on one track and the first block on the next track. A value of zero indicates that all tracks are identical.

SYSTEM\$ID Numerical code identifying the operating system that formatted the volume. The following codes are reserved for Intel operating systems:

<u>Operating System</u>	<u>Code</u>
iRMX 86	0 - 0Fh
iRMX 88	10h - 1Fh
OS 88	20h - 2Fh

Currently, the iRMX 86 Operating System places a zero in this field.

SYSTEM\$NAME(12) Name of the operating system which formatted the volume, in printable ASCII characters, left justified and space filled. Zeros (ASCII nulls) indicate that the operating system is unknown. The iRMX 86 Operating System currently places several pieces of information into this field, as follows:

- The left-most six bytes of this field contain the ASCII characters "iRMX86" to identify the operating system. Former iRMX 86 releases filled this field with zeros.
- The next byte is an ASCII character which identifies the program that formatted the volume. The following characters apply:

<u>Character</u>	<u>Formatting Program</u>
F	Human Interface FORMAT command
U	iRMX 86 Files Utility

If the formatting program is unable to provide this information, it places an ASCII space in this field.

- The next two bytes contain a two-digit ASCII sequence number which is incremented by the formatting program each time the formatting program changes in a way that affects the volume format. The Release 4 FORMAT Human Interface command places the characters "00" in this field.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

- The right-most three bytes of the field contain a three-digit ASCII number specifying the version of the Basic I/O System that was used in formatting the volume (for example, the characters "030" would indicate version 3.0). If the formatting program is unable to obtain this information, it places ASCII spaces in this field.

DEVICE\$SPECIAL(8) Reserved for special device-specific information. When no device-specific information exists, this field must contain zeros. If the device is a Winchester disk with an iSBC 215 controller or if the device is a disk with an iSBC 220 controller, the iRMX 86 Operating System imposes a structure on this field and supplies the following information:

SPECIAL	STRUCTURE(
CYLINDERS	WORD,
FIXED	BYTE,
REMOVABLE	BYTE,
SECTORS	BYTE,
SECTOR_SIZE	WORD,
ALTERNATES	BYTE);

where:

CYLINDERS	Total number of cylinders on the drive.
FIXED	Number of heads on the fixed disk or Winchester disk.
REMOVABLE	Number of heads on the removable disk cartridge.
SECTORS	Number of sectors in a track.
SECTOR_SIZE	Sector size, in bytes.
ALTERNATES	Number of alternate cylinders.

The remainder of the Volume Label (bytes 430 through 511) is reserved and must be set to zero.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

INITIAL FILES

Any mechanism that formats iRMX 86 named volumes must place five files on the volume during the format process. These five files are the fnode file, the volume free space map file, the free fnodes map file, the bad blocks file, and the root directory. The first of these files, the fnode file, contains information about all of the files on the volume. The general structure of the fnode file is discussed first. Then all of the files are discussed in terms of their fnode entries and their functions.

FNODE FILE

A data structure called a file descriptor node (or fnode) describes each file in a named file volume. All the fnodes for the entire volume are grouped together in a file called the fnode file. When the I/O System accesses a file on a named volume, it examines the iRMX 86 Volume Label (described in the previous section) to determine the location of the fnode file, and then examines the appropriate fnode to determine the actual location of the file.

When a volume is formatted, the fnode file contains six allocated fnodes. These fnodes represent the fnode file, the volume free space map file, the free fnodes map file, the bad blocks file, the root directory, and one other file. Later sections of this chapter describe these files. The size of the fnode file is determined by the number of fnodes that it contains. The number of fnodes in the fnode file also determines the number of files that can be created on the volume.

NOTE

When formatting a volume, you may be able to improve performance by placing the fnode file in the middle of the volume. By doing this, you reduce the average latency by 50%. For applications that have heavy file access, this may be desirable. However, the fnode file must start on a volume block boundary.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

The structure of an individual fnode in a named file volume is as follows:

```

DECLARE
  FNODE      STRUCTURE(
    FLAGS      WORD,
    TYPE       BYTE,
    GRAN       BYTE,
    OWNER      WORD,
    CR$TIME    DWORD,
    ACCESS$TIME DWORD,
    MOD$TIME    DWORD,
    TOTAL$SIZE DWORD,
    TOTAL$BLKS DWORD,
    POINTR(40) BYTE,
    THIS$SIZE  DWORD,
    RESERVED$A WORD,
    RESERVED$B WORD,
    ID$COUNT  WORD,
    ACC(9)     BYTE,
    PARENT     WORD,
    AUX(*)     BYTE);
  /* PTR(8)      STRUCTURE( */
  /*   NUM$BLOCKS WORD, */
  /*   BLK$PTR(3)  BYTE); */
  /* ACCESSOR(3) STRUCTURE( */
  /*   ACCESS      BYTE, */
  /*   ID          WORD); */

```

where:

FLAGS A WORD which defines a set of attributes for the file. The individual bits in this word indicate the following attributes (bit 0 is the rightmost bit):

<u>Bit</u>	<u>Meaning</u>
0	Allocation status. If set to one, this fnode describes an actual file. If set to zero, this fnode is available for allocation. When formatting a volume, this bit is set to one in the six allocated fnodes. In other fnodes, it is set to zero.
1	Long or short file attribute. This bit describes how the PTR fields of the fnode are interpreted. If set to zero, indicating a short file, the PTR fields identify the actual data blocks of the file. If set to one, indicating a long file, the PTR fields identify indirect blocks. Indirect blocks are described later in this section. When formatting a volume, this bit is always set to zero, since the initial files on the volume are short files.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>Bit</u>	<u>Meaning</u>
2	Reserved bit which is always set to one.
3-4	Reserved bits which are always set to zero.
5	Modification attribute. Whenever a file is modified, this bit is set to one. Initially, when a volume is formatted, this bit is set to zero in each fnode.
6	Deletion attribute. This bit is set to one to indicate that the file is a temporary file or that the file is going to be deleted (the deletion may be postponed because additional connections exist to the file). Initially, when the volume is formatted, this bit is set to zero in each fnode.
7-15	Reserved bits which are always set to zero.

TYPE Type of file. The following are acceptable types:

<u>Mnemonic</u>	<u>Value</u>	<u>Type</u>
FT\$FNODE	0	fnode file
FT\$VOLMAP	1	volume free space map
FT\$FNODEMAP	2	free fnodes map
FT\$ACCOUNT	3	space accounting file
FT\$BADBLOCK	4	bad device blocks file
FT\$DIR	6	directory file
FT\$DATA	8	data file

During system operation, only the I/O System can access file types other than FT\$DATA and FT\$DIR. These file types are discussed later in this section.

GRAN File granularity, specified in multiples of the volume granularity. The default value is 1. For the files initially present on the volume (fnode file, volume free space map file, free fnodes map file, bad blocks file, root directory), this value can be set to any multiple of the volume granularity.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

OWNER User ID of the owner of the file. For the files initially present on the volume, this parameter is important only for the root directory. For the root directory, this parameter should specify the user WORLD (FFFFH). The I/O System does not examine this parameter for the other files (fnode file, volume free space map file, free fnodes map file, bad blocks file) and so a value of zero can be specified.

CR\$TIME Time and date that the file was created, expressed as a 32-bit value. This value indicates the number of seconds since a fixed, user-determined point in time. By convention, this point in time is 12:00 A.M., January 1, 1978. For the files initially present on the volume, this parameter is important only for the root directory. A zero can be specified for the other files (fnode file, volume free space map file, free fnodes map file, bad blocks file).

ACCESS\$TIME Time and date of the last file access (read or write), expressed as a 32-bit value. For the files initially present on the volume, this parameter is important only for the root directory.

MOD\$TIME Time and date of the last file modification, expressed as a 32-bit value. For the files initially present on the volume, this parameter is important only for the root directory.

TOTAL\$SIZE Total size, in bytes, of the actual data in the file.

TOTAL\$BLKS Total number of volume blocks used by this file, including indirect block overhead. A volume block is a block of data whose size is the same as the volume granularity. All memory in the volume is divided into volume blocks, which are numbered sequentially, starting with the block containing the smallest addresses (block 0). Indirect blocks are discussed later in this section.

POINTR(40) A group of bytes on which the following structure is imposed:

```
    PTR(8)          STRUCTURE(  
        NUM$BLOCKS    WORD,  
        BLK$PTR(3)    BYTE);
```

STRUCTURE OF IRMX 86™ NAMED VOLUMES

This structure identifies the data blocks of the file. These data blocks may be scattered throughout the volume, but together they make up a complete file. If the file is a short file (bit 1 of the FLAGS field is set to zero), each PTR structure identifies an actual data block. In this case, the fields of the PTR structure contain the following:

NUM\$BLOCKS	Number of volume blocks in the data block.
BLK\$PTR(3)	A 24-bit value specifying the number of the first volume block in the data block. Volume blocks are numbered sequentially, starting with the block with the smallest address (block 0). The bytes in the BLK\$PTR array range from least significant (BLK\$PTR(0)) to most significant (BLK\$PTR(2)).

If the file is a long file (bit 1 of the FLAGS field is set to one), each PTR structure identifies an indirect block (possibly consisting of more than one contiguous volume block), which in turn identifies the data blocks of the file. In this case, the fields of the PTR structure contain the following:

NUM\$BLOCKS	Number of volume blocks pointed to by the indirect block.
BLK\$PTR(3)	A 24-bit volume block number of the indirect block.

Indirect blocks are discussed later in this section.

THIS\$SIZE	Size, in bytes, of the total data space allocated to the file. This figure does not include space used for indirect blocks, but it does include any data space allocated to the file, regardless of whether the file fills that allocated space.
RESERVED\$A	Reserved field which is set to zero.
RESERVED\$B	Reserved field which is set to zero.
ID\$COUNT	Number of access-ID pairs declared in the ACC(9) field.
ACC(9)	A group of bytes on which the following structure is imposed:

ACCESSOR(3)	STRUCTURE(
ACCESS	BYTE,
ID	WORD);

STRUCTURE OF IRMX 86™ NAMED VOLUMES

This structure contains the access-ID pairs which define the access rights for the users of the file. By convention, when a file is created, the owning user's ID is inserted in ACCESSOR(0), along with the code for the access rights. The fields of the ACCESSOR structure contain the following:

ACCESS Encoded access rights for the file. The settings of the individual bits in this field grant (if set to one) or deny (if set to zero) permission for the corresponding operation. Bit 0 is the rightmost bit.

<u>Bit</u>	<u>Data File Operation</u>	<u>Directory Operation</u>
0	delete	delete
1	read	display
2	append	add entry
3	update	change entry
4-7	reserved (must be 0)	

ID ID of the user who gains the corresponding access permission.

PARENT Fnode number of directory file which lists this file. For files initially present on the volume, this parameter is important only for the root directory. For the root directory, this parameter should specify the number of the root directory's own fnode. For other files (fnode file, volume free space map file, free fnodes map file, bad blocks file) the I/O System does not examine this field.

AUX(*) Auxiliary bytes associated with the file. The named file driver does not interpret this field, but the user can access it by making GET\$EXTENSION\$DATA and SET\$EXTENSION\$DATA system calls (refer to the IRMX 86 SYSTEM PROGRAMMER'S REFERENCE MANUAL). The size of this field is determined by the size of the fnode, which is specified in the IRMX 86 Volume Label. The Files Utility allocates three bytes for this field by default. If you use the Human Interface FORMAT command or create your own utility to format a volume, you can make this field as large as you wish; however, a larger AUX field implies slower file access.

Certain fnodes designate special files that appear on the volume. The following sections discuss these fnodes and the associated files.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

FNODE 0 (FNODE FILE)

The first fnode structure in the fnode file describes the fnode file itself. This file contains all the fnode structures for the entire volume. It must reside in contiguous locations in the volume. Fields of fnode 0 must be set as follows:

- The bits in the FLAGS field are set to the following (bit 0 is the rightmost bit):

<u>Bit</u>	<u>Value</u>	<u>Description</u>
0	1	Allocated file
1	0	Short file
2	1	Primary fnode
3-4	0	Reserved bits
5	0	Initial status is unmodified
6	0	File will not be deleted
7-15	0	Reserved bits

- The TYPE field is set to FT\$FNODE.
- The GRAN field is set to 1.
- The OWNER field is set to 0.
- The CR\$TIME, ACCESS\$TIME, and MOD\$TIME fields are set to 0.
- Since the iRMX 86 Volume Label specifies the size of an individual fnode structure and the number of fnodes in the fnode file, the value specified in the TOTAL\$SIZE field of fnode 0 must equal the product of the values in the FNODE\$SIZE and MAX\$FNODE fields of the iRMX 86 Volume Label.
- The TOTAL\$BLOCKS field specifies enough volume blocks to account for the memory listed in the TOTAL\$SIZE field. The product of the value in the TOTAL\$BLOCKS field and the volume granularity equals the value of the THIS\$SIZE field, since the fnode file is a short file.
- Since the fnode file must reside in contiguous locations in the volume, only one PTR structure describes the location of the file. The value in the NUM\$BLOCKS field of that PTR structure equals the value in the TOTAL\$BLOCKS field. The BLK\$PTR field indicates the number of the first block of the fnode file.
- The ID\$COUNT field is set to zero, indicating that no users can access the file.

FNODE 1 (VOLUME FREE SPACE MAP FILE)

The second fnode, fnode 1, describes the volume free space map file. The TYPE field for fnode 1 is set to FT\$VOLMAP to designate the file as such.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

The volume free space map file keeps track of all the space on the volume. It is a bit map of the volume, in which each bit represents one volume block (a block of space whose size is the same as the volume granularity). If a bit in the map is set to one, the corresponding volume block is free to be allocated to any file. If a bit in the map is set to zero, the corresponding volume block is already allocated to a file. The bits of the map correspond to volume blocks such that bit n of byte m represents volume block $(8 * m) + n$. The bits in the remaining space allocated to the map file (those that do not correspond to actual blocks of memory) must be set to zero.

When the volume is formatted, the volume free space map file indicates that the first 3328 bytes of the volume (the label and bootstrap information) plus any files initially placed on the volume (fnode file, volume free space map file, free fnodes map file, bad blocks file) are allocated.

FNODE 2 (FREE FNODES MAP FILE)

The third fnode, fnode 2, describes the free fnodes map file. The TYPE field of fnode 2 is set to FT\$FNODEMAP to designate the file as such.

The free fnodes map file keeps track of all the fnodes in the fnodes file. It is a bit map in which each bit represents an fnode. If a bit in the map is set to one, the corresponding fnode is not in use and does not represent an actual file. If a bit in the map is set to zero, the corresponding fnode already describes an existing file. The bits in the map correspond to fnodes such that bit n of byte m represents fnode number $(8 * m) + n$. The bits in the remaining space allocated to the map file (those that do not correspond to actual fnode structures) must be set to zero.

When the volume is formatted, the free fnodes map file indicates that fnodes 0, 1, 2, 3, and 4 are in use. If other files are initially placed on the volume, the free fnodes map file must be set to indicate this as well.

FNODE 4 (BAD BLOCKS FILE)

The fifth fnode, fnode 4, contains all the bad blocks on the volume. The TYPE field of fnode 4 is set to FT\$BADBLOCK to indicate this.

If there are any unusable blocks on a volume, this fnode must be initialized to describe a file which consists of all such bad blocks. If there are no bad blocks on the volume, the fnode must still be set up as allocated, and of the indicated type, but it should not assign any actual space for the file.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

ROOT DIRECTORY

The root directory is a special directory file. It is the root of the named file heirarchy for the volume. The IRMX 86 Volume Label specifies the fnode number of the root directory (normally fnode 5). The root directory is its own parent. That is, the PARENT field of its fnode specifies its own fnode number.

The root directory (and all directory files) associates file names with fnode numbers. It consists of a number of entries that have the following structure:

```
DECLARE
    DIR$ENTRY      STRUCTURE(
        FNODE      WORD,
        COMPONENT(14) BYTE);
```

where:

FNODE Fnode number of a file listed in the directory.

COMPONENT(14) A string of ASCII characters that is the final component of the path name identifying the file. This string is left justified and null padded to 14 characters.

When a file is deleted, its fnode number in the directory entry is set to zero.

OTHER FNODES

When a volume is formatted, one other fnode is set up, fnode 3, representing a file of type FT\$ACCOUNT, The fnode is set up as allocated, and of the indicated type, but it does not assign any actual space for the file.

When formatting a volume, no other fnodes in the fnode file represent actual files. The remaining fnodes must have bit zero (allocation status) set to zero.

LONG AND SHORT FILES

A file on a volume is not necessarily one contiguous string of bytes. In many cases, it consists of several contiguous blocks of data scattered throughout the volume. The fnode for the file indicates the locations and sizes of these blocks in one of two ways, as short files or as long files.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

SHORT FILES

If the file consists of eight or less distinct blocks of data, its fnode can specify it as a short file. The fnode for a short file has bit 1 of the FLAGS field set to zero. This indicates to the I/O System that the PTR structures of the fnode identify the actual data blocks that make up the file. Figure A-2 illustrates an fnode for a short file. Decimal numbers are used in the figure for clarity.

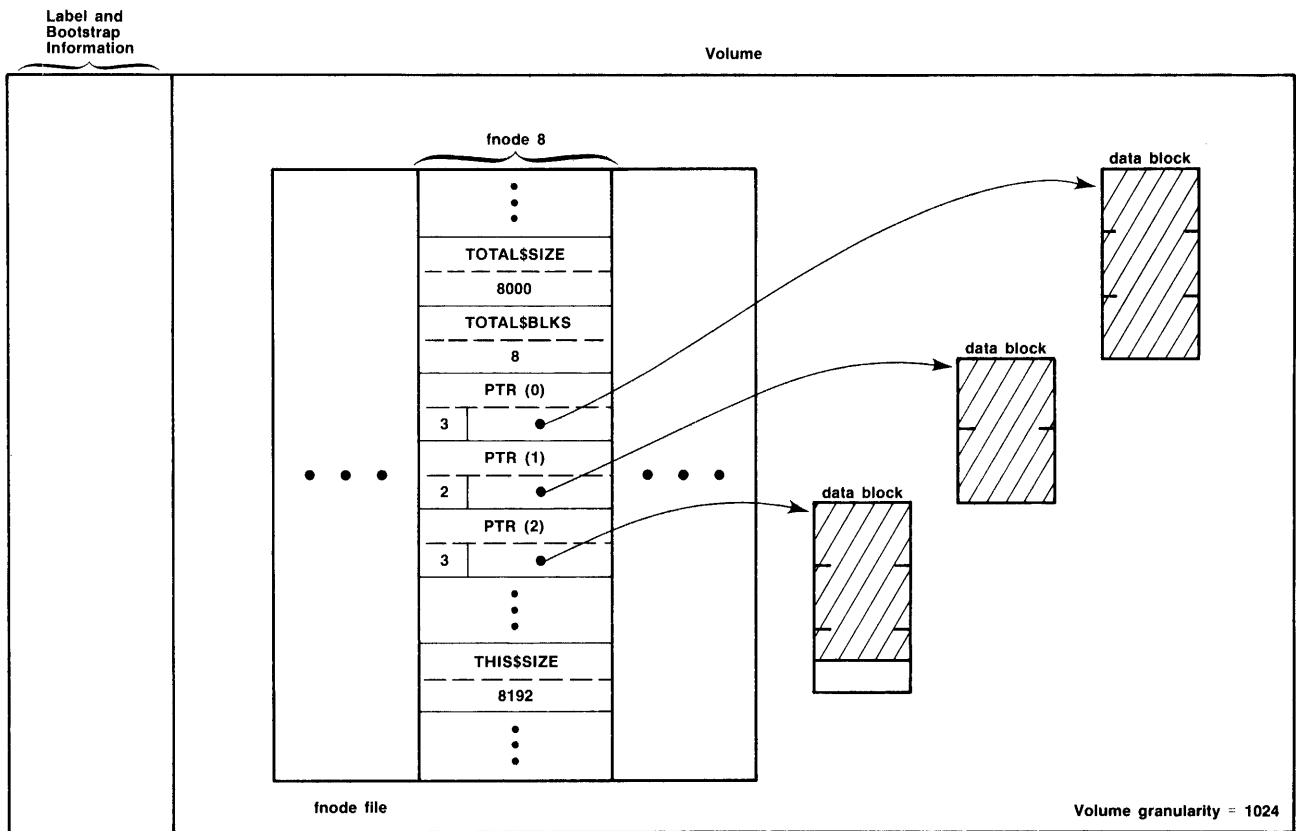


Figure A-2. Short File Fnode

As you can see from Figure A-2, fnode 8 identifies the short file. The file consists of three distinct data blocks. Three PTR structures give the locations of the data blocks. The NUM\$BLOCKS field of each PTR structure gives the length of the data block (in volume blocks) and the BLK\$PTR field points to the first volume block of the data block.

STRUCTURE OF IRMX 86™ NAMED VOLUMES

The other fields shown in Figure A-2 include TOTAL\$BLKS, THIS\$SIZE, and TOTAL\$SIZE. The TOTAL\$BLKS field specifies the number of volume blocks allocated to the file, which in this case is eight. This equals the sum of NUM\$BLOCKS values (3 + 2 + 3), since short files use all allocated space as data space.

The THIS\$SIZE field specifies the number of bytes of data space allocated to the file. This is the sum of the NUM\$BLOCKS values (3 + 2 + 3) multiplied by the volume granularity (1024) and equals 8192.

The TOTAL\$SIZE field specifies the number of bytes of data space that the file occupies. This is designated in Figure A-2 by the shaded area. As you can see, the file does not occupy all the space allocated for it, and so the TOTAL\$SIZE value (8000) is not as large as the THIS\$SIZE value.

LONG FILES

If the file consists of more than eight distinct blocks of data, its fnode must specify it as a long file. The fnode for a long file has bit 1 of the FLAGS field set to one. This tells the I/O System that the PTR structures of the fnode identify indirect blocks. The indirect blocks identify the actual data blocks that make up the file.

Each indirect block contains of a number of indirect pointers, which are structures similar to the PTR structures. However, an indirect block can contain more than eight structures and thus can point to more than eight data blocks. In fact, an indirect block can consist of more than one volume block; however, all volume blocks of of an indirect block must be contiguous. The structure of each indirect pointer is as follows:

```
DECLARE
    IND$PTR  STRUCTURE(
        NBLOCKS  BYTE,
        BLK$PTR  BLOCK$NUM);
```

where:

NBLOCKS	Number of volume blocks in the data block.
BLK\$PTR	A 24-bit volume block number of first volume block in the data block. Volume blocks are numbered sequentially throughout the volume, starting with the block with the smallest address (block 0).

The IRMX 86 Operating System determines how many indirect pointers there are in an indirect block by comparing the NBLOCKS fields of the indirect pointers with the NUM\$BLOCKS field of the fnode. It assumes that the indirect block contains as many pointers as necessary for the sum of the NBLOCKS fields to equal the NUM\$BLOCKS field.

Figure A-3 illustrates an fnode for a long file. Decimal numbers are used in the figure for clarity.

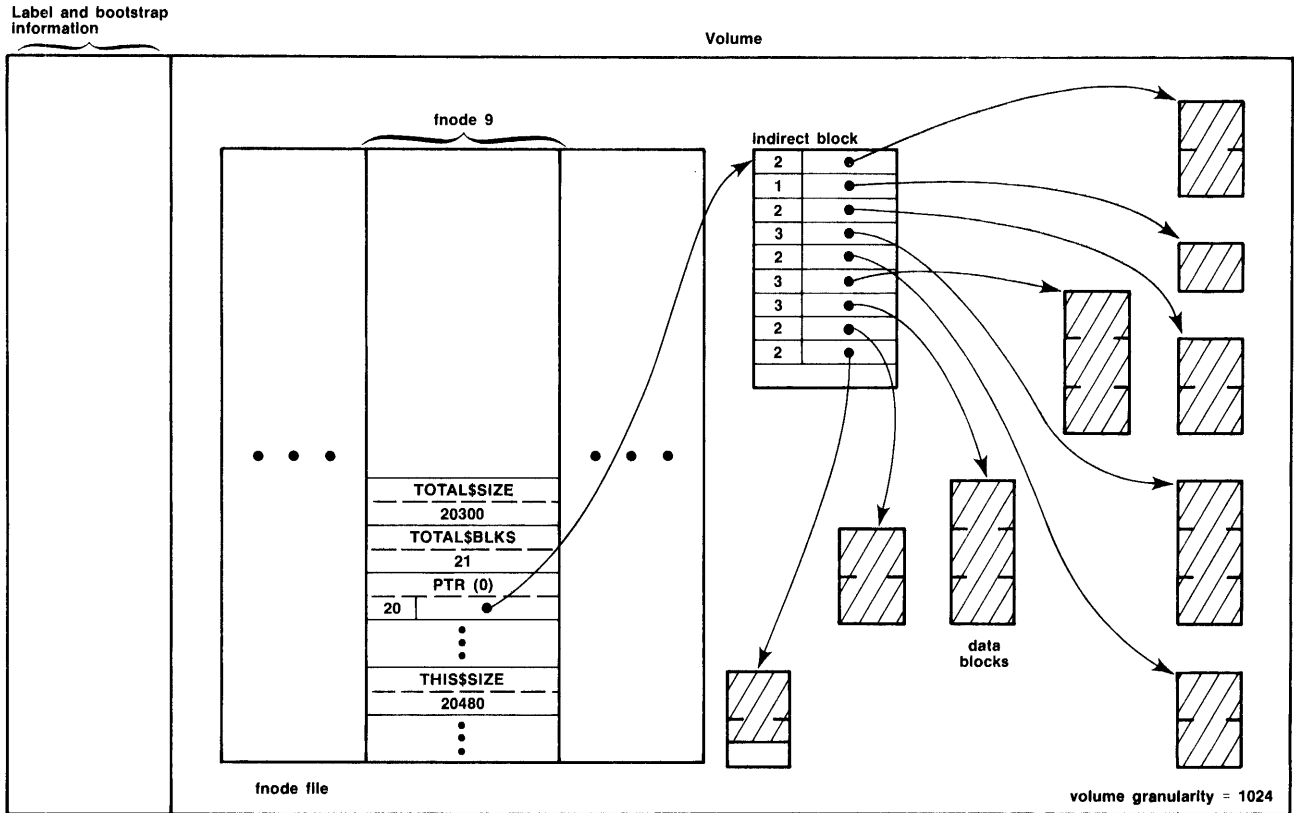


Figure A-3. Long File Fnode

As you can see from Figure A-3, fnode 9 identifies the long file. The actual file consists of nine distinct data blocks. One PTR structure and an indirect block give the locations of the data blocks. The NUM\$BLOCKS field of the PTR structure contains the number of volume blocks pointed to by the indirect block. The BLK\$PTR field points to the first volume block of the indirect block.

In the indirect block, each NBLOCKS field gives the length of an individual data block and each BLK\$PTR field points to the first volume block of a data block.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

Figure A-3 also lists the TOTAL\$BLKS, THIS\$SIZE, and TOTAL\$SIZE values, which are more complex than for a short file. The TOTAL\$BLKS field specifies the number of volume blocks allocated to the file, which in this case is 21. Twenty of the volume blocks are used for actual data storage and one of the blocks is used for the indirect block.

The THIS\$SIZE field specifies the number of bytes of data space allocated to the file, and does not include the size of the indirect block. This size is equal to the NUM\$BLOCKS value (20) or the sum of NBLOCKS values in the indirect block (2 + 1 + 2 + 3 + 2 + 3 + 3 + 2 + 2 = 20) multiplied by the volume granularity (1024) and equals 20480.

The TOTAL\$SIZE field specifies the number of bytes of data space that the file currently occupies. This is designated in Figure A-3 by the shaded areas. As you can see, the file does not occupy all the space allocated for it, and so the TOTAL\$SIZE value (20300) is not as large as the THIS\$SIZE value.

FLEXIBLE DISKETTE FORMATS

The flexible diskette device drivers supplied with the iRMX 86 Basic I/O System can support several diskette characteristics. Tables A-1 and A-2 list these characteristics.

Table A-1. Eight-Inch Diskette Characteristics

Sector Size	Density	Sectors per Track	Device Size (in bytes)	
			One sided	Two Sided
128	Single	26	256256	512512
256	Single	15	295168	590848
512	Single	8	314880	630272
1024	Single	4	315392	630784
256	Double	26	509184	1021696
512	Double	15	587264	1177600
1024	Double	8	626688	1255424

STRUCTURE OF iRMX 86™ NAMED VOLUMES

Table A-2. 5 1/4-Inch Diskette Characteristics

Sector Size	Density	Sectors per Track	Device Size (in bytes)			
			One Sided		Two Sided	
			35 Tracks	80 Tracks	35 Tracks	80 Tracks
128	Single	16	71680	163840	143360	327680
256	Single	9	80384	184064	161024	368384
512	Single	4	71680	163840	143360	327680
1024	Single	2	71680	163840	143360	327680
256	Double	16	141312	325632	284672	653312
512	Double	8	158720	366080	284672	653312
1024	Double	4	141312	325632	284672	653312

For compatibility with ECMA (European Computer Manufacturers Association) and ISO (International Organization for Standardization), the iRMX 86 device drivers, when called by the formatting tools (the FORMAT Human Interface command and the FORMAT Files Utility command), format the beginning tracks of all flexible diskettes in the same manner, as follows:

- For all 5 1/4-inch and 8-inch flexible diskettes, they format track 0 of side 0 with single-density, 128-byte sectors, with an interleave factor of 1.
- In addition, for 8-inch, double-sided, double-density flexible diskettes, they format track 0 of side 1 with double-density, 256-byte sectors.

The iRMX 86 device drivers map the sectors on these beginning tracks into blocks of device granularity size so that the Basic I/O System and Bootstrap Loader can treat flexible diskettes as if they contain a contiguous string of blocks, all of the same size.

However, this mapping is not exact when you use 8-inch, double-sided, double-density diskettes and specify a device granularity of 512 or 1024. A problem arises because there are 26 128-byte sectors in a track, which is not an integral mapping for device granularities of 512 or 1024. Thus the device driver combines the left-over 128-byte sectors of track 0, side 0 with the first sectors of track 0, side 1 in order to make a block of device granularity size. This continues throughout track 0, side 1, but the same problem occurs with the last 256-byte sectors of track 0, side 1; there are not enough sectors to make a block of device granularity size. When the device driver tries to combine these left-over sectors of track 0, side 1 with the first sectors of track 1, side 0, it finds that the sectors of track 1, side 0 are already of device granularity size. Therefore, since the device driver cannot access partial sectors, it is left with one block (the left-over sectors of track 0, side 1) that is less than device granularity size. When the device granularity is 512, this small block is block 19; when the device granularity is 1024, the small block is block 9.

STRUCTURE OF iRMX 86™ NAMED VOLUMES

If nothing is done to exclude this smaller-than-normal block from use, the device driver will treat this block as a normal block, assuming it is of device granularity size. Thus if you try to write information to that block, the driver will attempt to write an entire device granularity block of information into a block that is much shorter, causing you to lose information.

To prevent this situation, the Human Interface FORMAT command automatically declares this smaller-than-normal block as allocated in the volume free space map when it formats the volume. This prevents the Basic I/O System from ever writing information into this block. If you write your own formatting utility, you should also declare this block as allocated.

EXAMPLE VOLUME

This section lists the labels, fnode file, volume free space map file, free fnode map file, and root directory of a single density diskette which has been formatted by using the Human Interface FORMAT command with default parameters. Refer to the iRMX 86 HUMAN INTERFACE REFERENCE MANUAL for further information about the FORMAT command. This volume also contains one additional file whose fnode is shown.

ISO VOLUME LABEL

The following lists the individual fields of the ISO Label. Each two-digit number represents one byte, and thus one ASCII character. This label begins with byte number 768 of the diskette.

<u>field</u>	<u>value (hex)</u>	<u>ASCII equivalent</u>
LABEL\$ID(3)	56 4F 4C	VOL
LABEL\$NO	31	1
VOL\$NAME(6)	20 20 20 20 20 20	(spaces)
VOL\$STRUC	4E	N
RESERVED\$A(60)	20 (60 times)	(spaces)
REC\$SIDE	31	1
RESERVED\$B(4)	20 (four times)	(spaces)

STRUCTURE OF iRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>ASCII equivalent</u>
ILEAVE(2)	31 30	10
RESERVED\$C	20	(space)
ISO\$VERSION	31	1
RESERVED\$D(48)	20 (48 times)	(spaces)

iRMX 86 VOLUME LABEL

The following lists the individual fields of the iRMX 86 Volume Label. This label begins with byte 384 of the diskette. Following this listing, the individual fields are shown.

<u>field</u>	<u>value</u>	<u>ASCII or decimal equivalent</u>
VOL\$NAME(10)	45 58 41 4D 50 4C 45 00 00 00	EXAMPLE
FLAGS	01	
FILE\$DRIVER	04	4
VOL\$GRAN	0080	128
VOL\$SIZE	E900 0003	256256
MAX\$FNODE	0064	100
FNODE\$START	0D00 0000	3328
FNODE\$SIZE	005A	90
ROOT\$FNODE	0005	5
DEV\$GRAN	0080	128
INTERLEAVE	000A	10
TRACK\$SKEW	0000	0
SYSTEM\$ID	0000	0
SYSTEM\$NAME	69 52 4D 58 20 38 36 20 20 20 20 20	iRMX 86
DEVICE\$SPECIAL	00 00 00 00 00 00 00 00	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

FNODE FILE

The following lists the individual fields of the fnodes in the fnode file. Included are fnodes for the fnode file, the free space map file, the free fnodes map file, the accounting file, the bad blocks file, the root directory, and the example file. The fnode file begins at byte number 3328 decimal (0D00H) of the diskette, as shown in the IRMX 86 Volume Label.

Fnode 0 (Fnode File)

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0005	
TYPE	00	0 (FT\$FNODE)
GRAN	01	1
OWNER	0000	0000
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	2328 0000	9000
TOTAL\$BLKS	0047 0000	71
POINTR(40)		
PTR(0)		
NUM\$BLOCKS	0047	71
BLK\$PTR(0) - BLK\$PTR(2)	1A 00 00	26
PTR(1) - PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	2380 0000	9088
RESERVED\$A	0000	0
RESERVED\$B	0000	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
ID\$COUNT	0000	0
ACC(9)		
ACCESSOR(0) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0000	
AUX(*)	00 00 00	

Fnode 1 (Free Space Map)

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0005	
TYPE	01	1 (FT\$VOLMAP)
GRAN	01	1
OWNER	0000	0000
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	00FB 0000	251
TOTAL\$BLKS	0002 0000	2
POINTR(40)		
PTR(0)		
NUM\$BLOCKS	0002	2
BLK\$PTR(0) - BLK\$PTR(2)	61 00 00	97
PTR(1) - PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
THIS\$SIZE	0100 0000	256
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0000	0
ACC(9)		
ACCESSOR(0) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0000	
AUX(*)	00 00 00	

Fnode 2 (Free Fnode Map)

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0005	
TYPE	02	1 (FT\$FNODEMAP)
GRAN	01	1
OWNER	0000	0000
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	000D 0000	13
TOTAL\$BLKS	0001 0000	1
POINTR(40)		
PTR(0)		
NUM\$BLOCKS	0001	1
BLK\$PTR(0) - BLK\$PTR(2)	63 00 00	99

STRUCTURE OF iRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
PTR(1) - PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	0080 0000	128
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0000	0
ACC(9)		
ACCESSOR(0) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0000	
AUX(*)	00 00 00	

Fnode 3 (Accounting File)

No space for this file is allocated on the volume. However, its fnode must appear in the fnode file.

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0005	
TYPE	03	3 (FT\$ACCOUNT)
GRAN	01	1
OWNER	0000	0000
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	0000 0000	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
TOTAL\$BLKS	0000 0000	0
POINTR(40)		
PTR(0)-PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	0000 0000	0
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0000	0
ACC(9)		
ACCESSOR(0) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0000	
AUX(*)	00 00 00	

Fnode 4 (Bad Blocks File)

No space for this file is allocated on the volume. However, its fnode must appear in the fnode file.

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0005	
TYPE	04	3 (FT\$BADBLOCK)
GRAN	01	1
OWNER	0000	0000
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	0000 0000	0
TOTAL\$BLKS	0000 0000	0
POINTR(40)		
PTR(0)-PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	0000 0000	0
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0000	0
ACC(9)		
ACCESSOR(0) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0000	
AUX(*)	00 00 00	

Fnode 5 (Root Directory)

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
FLAGS	0025	
TYPE	06	1 (FT\$DIR)
GRAN	01	1
OWNER	FFFF	(WORLD)
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value (hex)</u>	<u>decimal equivalent</u>
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	0010 0000	16
TOTAL\$BLKS	0001 0000	1
POINTR(40)		
PTR(0)		
NUM\$BLOCKS	0001	1
BLK\$PTR(0) - BLK\$PTR(2)	70 00 00	112
PTR(1) - PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	0080 0000	128
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0001	1
ACC(9)		
ACCESSOR(0)		
ACCESS	FF	
ID	FFFF	(WORLD)
ACCESSOR(1) - ACCESSOR(2)		
ACCESS	FF	
ID	0000	
PARENT	0005	
AUX(*)	00 00 00	

STRUCTURE OF iRMX 86™ NAMED VOLUMES

Fnode 6 (Example File)

<u>field</u>	<u>value</u>	<u>decimal equivalent</u>
FLAGS	0025	
TYPE	08	8 (FT\$DATA)
GRAN	01	1
OWNER	FFFF	(WORLD)
CR\$TIME	0000 0000	0
ACCESS\$TIME	0000 0000	0
MOD\$TIME	0000 0000	0
TOTAL\$SIZE	01F4 0000	500
TOTAL\$BLKS	0004 0000	4
POINTR(40)		
PTR(0)		
NUM\$BLOCKS	0004	4
BLK\$PTR(0) - BLK\$PTR(2)	80 00 00	128
PTR(1) - PTR(7)		
NUM\$BLOCKS	0000	0
BLK\$PTR(0) - BLK\$PTR(2)	00 00 00	0
THIS\$SIZE	0200 0000	512
RESERVED\$A	0000	0
RESERVED\$B	0000	0
ID\$COUNT	0001	1
ACC(9)		
ACCESSOR(0)		
ACCESS	0F	
ID	FFFF	(WORLD)

STRUCTURE OF IRMX 86™ NAMED VOLUMES

<u>field</u>	<u>value</u>	<u>decimal equivalent</u>
ACCESSOR(1) - ACCESSOR(2)		
ACCESS	00	
ID	0000	
PARENT	0005	
AUX(*)	00 00 00	

FREE SPACE MAP FILE

The following is a listing of the free space map file. This file starts at byte 12416 of the volume (volume block 61H).

byte

```

12416  0000 0000 0000 0000 0000 0000 FFF0 FFFE
12432  FFF0 FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12448  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12464  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12480  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12496  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12512  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12528  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12544  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12560  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12576  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12592  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12608  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12624  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12640  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
12656  FFFF FFFF FFFF FFFF FFFF 0003 0000 0000
    
```

STRUCTURE OF IRMX 86™ NAMED VOLUMES

FREE FNODES MAP FILE

The following is a listing of the free fnodes map file. This file starts at byte 12672 of the volume (volume block 63H).

<u>byte</u>									
12672	FF80	FFFF	FFFF	FFFF	FFFF	FFFF	000F	0000	
12688	0000	0000	0000	0000	0000	0000	0000	0000	
12704	0000	0000	0000	0000	0000	0000	0000	0000	
12720	0000	0000	0000	0000	0000	0000	0000	0000	
12736	0000	0000	0000	0000	0000	0000	0000	0000	
12752	0000	0000	0000	0000	0000	0000	0000	0000	
12768	0000	0000	0000	0000	0000	0000	0000	0000	
12784	0000	0000	0000	0000	0000	0000	0000	0000	

ROOT DIRECTORY

The following is a listing of the root directory. This file starts at byte 14336 of the volume (volume block 70H).

<u>byte</u>																
14336	06	00	45	58	41	4D	50	4C	45	2E	46	49	4C	45	00	00
14352	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14368	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14384	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14400	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14416	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14432	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5
14448	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5	E5

INDEX

Underscored entries are primary references.

abbreviations 2-1
access rights 2-16
ADD command 2-25
ADDRESS command 2-25
ALLOCATE command 2-5
allocation status 2-15, A-9
attributes 2-7
automatic device recognition A-4
auxiliary bytes A-13

bad blocks file A-15, A-28
BLOCK command 2-26
buffer 2-9, 2-18, 2-31, 2-34, 2-37, 2-48

command dictionary 2-4
commands 2-1

DEC command 2-27
density A-4
device granularity A-5
device recognition A-4
dictionary 2-4
directory 2-12, A-16
DISK command 2-7
DISKVERIFY
 command 1-2
 output 1-4
DISPLAYBYTE command 2-9
DISPLAYDIRECTORY command 2-12
DISPLAYFNODE command 2-14
DISPLAYWORD command 2-18
DIV command 2-27

example volume A-22
EXIT command 2-21

file
 driver A-5
 granularity 2-15, A-10
 owner 2-15, A-11
 type 2-15, A-10
fnode file A-8, A-14, A-24
fnodes 2-6, 2-14, 2-22, 2-32, 2-40, A-5, A-7
FREE command 2-22
free fnodes map 2-32, 2-41, A-15, A-26, A-33
free space map 2-32, 2-41, A-14, A-25, A-32

INDEX (continued)

granularity 2-15, A-5, A-10

HELP command 2-24
HEX command 2-27

initial files A-8
input radices 2-2
interleave factor A-3, A-5
invocation 1-2
iRMX 86 volume label A-4, A-23
ISO label A-2, A-22

long files 2-15, A-9, A-12, A-18

miscellaneous commands 2-25
 ADD 2-25
 ADDRESS 2-25
 BLOCK 2-26
 DEC 2-27
 DIV 2-27
 HEX 2-27
 MOD 2-28
 MUL 2-28
 SUB 2-29
MOD command 2-28
MUL command 2-28

NAMED verification 1-3, 2-41
named volume structure A-1
NAMED1 verification 1-3, 2-40
NAMED2 verification 1-3, 2-40

owner 2-15, A-11

parameters 2-1
parent directory A-13
PHYSICAL verification 1-3, 2-41

QUIT command 2-30

radices 2-2
READ command 2-31
recording
 density A-4
 sides A-5
 size A-5
root directory A-5, A-16, A-29, A-33

SAVE command 2-32
short files 2-15, A-9, A-12, A-17
size A-5
structure of iRMX 86 named volumes A-1
SUB command 2-29
SUBSTITUTEBYTE command 2-34
SUBSTITUTEWORD command 2-37

INDEX (continued)

track skew A-6

VERIFY 1-3, 2-32, 2-40

volume

blocks 2-5, 2-22, 2-31

free space map 2-32, 2-40, A-14, A-25, A-32

granularity A-5

labels A-2, A-4

name A-3, A-4

size A-5

working buffer 2-9, 2-18, 2-31, 2-34, 2-37, 2-48

WRITE command 2-48



REQUEST FOR READER'S COMMENTS

Intel Corporation attempts to provide documents that meet the needs of all Intel product users. This form lets you participate directly in the documentation process.

Please restrict your comments to the usability, accuracy, readability, organization, and completeness of this document.

1. Please specify by page any errors you found in this manual.

2. Does the document cover the information you expected or required? Please make suggestions for improvement.

3. Is this the right type of document for your needs? Is it at the right level? What other types of documents are needed?

4. Did you have any difficulty understanding descriptions or wording? Where?

5. Please rate this document on a scale of 1 to 10 with 10 being the best rating. _____

NAME _____ DATE _____

TITLE _____

COMPANY NAME/DEPARTMENT _____

ADDRESS _____

CITY _____ STATE _____ ZIP CODE _____

Please check here if you require a written reply.

WE'D LIKE YOUR COMMENTS . . .

This document is one of a series describing Intel products. Your comments on the back of this form will help us produce better manuals. Each reply will be carefully reviewed by the responsible person. All comments and suggestions become the property of Intel Corporation.



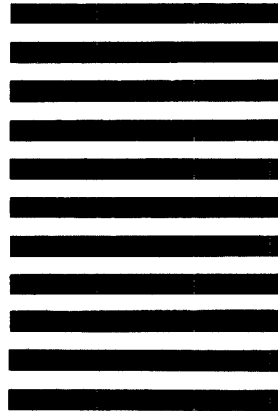
NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 79 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Intel Corporation
5200 N.E. Elam Young Pkwy.
Hillsboro, Oregon 97123

O.M.S. Technical Publications





INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 (408) 987-8080

Printed in U.S.A.