FORMAT Command and Disk Verification Utility Enhancements

iRMX™ 86 Release 6.Ø, Update 1

Part Number: 147155-ØØ1

CONTENTS

Introduction

Update 1 offers two new iRMX 86 capabilities--the capability of backing up and restoring volume fnodes, and the capability of adding the second stage of the Bootstrap Loader to a volume without re-formatting the volume. These features have been implemented by adding two switches to the FORMAT system command and by creating four new Disk Verification Utility commands.

The fnode backup and restore feature offers a minimum level of protection for the volume label and the fnode file of a named volume. This feature is not intended to provide extensive fnode backup capabilities for the volume. However, it does offer a limited mechanism for attempting to recover data when the volume label or the fnode file has been damaged.

The Bootstrap Loader feature allows you to take advantage of the Release 6.Ø Bootstrap Loader capabilities (such as the "debug switch") without having to back up and re-format volumes created under earlier releases of the iRMX 86 Operating System.

This write up is intended to serve as stand alone documentation for these enhancements. It is available only through the iRMX 86 Release 6.Ø update service. The write-up is not intended to replace pages in the existing documentation. However, it can be included in the same binders as the rest of the documentation. For instance, the pages in the Reference section can easily be inserted in the appropriate section of the iRMX 86 DISK VERIFICATION REFERENCE MANUAL or the iRMX 86 OPERATOR'S MANUAL.

In this write-up, Section One is devoted to the fnode backup and restore feature. Section Two covers the Bootstrap Loader feature. Within each section a general overview of the feature is presented, followed by a brief explanation of how that feature is used. In addition, Section 3 of the write-up contains a reference section that describes the modified FORMAT system command, the four new Disk Verification Utility commands, and two existing Disk Verification Utility commands (DISK and HELP) that have been changed to reflect the addition of the new commands.

Section One

Backing Up and Restoring Fnodes

1.1 Overview

To access data on a named volume (such as a disk), the iRMX 86 Operating System uses a mechanism common to virtually all operating systems. This approach entails maintaining an index to every file on the disk. The index is created when the disk is formatted and thereafter it remains as a permanent structure at a dedicated location on the disk. The index consists of a system of pointers that indicate the location of the data files on the disk. Thus, when data must be stored or retrieved from the disk, the operating system can find the exact location of the appropriate file by looking up the filename in the index.

In the iRMX 86 Operating System, the index consists of a volume label and an fnode file. The volume label resides at the same location in all devices and serves as the initial entry point into the device. The fnode file contains a series of individual structures called file descriptor nodes or "fnodes". There is one fnode for each file on the disk. The fnode contains information essential to accessing and maintaining the respective file.

The iRMX 86 file structure for a named volume is organized as a hierarchical tree. That is, there is a root directory with branches to other directories and ultimately, to files. The organization of the fnode file reflects this hierarchical structure. The volume label contains a pointer to the fnode of the file structure's root directory. The root directory points to blocks on the volume. These blocks may represent a first level data file or a first level directory file.

A block that represents a data file contains data. A block that represents a directory file contains a data structure which provides the names of all of that directory's files and identifies the fnodes associated with those files.

The Operating System creates the volume label and the fnode file when the disk is formatted. The number of un-allocated fnodes in the file is controlled by the FILES parameter of the FORMAT command. In addition to the un-allocated fnodes, seven allocated fnodes are established when the fnode file is created. These allocated fnodes represent the fnode file, the volume free space map file, the free fnodes map file, the bad blocks file, the volume label file, the root directory, and an accounting file. (For a full description of these files, see the Disk Verification Utility section of the "iRMX 86 Introduction and Operator's Reference Manual for Release 6".)

Thereafter, when files or directories are created directly subordinate to the root, the Operating System must adjust a pointer in the root fnode to indicate the fnode number of the new data file or directory file. Subsequently, directories subordinate to the root must also have their pointers adjusted when they become parents to a new data file or directory.

This method of storing and retrieving data on a disk has one major drawback. All access to files on the disk is through the volume label and the fnode file. If either the volume label file or the fnode file are damaged or destroyed, there is no practical way to recover data on the disk.

The backup and restore fnodes feature provides a means of attempting to recover data lost as a result of catastrophic damage to the fnode file or the volume label. This feature allows you to create a backup version of the volume label and all the fnodes on the disk. The backup version is stored in one of the innermost tracks of the disk where the chance of accidental loss of data is minimal. (In normal usage, the disk heads do not extend to the innermost tracks.)

To implement this feature, Intel has modified the Human Interface FORMAT system command. A new optional parameter--RESERVE--has been added to the command. This version of the FORMAT command creates a new file named R?SAVE in the innermost track of the volume. A copy of the volume label is placed in the front of the file (that is, at the physical end of the volume) and an fnode is allocated for R?SAVE in the fnode file. (The fnode for the R?SAVE file is allocated out of the fnodes reserved through the FILES parameter of the FORMAT command. Thus, if you specify "FILES = 3ØØØ" when you format, only 2999 of those fnodes will remain available after the R?SAVE fnode has been allocated.) Finally, FORMAT copies the fnode file into R?SAVE.

Notice that the new FORMAT command creates a backup of the fnode file in its initialized state. R?SAVE is not subsequently updated as files are written to or deleted from the volume. Therefore, you will have to use the new BACKUPFNODES Disk Verification Utility command to backup the fnode file when the volume has been modified. If the volume label or the fnode file become damaged, you can attempt to recover files on the volume by using the new Disk Verification Utility commands (RESTOREFNODES and RESTOREVOLUMELABEL) to rebuild the index. To assist in this process, the new DISPLAYSAVEFNODE Disk Verification Utility command allows you to look at individual fnodes stored in the R?SAVE file.

Since the contents of the volume label do not change, the copy of the volume label in R?SAVE is always valid. Therefore, you can restore the volume label at any time regardless of when the R?SAVE file was last updated. (When the Disk Verification Utility encounters a damaged volume label, it now automatically uses the backup volume label if the R?SAVE file is present.)

One note of caution: The fnode file is changed each time a volume is modified (that is each time a file is created, written to, or deleted from the volume). Therefore, valid restoration can be assured only if the fnode file is backed up each time the volume is modified. If the fnodes are not backed up after each modification, the structure of the R?SAVE file will differ from that of the fnode file. That is, some fnodes in R?SAVE may not be associated with the same files as the corresponding fnodes in the fnode file. Attempting to recover fnodes under these conditions is dangerous because the RESTOREFNODES command will overwrite what may be a valid fnode in the fnode file.

While the backup and restore fnodes feature is a useful aid in attempting to recover data on a volume, this capability is limited in scope. If you are troubleshooting your system, you may want to back up the fnodes on the system disk before taking any action that may risk the disk's integrity. You may also decide to back up the fnodes on a routine basis (before each system shutdown, for instance) so that the R?SAVE file is always relatively current. However, under normal circumstances, where a volume is accessed and modified frequently, backing up the fnodes after each modification is not practical.

Therefore, the limited capability of the fnode backup and restore feature must be clearly understood. This feature is not intended to provide comprehensive protection from the loss of data associated with damaged volume labels or fnode files. Rather, the purpose is to offer a tool that, when properly applied, can be useful in maintaining volume integrity in certain situations.

1.2 Using Fnode Backup and Restore

The fnode backup and restore feature requires the use of Version 3.1 of the Release 6.Ø Human Interface FORMAT command and the Version 3.1 of the Release 6.Ø Disk Verification Utility. Both of these items are distributed in the iRMX 86 Update Package (Update 1.Ø and later).

Used together, the new versions of the FORMAT command and the Disk Verification Utility allow you to (1) format a volume to create the backup file (R?SAVE), (2) backup the fnodes of any files that are written to the volume (3) examine the contents of the backup file (R?SAVE), (4) restore damaged fnodes, and (5) restore the volume label.

This section describes how to perform each of these operations. A brief overview of the operation is followed by one or more examples of a typical implementation. In the examples, boldface type (**this is boldface type**) is used to indicate an entry made from your terminal. Standard type (this is standard type) is used to indicate system output to your terminal.

This section is organized as follows:

1.2.1 Creating the R?SAVE Fnode Backup File

Description

If you intend to backup the volume label and the fnodes on a volume, you must first create the R?SAVE backup file on the innermost track of the volume. To do so you must invoke Release 6.Ø, Version 3.1 of the Human Interface FORMAT command, specifying the RESERVE option. NOTICE THAT THE FORMAT COMMAND OVERWRITES ALL OF THE DATA CURRENTLY ON THE DISK. Therefore, make a backup copy of any files you wish to save.

Once the volume has been formatted, the R?SAVE file will contain a copy of the fnode file including the allocated fnodes (R?SPACEMAP, R?FNODEMAP, etc.). Therefore, you need not backup the fnode file immediately after formatting the volume.

Procedure

1. From the Human Interface, invoke the FORMAT command, specifying the RESERVE parameter.

Example

Assume that you have booted your system from a floppy diskette in order to format the winchester disk. The disk is attached as logical device :se:. The command listed below formats the disk and creates the R?SAVE backup file. The initialized fnode file is copied into R?SAVE.

```
-format :se: il = 4 files = 3000 reserve <CR>

volume ( ) will be formatted as a NAMED volume
    granularity          =    1,Ø24     map start      =    7,859
    interleave           =      4
    files                =    3ØØØ
    extensionsize        =      3
    save area reserved   =    yes
    volume size          =    15,984K

TTTTTTTTTTTTTTTTT
volume formatted
```

The disk has now been formatted. A file named R?SAVE has been reserved in one of the innermost tracks of the disk. (If you use the Disk Verification Utility DISPLAYDIRECTORY command on the volume root fnode (fnode 6), you will find an fnode listed for R?SAVE.) R?SAVE contains a duplicate copy of the fnodes in the fnode file. That is, R?SAVE contains eight allocated fnodes (R?SAVE, R?SPACEMAP, R?FNODEMAP, etc.) and 2,999 un-allocated fnodes. (Remember, the R?SAVE fnode is allocated out of the 3,ØØØ fnodes specified through the FILES parameter.)

1.2.2 Backing Up Fnodes on a Volume

Description

To back up the fnodes on a volume, you must have previously reserved the back up file R?SAVE when the volume was formatted. Thereafter, any modification to the volume (creating, writing to, or deleting a file) requires that the fnodes be backed up if the R?SAVE file is to contain an accurate copy of the fnode file.

To back up the fnode file on a volume, you must invoke the Disk Verification Utility using the logical device name of the volume to be backed up. When the Disk Verification Utility is activated, enter the Disk Verification Utility BACKUPFNODES command. A duplicate copy of all the fnodes in the fnode file will be written to the R?SAVE file.

Procedure

1. Invoke the Disk Verification Utility using the logical device name of the device containing the volume to be backed up.

2. When you receive the Disk Verification Utility prompt (*), enter the BACKUPFNODES command.

3. When the fnodes have been backed up, the Disk Verification Utility returns the message "fnode file backed up to save area".

Example

Assume that the system disk is attached as logical device :sd:. The initial contents of the :sd: fnode file were copied to R?SAVE by the FORMAT command. A file has just been written to the volume. An fnode for this file is entered in the fnode file. However, no corresponding entry has been made in R?SAVE. The following sequence of commands will copy all fnodes in the fnode file into the R?SAVE file.

```
-diskverify :sd: <CR>
iRMX 86 Disk Verifiy Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*backupfnodes <CR>     or     bf <CR>
fnode file backed up to save area
*
```

R?SAVE now contains a duplicate copy of all fnodes (allocated and un-allocated) in the fnode file.

1.2.3 Backing Up the Volume Label

Description

The volume label is initially copied to R?SAVE when the volume is formatted. Since the contents of the volume label do not change, there are no other volume label backup procedures required.

1.2.4 Restoring Fnodes

Description

To restore fnodes on a volume, you must have previously reserved the backup file R?SAVE when the volume was formatted. If damage has occured to the fnode file, you can attempt to rebuild the file (or portions of it) by using the Disk Verification Utility RESTOREFNODE command.

RESTOREFNODE allows you to restore a single fnode or a range of fnodes. You designate the fnodes to be restored by entering the fnode numbers. The specified fnodes in R?SAVE are copied into the corresponding fnodes in the fnode file.

Prior to restoring each fnode, RESTOREFNODE prompts you with the message "restore fnode <fnode number>?". To restore the fnode, you must enter a letter "y" (either Y or y). If you enter any other response, the fnode will not be restored.

When restoring fnodes, you must be very careful to assure that you are not overwriting a valid fnode in the fnode file with an invalid fnode from R?SAVE. You are assured that this will not happen only if the volume has not been modified since the fnodes were last backed up.

Procedure

1. Invoke the Disk Verification Utility, using the logical device name of the volume to be backed up.

2. When you receive the Disk Verification Utility prompt (*), enter the appropriate Disk Verification Utility commands (VERIFY, DISPLAYFNODE, etc.) to examine the fnodes file and determine which fnode must be restored.

3. Invoke the Disk Verification Utility RESTOREFNODE command to replace the damaged fnodes. The Disk Verification Utility prompts you to confirm that the proper fnode is being restored. Check to assure that you have specified the correct hexadecimal number for the fnode, then enter the letter "y" in response to the prompt.

4. RESTOREFNODE returns the message "restored fnode < fnode number >" after the fnode in the R?SAVE file has been written over the corresponding fnode in the fnode file.

Example 1

Assume that a disk drive is attached as logical device :sd:. The volume :sd: contains the R?SAVE fnode backup file. You have not modified the disk since the fnodes were last backed up. Subsequently, you have reason to suspect that the fnode file has been damaged. You use the Disk Verification Utility utility to confirm your suspicions:

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*verify
     .
     .
     .
```

After using the Disk Verification Utility to examine the structure of the disk, you find that fnodes 9 through ØC have probably been destroyed. You decide to use the RESTOREFNODE command to recover these fnodes.

```
*restorefnode 9, 0C <CR>     or    rf 9, 0C <CR>
restore fnode      9?  Y <CR>
restored fnode number:    9
restore fnode     ØA?  Y <CR>
restored fnode number:   ØA
restore fnode     ØB?  Y <CR>
restored fnode number:   ØB
restore fnode     ØC?  Y <CR>
restored fnode number:   ØC
```

Fnodes Ø9 through ØC in the R?SAVE file have been copied into fnode Ø9 through ØC in the fnode file. Since the disk has not been modified since the last fnode backup, restoring the damaged fnodes should now enable you to recover the data on the disk.

Example 2

Assume the same initial conditions as example 1 with the following exception: two files have been modified since the last time the fnodes were backed up. In the fnode file, the new files are represented by fnodes ØD and ØE. Again, you suspect that the fnode file has been damaged. You use the Disk Verification Utility to check the condition of data on the disk:

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*verify
     .
     .
     .
```

After using the Disk Verification Utility to examine the structure of the disk, you find that fnodes 9 through 1Ø have probably been destroyed. You decide to use the RESTOREFNODE command to recover these fnodes. You do not wish to restore fnodes ØD and ØE because these fnodes were not backed up. Since the data fields of fnodes ØD and ØE in R?SAVE contain all zeros, you would be destroying possibly useful data in the corresponding fnodes. You decide to use RESTOREFNODE to restore a range of fnodes that includes ØD and ØE. However, you intend to pass over the restoration of these two fnodes by responding to the confirmation prompt with some character other than 'y'.

```
*restorefnode 9,10 <CR>     or    rf 9,10 <CR>
restore fnode      9?  Y <CR>
restored fnode number:   9
restore fnode     ØA?  Y <CR>
restored fnode number:  ØA
restore fnode     ØB?  Y <CR>
restored fnode number:  ØB
restore fnode     ØC?  Y <CR>
restored fnode number:  ØC
```

Notice, that because fnodes ØD and ØE were never backed up, those fnodes in R?SAVE are un-allocated. Therefore the Disk Verification Utility returns the "allocation bit not set for saved fnode" message. Since you do not wish to restore this fnode, you respond to the confirmation prompt with a 'non-y' character.

```
allocation bit not set for saved fnode
restore fnode     ØD?   <CR>
allocation bit not set for saved fnode
restore fnode     ØE?  n <CR>
restore fnode     ØF?  Y <CR>
restored fnode number:  ØF
restore fnode     1Ø?  Y <CR>
restored fnode number:  1Ø
```

The R?SAVE fnodes Ø9 through ØC and fnodes ØF through 1Ø have been copied over the corresponding fnodes in the fnode file. Fnodes ØD and ØE were not restored.

1.2.5 Restoring the Volume Label

Description

To restore the volume label, you must have previously reserved the backup file R?SAVE when you formatted the volume. If the volume contains the R?SAVE file, a backup copy of the volume label already exists. The FORMAT command automatically places a copy of the volume label into R?SAVE when the file is created. Thereafter, the contents of the volume label do not change. Therefore, you can restore the label without fear of destroying data in the existing label.

To restore the volume label, you must invoke the Disk Verification Utility using the logical device name of the appropriate volume. If the volume label is corrupted, the Disk Verification Utility attempts to use the backup copy of the volume label in R?SAVE. When the backup label is used, the Disk Verification Utility issues a message that reads "DUPLICATE VOLUME LABEL USED". If this message appears when the Disk Verification Utility is activated, then the volume label is damaged. To restore the volume label, enter the Disk Verification Utility RESTOREVOLUMELABEL command. The current volume label will be overwritten with the volume label copy from R?SAVE.

Procedure

1. Invoke the Disk Verification Utility, using the logical device name of the volume to be backed up.

2. If the "DUPLICATE VOLUME LABEL USED" message appears when the utility is activated, the volume label must be restored. Enter the Disk Verification Utility RESTOREVOLUMELABEL command.

3. When the volume label has been restored, the Disk Verification Utility returns the message "volume label restored".

Example

Assume that a disk drive is attached as logical device :sd:. The volume :sd: contains the R?SAVE fnode backup file. When you attempt to access files on :sd:, the system returns and E$ILLEGAL_VOLUME message. You suspect that the volume label may be damaged. You decide to check your suspicions using the Disk Verification Utility.

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
DUPLICATE VOLUME LABEL USED
*
```

The "DUPLICATE VOLUME LABEL USED" message confirms that the volume label has been damaged. You restore the volume label using the RESTOREVOLUMELABEL command.

```
*restorevolumelabel <CR>     or    rvl <CR>     or     rv  <CR>
volume label restored
*
```

The original volume label has been overwritten with the duplicate copy from the R?SAVE file. Attempts to access files on volume :sd: should now be successful.

1.2.6 Displaying R?SAVE Fnodes

<u>Description</u>

Any fnode (both allocated and un-allocated) in the R?SAVE file can be examined by using the Disk Verification Utility DISPLAYSAVEFNODE command. The Disk Verification Utility will display vital information about the fnode (total blocks, total size, block pointers, parent node, etc.). The fnode will be displayed in the same format used by the Disk Verification Utility DISPLAYFNODE command.

To display an R?SAVE fnode, enter the Disk Verification Utility DISPLAYSAVEFNODE command and specify the hexadecimal number of the fnode to be displayed.

<u>Procedure</u>

1. Invoke the Disk Verification Utility using the logical device name of the appropriate volume.

2. When you receive the Disk Verification Utility prompt (*), enter the Disk Verification Utility DISPLAYSAVEFNODE command. Specify the hexadecimal number of the fnode to be displayed.

3. The Disk Verification Utility will return with an fnode display.

Example

Assume that you can not access a file on a disk attached as :sd:. You suspect that the fnode file may be damaged. By entering the Disk Verification Utility and displaying the file's directory, you find that the file you were unable to access is represented by fnode 3C8. You use DISPLAYFNODE to display fnode 3C8 but you are not confident of the data you see. Since the fnode for the file has been backed up since the file was last modified, you decide to use data in the R?SAVE fnode to examine the fnode file. The following command displays the data for fnode 3C8 in R?SAVE.

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
         .
         .
         .

*displaysavefnode 3C8 <CR>      or    dsf 3C8 <CR>


    Fnode number = 3C8 (saved)
                       flags : ØØ25  =>  short file
                        type : Ø8  =>  data file
          file gran/vol gran : Ø1
                       owner : ØØØ1
     create,access,mod times : ØØØØØØØØ, ØØØØØØØØ, ØØØØØØØØ
                  total size : ØØØØ2DØ1
                total blocks : ØØØØØØØC
            block pointer(1) : ØØØC, ØØ491Ø
            block pointer(2) : ØØØØ, ØØØØØØ
            block pointer(3) : ØØØØ, ØØØØØØ
            block pointer(4) : ØØØØ, ØØØØØØ
            block pointer(5) : ØØØØ, ØØØØØØ
            block pointer(6) : ØØØØ, ØØØØØØ
            block pointer(7) : ØØØØ, ØØØØØØ
            block pointer(8) : ØØØØ, ØØØØØØ
                   this size : ØØØØ3ØØØ
                    id count : ØØØ1
                 accessor(1) : ØF, ØØØ1
                 accessor(2) : ØØ, ØØØØ
                 accessor(3) : ØØ, ØØØØ
                      parent : Ø3C4
                     aux (*) : ØØØØØØ
*
```

Section Two

Adding the Second Stage of the New Bootstrap Loader to a Formatted Disk

2.1 Overview

The Bootstrap Loader operates in two stages. The first stage establishes a location in RAM for the second stage, names the load file, loads part of the second stage and transfers control to the second stage. The second stage finishes loading itself, transfers the load file into memory and passes control to the load file.

The first stage resides on ROM or is stored on a secondary storage device. The second stage always resides on track Ø of every (Standard format) named volume. The Human Interface FORMAT system command automatically places the second stage of the Bootstrap Loader in track Ø of every named volume it formats.

The iRMX 86 Release 6.Ø version of the Bootstrap Loader provides a new feature called the "Debug Switch". The Debug Switch allows you to use the monitor during the initialization of application jobs. Thus, you can single step through the initialization process, establishing breakpoints, examining register contents, etc.

In order to use this feature, you will need the first stage of the Release 6.Ø Bootstrap Loader in your system. You will also need the second stage of the Release 6.Ø Bootstrap Loader on track Ø of any named volumes used to load applications. To avoid forcing you to re-format application diskettes that do not contain the Release 6.Ø version, Intel has added a new parameter--the BOOTSTRAP parameter--to the FORMAT system command. When the BOOTSTRAP parameter is specified, FORMAT writes the second stage of the Release 6 Bootstrap Loader onto track Ø without re-formatting the rest of the volume.

2.2 Using the FORMAT Bootstrap Loader Switch

<u>Description</u>

To install the second stage of the Release 6.Ø Bootstrap Loader on a named volume, invoke the FORMAT command, indicating the logical name of the appropriate volume and specifying the BOOTSTRAP parameter. When the BOOTSTRAP parameter is specified, any other parameters entered with the command are disregarded. FORMAT writes the second stage of the Release 6.Ø Bootstrap Loader onto track Ø without re-formatting the volume.

Procedure

1. Invoke the FORMAT command using the logical device name of the volume to which the second stage of the Bootstrap Loader is to be added. Specify the BOOTSTRAP parameter by entering "B", "BS", or "BOOTSTRAP" on the same logical line as the FORMAT command. (Remember, if you fail to specify the BOOTSTRAP parameter, FORMAT will format the volume.)

Example

Assume that you have a number of diskettes formatted by the Release 5.Ø version of FORMAT command (V2.Ø). You plan to use the Release 6.Ø version of the Bootstrap Loader with some of the files on these diskettes. However, you do not want to undergo the time consuming process of copying all of these files onto newly formatted diskettes. Therefore, you are using the FORMAT command with the BOOTSTRAP switch set to copy the second stage of the Bootstrap Loader onto track Ø of the Release 5.Ø diskettes.

In this example, assume that the logical name :fØ: applies to a floppy disk drive containing a diskette formatted under iRMX 86 Release 5.Ø. On the diskette are files containing application programs. Any of the following commands will copy the second stage of the Release 6.Ø Bootstrap Loader onto track Ø of the diskette without re-formatting the volume.

```
-FORMAT :f0: BS <CR>
-FORMAT :f0: BOOTSTRAP <CR>
-FORMAT :f0: B <CR>
-FORMAT :f0: FILES= 300 GRANULARITY=200 FORCE BOOTSTRAP <CR>

Bootstrap Loader written
```

When the FORMAT command has completed executing, track Ø of the diskette contains the Release 6.Ø version of the Bootstrap Loader's second stage. The remainder of the files on the diskette are unaffected. (Notice, in the fourth example, the FILES, GRANULARITY, and FORCE switches are ignored since the BOOTSTRAP switch has precedence over any other FORMAT switch).

Section Three

Reference

The following section provides reference material on (1) the four new Disk Verification Utility commands, (2) the two modified Disk Verification Utility commands, and (3) the modified FORMAT command.

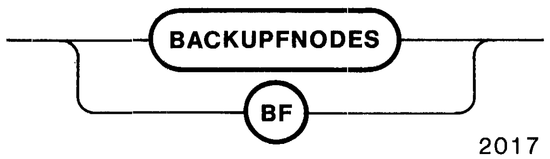
The section begins with the following Disk Verification Utility commands presented in alphabetical order:

BACKUPFNODES
DISK
DISPLAYSAVEFNODE
HELP
RESTOREFNODE
RESTOREVOLUMELABEL

The modified FORMAT command is at the end of the section.

BACKUPFNODES

This command copies the current fnode file into a designated fnode backup file named R?SAVE. R?SAVE must have been reserved when the volume was formatted. (That is, the RESERVE option of the FORMAT command must have been specified.) The format of the BACKUPFNODES command is as follows:



INPUT PARAMETERS

None.

OUTPUT

BACKUPFNODES displays the following message:

```
        fnode file backed up to save area
```

DESCRIPTION

The BACKUPFNODES command provides a means of avoiding the loss of data that occurs when the fnode file is damaged or destroyed. To use this command, you must have formatted the volume using Version 3.1 (or later) of the FORMAT command to create a special reserve area (R?SAVE). A switch in the FORMAT command (the RESERVE switch) controls the creation of R?SAVE. If you did not specify the RESERVE parameter when the volume was formatted, the BACKUPFNODES command will be unable to copy the fnode file to R?SAVE. An error message will be returned indicating that no save area has been reserved. (In this case, the volume must be reformatted if you wish to use the BACKUPFNODES command.)

The FORMAT system command writes the initialized copy of the fnode file into R?SAVE. Therefore you do not have to use BACKUPFNODES to back up a newly formatted volume. Subsequently, anytime a file is created, modified, or deleted, you will have to back up the fnodes to assure that the data in R?SAVE matches the data in the fnode file.

When the BACKUPFNODES command is invoked, all of the fnodes currently in the fnode file are copied to the R?SAVE file. Any fnodes currently saved in R?SAVE are overwritten.

ERROR MESSAGES

Message	Description
argument error	When you entered the command, you supplied an argument. BACKUPFNODES does not accept an argument.
no save area reserved when volume was formatted	The volume has not been formatted to support fnode backup. Re-format the volume using Version 3.1 (or later) of the FORMAT command with the RESERVE parameter specified.
not a named disk	The volume you specified when you invoked the Disk Verification Utility is a physical volume not a named volume.

EXAMPLE

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*backupfnodes <CR>     or     bf <CR>
fnode file backed up to save area
*
```

DISK

This command displays the attributes of the volume being verified. You can abort this command by typing a CONTROL-C (press the CONTROL key, and while holding it down, press the C key). The format of the DISK command is as follows:

2018

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 granularity = <devgran>
                block size = <volgran>
          number of blocks = <numblocks>
     number of free blocks = <numfreeblocks>
               volume size = <size>
                interleave = <inleave>
            extension size = <xsize>
          number of fnodes = <numfnodes>
     number of free fnodes = <numfreefnodes>
        save area reserved = (yes/no)
```

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

<volname>	Name of the volume, as specified when the volume was formatted.
<volgran>	Volume granularity, as specified when the volume was formatted.
<numfreeblocks>	Number of available volume blocks in the volume.
<inleave>	The interleave factor for a named volume.
<xsize>	Size, in bytes, of the extension data portion of each file descriptor node (fnode).
<numfnodes>	Number of fnodes in the volume. The fnodes were created when the volume was formatted.
<numfreefnodes>	Number of available fnodes in the named volume.
save area reserved	Indicates whether the R?SAVE file is reserved for volume label and fnode file backups.

Refer to THE iRMX DISK VERIFICATION UTILITY REFERENCE MANUAL, Appendix A or to the description of the FORMAT command in the iRMX 86 OPERATOR'S MANUAL for more information about the named disk fields.

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.

ERROR MESSAGES

None

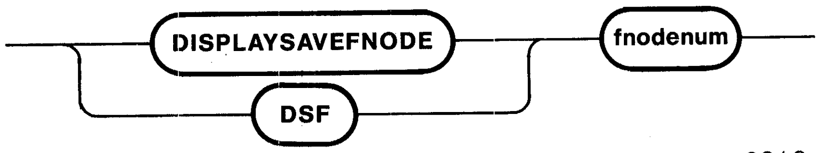
EXAMPLE

```
-diskverify :f0: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*disk

                   device name = wfdØ
         named disk, volume name =
              device granularity = Ø1ØØ
                      block size = Ø1ØØ
                number of blocks = ØØØØØ7C5
           number of free blocks = ØØØØØ6C3
                     volume size = ØØØ7C5ØØ
                      interleave = ØØØ5
                  extension size = Ø3
                number of fnodes = ØØCF
           number of free fnodes = ØØC2
              save area reserved = yes
```

DISPLAYSAVEFNODE

This command displays the fields associated with a single fnode in the R?SAVE file. R?SAVE must have been reserved when the volume was formatted. (That is, the RESERVE option in the FORMAT command must have been specified.) The format of the DISPLAYSAVEFNODE command is as follows:



2013

INPUT PARAMETER

fnodenum	The Hexadecimal number of the fnode to be displayed. This number can range from Ø through (maxfnodes - 1), where maxfnodes is the maximum number of fnodes defined when the volume was originally formatted.

OUTPUT

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

```
    Fnode number = <fnodenum>(saved)
                       flags : <flgs>
                        type : <typ>
          file gran/vol gran : <gran>
                       owner : <own>
     create,access,mod times : <crtime>, <acctime>, <modtime>
                  total size : <totsize>
                total blocks : <totblks>
            block pointer(1) : <blks>, <blkptr>
            block pointer(2) : <blks>, <blkptr>
            block pointer(3) : <blks>, <blkptr>
            block pointer(4) : <blks>, <blkptr>
            block pointer(5) : <blks>, <blkptr>
            block pointer(6) : <blks>, <blkptr>
            block pointer(7) : <blks>, <blkptr>
            block 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>
```

where:

<fnodenum> The Hexadecimal 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:

Bit	Meaning
Ø	Allocation status. This bit is set to 1 for allocated fnodes and set to Ø for free fnodes.
1	Long or short file attribute. This bit is set to 1 for long files and set to Ø 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 DISPLAYSAVEFNODE 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:

Value	Message
ØØ	fnode file
Ø1	volume map file
Ø2	fnode map file
Ø3	account file
Ø4	bad block file
Ø6	directory file
Ø8	data file
Ø9	volume label file

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

<own> User ID of the owner of the file.

<crtime> <acctime> <modtime>	Time and date of file creation, last access, and last modification. These values are expressed as the time 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.
<thissize>	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 has that access ID. Bits in the <access> field are set to indicate the following access rights:

Bit	Data File Operation	Directory Operation
Ø	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.

DESCRIPTION

The DISPLAYSAVEFNODE command provides a means of examining a single fnode in the R?SAVE file that serves as a backup for the fnode file. Since DISPLAYSAVEFNODE operates on the R?SAVE file, you must have reserved this file when the volume was formatted. (You reserve R?SAVE by specifying the RESERVE parameter when you invoke the FORMAT command.) If the R?SAVE file was not reserved when the volume was formatted, DISPLAYSAVEFNODE will return an error message.

ERROR MESSAGES

Message	Description
argument error	When you entered the command, you did not supply an argument. DISPLAYSAVEFNODE requires an argument.
no save area reserved when volume was formatted	The volume has not been formatted to support fnode backup. Format the volume using the Release 6.Ø, Version 3.1 (or later) FORMAT command with the RESERVE parameter specified.
not a named disk	The volume you specified when you invoked the Disk Verification Utility is a physical volume not a named volume.

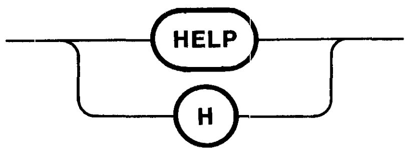
EXAMPLE

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*displaysavefnode 3C8 <CR>     or    dsf 3C8 <CR>


    Fnode number = 3C8 (saved)
                       flags : ØØ25  =>  short file
                        type : Ø8  =>  data file
          file gran/vol gran : Ø1
                       owner : ØØØ1
     create,access,mod times : ØØØØØØØØ, ØØØØØØØØ, ØØØØØØØØ
                  total size : ØØØØ2DØ1
                total blocks : ØØØØØØØC
            block pointer(1) : ØØØC, ØØ491Ø
            block pointer(2) : ØØØØ, ØØØØØØ
            block pointer(3) : ØØØØ, ØØØØØØ
            block pointer(4) : ØØØØ, ØØØØØØ
            block pointer(5) : ØØØØ, ØØØØØØ
            block pointer(6) : ØØØØ, ØØØØØØ
            block pointer(7) : ØØØØ, ØØØØØØ
            block pointer(8) : ØØØØ, ØØØØØØ
                   this size : ØØØØ3ØØØ
                    id count : ØØØ1
                 accessor(1) : ØF, ØØØ1
                 accessor(2) : ØØ, ØØØØ
                 accessor(3) : ØØ, ØØØØ
                      parent : Ø3C4
                     aux (*) : ØØØØØØ
```

HELP

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



2014

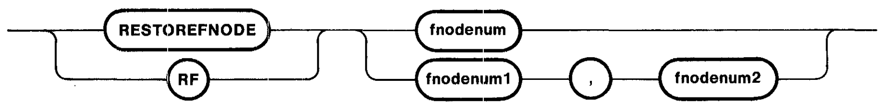
OUTPUT

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

```
                 allocate/free : allocate/free fnodes, space blocks, bad blocks
 backup/restore fnodes (bf/rf) : backup/restore fnode file to/from save area
                     Control-C : abort the command in progress
                          disk : display disk attributes
   display byte/word (d,db/dw) : display the buffer in (byte/word format)
        display directory (dd) : display the directory contents
            display fnode (df) : display fnode information
    display next block (>,dnb) : read and display 'next' volume block
display previous block (<,dpb) : read and display 'previous' volume block
      display save fnode (dsf) : display saved fnode information
                     exit,quit : quit disk verify
         list bad blocks (lbb) : list bad blocks on the volume
                      read (r) : read a disk block into the buffer
    restore volume label (rvl) : copy volume label from save area
                          save : save free fnodes, free space & bad block maps
substitute byte/word (s,sb/sw) : modify the buffer (byte/word format)
                        verify : verify the disk
                     write (w) : write to the disk block from the buffer
miscellaneous command-
                       address : convert block number to absolute address
                         block : convert absolute address to block number
                       hex/dec : display number as hexadecimal/decimal number
    add,+,sub,-,mul,*,div,/mod : arithmetic operations on unsigned numbers
```

RESTOREFNODE

This command copies an fnode or a range of fnodes from the R?SAVE file to the fnode file. Before changing the fnode file, RESTOREFNODE displays the fnode number to be changed and prompts you to confirm (by entering a 'y') that the fnode is to be restored. R?SAVE must have been reserved when the volume was formatted. (That is, the RESERVE option of the FORMAT command must have been specified.) The format of the RESTOREFNODE command is as follows:



2016

INPUT PARAMETER

fnodenum	The Hexadecimal number of the fnode to be restored. This number must be greater than or equal to zero and less than the maximum number of fnodes defined when the volume was formatted.
fnodenuml	The initial Hexadecimal fnode number in a range of fnodes to be restored. This number must be greater than or equal to zero and less than or equal to the final fnode number in the range (fnodenum2).
fnodenum2	The final Hexadecimal fnode number in a range of fnodes to be restored. This number must be greater than or equal to the initial fnode number in the range (fnodenuml) and less than the maximum number of fnodes defined when the volume was formatted.

OUTPUT

When the fnode is restored (the response to the confirmation query is 'Y'):

```
restore fnode    (fnodenum)?  Y <CR>
restored fnode number:    (fnodenum)
*
```

When the fnode is not restored (the response to the confirmation query is not 'Y'):

```
restore fnode    (fnodenum)?  <CR>
*
```

DESCRIPTION

The RESTOREFNODE command allows you to re-build a damaged fnode file, thereby re-establishing links to data that would otherwise be lost. RESTOREFNODE copys an fnode or a range of fnodes from the R?SAVE file (the fnode backup file) to the fnode file. Before each of the specified fnodes is copied, RESTOREFNODE displays a query prompting you to confirm that the indicated fnode is to be restored. You must reply to this query with the letter 'y' (either 'Y' or 'y') to restore the fnode. If you enter any other response, RESTOREFNODE will not restore the fnode and will pass on to the next fnode in the range.

Since RESTOREFNODE operates on the R?SAVE file, you must have reserved this file when the volume was formatted. (You reserve R?SAVE by specifying the RESERVE parameter when you invoke the FORMAT command to format the volume.) If the R?SAVE file was not reserved when the volume was formatted, RESTOREFNODE will return an error message.

WARNING: When using this command, be sure that any fnode you restore represents a file that has not been modified since the last fnode backup. RESTOREFNODE overwrites the specified fnode in the fnode file with the corresponding fnode in the R?SAVE file. If that fnode has not been backed up since the last file modification, a valid fnode may be overwritten with invalid data. Thus, all links to the associated file will be destroyed, and YOU WILL LOSE ALL OF THE DATA IN THE FILE.

ERROR MESSAGES

Message	Description
argument error	When you entered the command, you failed to supply an argument. This command requires an argument.
no save area reserved when volume was formatted	The volume has not been formatted to support fnode backup. Format the volume using Version 3.1 (or later) of the FORMAT command with the RESERVE parameter specified.
not a named disk	The volume you specified when you invoked the Disk Verification Utility is a physical volume not a named volume.

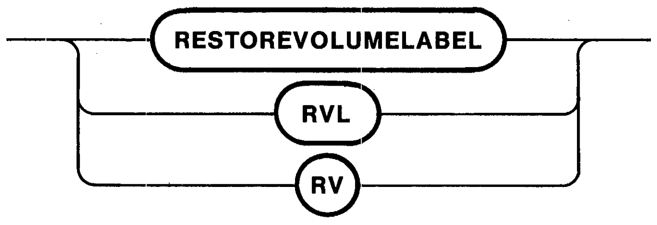
<fnode #>, fnode out of range	(1) The fnode number specified in the command is greater than or equal to the maximum number of fnodes defined when the volume was formatted; (2) The fnode number specified in the command is less than zero; or (3) the number specified for the initial fnode in the range is greater than the number specified for the final fnode.
allocation bit not set for saved fnode restore fnode <fnode #>?	The fnode you specified has not been backed up in the R?SAVE file. If you respond to the query with a 'Y', <u>THE DATA IN THE ASSOCIATED FILE WILL BE LOST</u>.

EXAMPLE

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
*restorefnode 9,0E <CR>     or    rf 9,0E <CR>
restore fnode     9?  Y <CR>
restored fnode number:    9
restore fnode    ØA?  y <CR>
restored fnode number:   ØA
restore fnode    ØB?  Y <CR>
restored fnode number:   ØB
restore fnode    ØC?  Y <CR>
restored fnode number:   ØC
allocation bit not set for saved fnode
restore fnode    ØD?   <CR>
allocation bit not set for saved fnode
restore fnode    ØE?  n <CR>
*
```

RESTOREVOLUMELABEL

This command copies the volume label from the R?SAVE file to the volume label offset on track Ø. R?SAVE must have been reserved when the volume was formatted. (That is, the RESERVE option of the FORMAT command must have been specified when the volume was formatted.) The format of the RESTOREVOLUMELABEL command is as follows:



2015

INPUT PARAMETERS

None

OUTPUT

volume label restored

DESCRIPTION

The RESTOREVOLUMELABEL command allows you to re-build a damaged volume label, thereby re-establishing links to data that would otherwise be lost. RESTOREVOLUMELABEL copies the volume label stored in the R?SAVE file to the volume label offset on track zero. When you use the Human Interface FORMAT command to create R?SAVE (by specifying the RESERVE parameter) the volume label is automatically copied to the file. Because the contents of the volume label do not change there is no other volume label backup required.

If an R?SAVE file has been reserved on a volume, the Disk Verification Utility can access that volume as a Named volume even if the volume label is damaged. When the original volume label is corrupted, the Disk Verification Utility attempts to use the backup copy in R?SAVE. If the backup label is used, a "DUPLICATE VOLUME LABEL USED" message appears when the utility is activated.

Since RESTOREVOLUMELABEL operates on the R?SAVE file, you must have reserved this file when the volume was formatted. (You reserve R?SAVE by specifying the RESERVE parameter when you invoked the FORMAT command.) If the R?SAVE file was not reserved when the volume was formatted, RESTOREVOLUMELABEL will return an error message.

ERROR MESSAGES

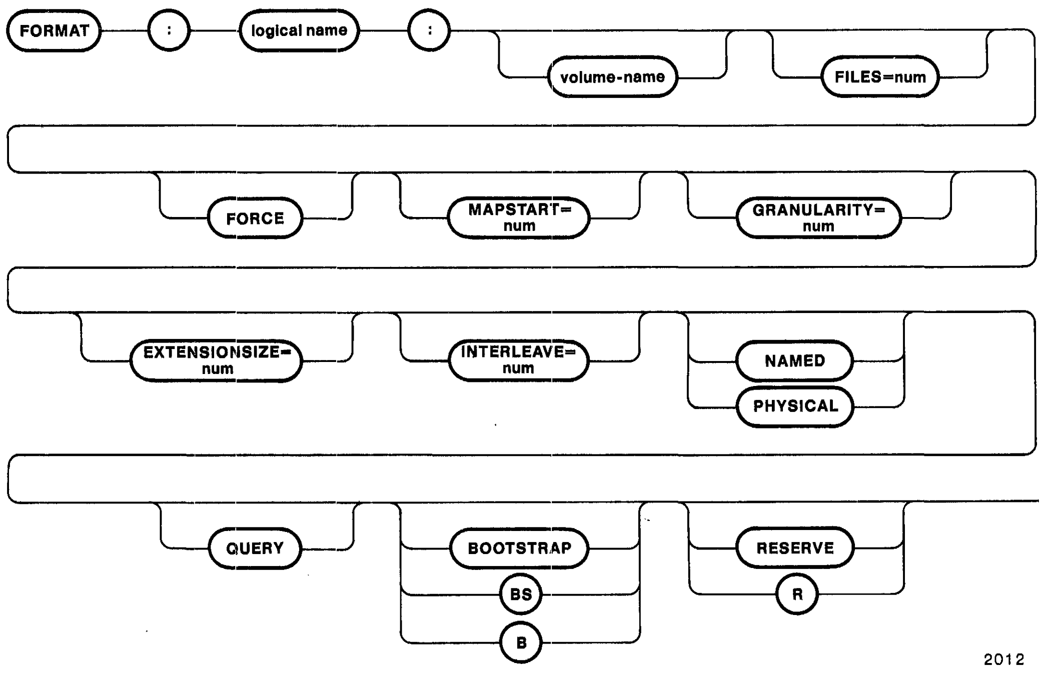
Message	Description
argument error	When you entered the command, you supplied an argument. This command does not accept an argument.
no save area reserved when volume was formatted	The volume has not been formatted to support volume label backup. Format the volume using the Release 6.Ø, Version 3.1 FORMAT command with the RESERVE parameter specified.
not a named disk	The volume you specified when you invoked the Disk Verification Utility utility is a physical volume not a named volume.

EXAMPLE

```
-diskverify :sd: <CR>
iRMX 86 Disk Verify Utility, V3.1
Copyright 1981, 1982, 1984 Intel Corporation
DUPLICATE VOLUME LABEL USED
*restorevolumelabel <CR>    or    rvl <CR>    or    rv <CR>
volume label restored
*
```

FORMAT

This command formats or re-formats a volume on an iRMX 86 secondary storage device, such as a diskette, tape drive, hard disk, or bubble memory. The format is as follows:



INPUT PARAMETERS

:logical-name:	Logical name of the physical device-unit to be formatted. You must surround the logical name with colons. Also, you must not leave space between the logical name and the succeeding volume name parameter.

volume-name	Six-character, alphanumeric ASCII name, without embedded blanks, to be assigned to the volume. If you include this parameter, you must not leave spaces between the logical name and the volume name.
FILES=num	Defines the maximum decimal number of user files that can be created on a NAMED volume. (This parameter is not meaningful when formatting a PHYSICAL volume and is ignored if specified for such volumes.) FORMAT uses the information specified in this parameter to determine how many structures (called <u>fnodes</u>) to create on the NAMED volume. The range for the FILES parameter is 1 through 32,761, although the maximum number of user files you can define depends on the settings of the GRANULARITY and EXTENSIONSIZE parameters (See the description of the FORMAT command in the iRMX 86 Operator's Manual). If no value is specified, FORMAT uses a default value of 2ØØ user files. When you use this parameter, FORMAT creates seven additional fnodes for internal system files. When the RESERVE parameter is used (setting aside the R?SAVE fnode backup file), the fnode for the R?SAVE file is allocated out of the fnodes reserved through the FILES parameter. (Thus, if you specify "FILES=3ØØØ", only 2999 fnodes remain available after the R?SAVE fnode has been allocated.)
FORCE	Forcibly deletes any existing connections to files on the volume before formatting the volume. If you do not specify FORCE, you cannot format the volume if any connections to files on the volume still exist.
MAPSTART=num	Gives the volume block number where the fnode file, bit map files, and the root directory should start. The size of the block is set by the GRANULARITY parameter. If no number is given, the Operating System puts the fnode file in the center of the volume. If the number is too low, the Operating System places the map files at the lowest available space on the volume.

GRANULARITY=num	Volume granularity; the minimum number of bytes to be allocated for each increment of file size on a NAMED volume. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) FORMAT rounds the value you specify up to the next multiple of the device granularity. Then it places the decimal number in the header of the volume, where it becomes the default file granularity when a file is created on the volume. The range is 1 through 65,535 (decimal) bytes, although the maximum allowable volume granularity depends on the settings of the FILES and EXTENSIONSIZE parameters. If not specified, the default granularity is the device granularity. Once the volume granularity is defined, it applies to every file created on that volume.

NOTE

Using a large volume granularity (in excess of 1Ø24), might cause users to exceed their memory limits when executing programs that reside on the volume. This error can occur because the Operating System uses the volume granularity as a minimum buffer size when reading and writing files.

EXTENSIONSIZE=num	Size, in bytes, of the extension data portion of each file. (This parameter is not meaningful for PHYSICAL volumes, and is ignored if specified for such volumes.) The range is Ø through 255 (decimal), although the maximum allowable extension size depends on the settings of the FILES and GRANULARITY parameters. If not specified, the default extension size is 3 bytes.
INTERLEAVE=num	Interleave factor for a NAMED or PHYSICAL volume. Acceptable values are 1 through 255 decimal. If not specified, the default value is 5.

NAMED	The volume can store only named files; that is, the volume can hold many files (up to the number of fnodes allocated), each of which can be accessed by its pathname. A diskette or hard disk surface are examples of devices that would be formatted as named files. If neither NAMED nor PHYSICAL is specified, the volume is formatted for the file specified when you attached the device (with the ATTACHDEVICE command).
PHYSICAL	The volume can be used only as a single, physical file. The GRANULARITY and FILES parameters are not meaningful when PHYSICAL is specified for the volume. If neither NAMED nor PHYSICAL is specified, the volume is formatted as the file type specified when you attached the device (with the ATTACHDEVICE command).
QUERY	Prompts the user for permission to format the volume. The Human Interface displays the following: `<volume name>, FORMAT?` If the user replies with a 'Y', 'y', 'R', or 'r', then the volume is formatted. Any other response is considered by the Human Interface to be a 'no'.
BOOTSTRAP	Writes the second stage of the bootstrap loader onto track Ø without formatting the volume. When this parameter is specified, all other parameters are ignored.
RESERVE	Creates the special file R?SAVE at the end of a volume after the volume has been initialized. The volume label file and the fnode file are then copied to R?SAVE. Later this file may be used in conjunction with the Disk Verification Utility to back up the fnode file on the volume.

DESCRIPTION

Every physical device-unit used for secondary storage must be formatted before it can be used for storing and then accessing its files. For example, every time you mount a previously unused diskette into a drive, you must enter the FORMAT command to format that diskette as a new volume before you can create, store and access files on it.

FORMAT provides a number of optional parameters that control the structure of the formatted volume (e.g. Files=, Interleave=, etc.) or dictate the method of formatting the volume (e.g. FORCE, QUERY, RESERVE, etc.) These parameters are optional and may be entered in any order.

Once a volume is formatted, its name becomes a volume identifier when you display any directory of the volume, and the name appears in the directory's heading. Although the Human Interface uses the volume name in its own internal processing when you access the volume, you need not specify the volume name in any subsequent command after the volume is formatted. You must specify only the logical name of the secondary storage device that contains the volume.

Internal Files

When you format a named volume, FORMAT creates either seven or eight (depending on whether the RESERVE option is set) internal system files. It places the names of four (or five) of these files in the root directory of the volume. The remaining files are not listed in the root directory. Unless you specify the INVISIBLE option, none of these files appear when the DIR command is invoked. The files are:

File	Description
R?SPACEMAP	Volume free space map
R?FNODEMAP	Free fnodes map
R?BADBLOCKMAP	Bad blocks map
R?VOLUMELABEL	Volume label
R?SAVE	Save area for fnodes

The Operating System grants the user WORLD read access to these files. Refer to the iRMX 86 DISK VERIFICATION UTILITY REFERENCE MANUAL for more information about these files.

Root Directory

FORMAT also uses one of the fnodes for the root directory. It lists the user who formats the volume as the owner, giving that user all access rights. No other user has access to the root directory until the owner explicitly grants access. The owner can grant other users access to the volume via the PERMIT command. However, because the owner has all access rights to the root directory, the owner can obtain exclusive access to the volume, and can obtain delete access to any file created on the volume, even files created by other users.

Extension Data

Each fnode contains a field that stores extension data for its associated file. An operating system extension can access and modify this extension data by invoking the A$GET$EXTENSION$DATA and A$SET$EXTENSION$DATA system calls (refer to the iRMX 86 BASIC I/O SYSTEM REFERENCE MANUAL for more information). When you format a volume, you can use the EXTENSIONSIZE parameter to set the size of the extension data field in each fnode. Although you can specify any size from Ø to 255 bytes, the Human Interface requires all fnodes to have at least 3 bytes of extension data.

Output Display

The FORMAT command displays one of the following messages when volume formatting is completed. For physical volumes:

```
    volume (<volume name>) will be formatted as a PHYSICAL volume
        device gran.        = <number>
        interleave          = <number>
        volume size         = <k-number> K (or M)

    TTTTTTTTTTTTTTTTTTTT...

    volume formatted
```

While the storage device is being formatted, FORMAT displays on the console the letter "T" for every 1ØØ tracks formatted. (This does not occur if the storage device is a floppy diskette.) For example, if you see three T's on the screen, the Operating System has finished formatting at least 3ØØ tracks. Displaying the T's on the screen is useful when you format large capacity disks. A continuous stream of T's lets you know that the system hasn't failed during the FORMAT operation.

For named volumes:

```
    volume (<volume name>) will be formatted as a NAMED volume
        granularity          = <number>     map start = <number>
        interleave           = <number>     sides = <sides>
        files                = <number>     density = <density>
        extensionsize        = <number>     disk size = <d-size>
        save area reserved   = (yes/no)
        volume size          = <k-number> K (or M)

    TTTTTTTTTTTT...

    volume formatted
```

where:

<number>	Position where the fnodes start.
<volume name>	Volume name specified in the FORMAT command.
<number>	Decimal number as specified in the command (or the default).
<k-number>	Volume size in K (1Ø24-byte units) or M (1Ø48576-byte units). FORMAT displays the volume size in Kbyte units unless the size is greater than 25 Mbytes.
<sides>	Number of sides of the volume that will be formatted (1 or 2). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<density>	Density at which the volume will be formatted (single or double). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
<d-size>	Size of the volume (8 or 5.25). This field is displayed only for flexible diskettes in which FORMAT can recognize this characteristic.
save area reserved	Indicates whether the R?SAVE file has been reserved for backing up the volume label and the fnode file. (Reserving the R?SAVE file is controlled by the RESERVE parameter.)

When the BOOTSTRAP parameter is specified:

```
    Bootstrap Loader written
```

ERROR MESSAGES

- <logical name>, can't attach device
  <logical name>, <exception value> : <exception mnemonic>

  FORMAT cannot attach the device for formatting, or it cannot re-attach the device (that is, restore it to its original condition) after formatting takes place.

- <logical name>, can't detach device
  <logical name>, <exception value> : <exception mnemonic>

  FORMAT cannot detach the device for formatting, which means that the volume does not exist, the volume is busy, or the device on which the volume is mounted is not currently attached to the system.

- <logical name>, device is in use

  You cannot format the volume because there are outstanding connections to files on the volume and you did not specify the FORCE parameter.

- map files do not fit with save area

  The volume is too small for both map files and save area, or the map start block is too high to allow room for map files and the save area.

- <vol-name>, fnode file size exceeds 65535 volume blocks

  The values you specified for fnode size, granularity, and extension data size is too large.

- <number>, invalid number

  You specified an out-of-range number for any of the FILES, GRANULARITY, EXTENSIONSIZE, or INTERLEAVE parameters.

- <logical-name>, map files do not fit

  The volume is too small for the map files or the map start block is too high to allow room for the map files.

- <logical name>, outstanding connections to device have been deleted

  There were outstanding connections to files on the volume. However, because you specified the FORCE parameter, FORMAT deleted those connections. This is a warning message that does not prevent FORMAT from formatting the volume.

- ØØ85 : E$LIST, too many values

  You entered multiple logical-name/volume-name combinations separated by commas. FORMAT can format only one volume per invocation.

- <logical-name>: <exception code> unit status <xx>

  An I/O error occurred while physically formatting the volume. <exception code> informs you of the type of error.

- <volume name>, volume name is too long

  FORMAT requires the volume name you specify to be 6 characters or less.