# VMS

VMS Analyze/Disk__Structure Utility Manual

digital

# VMS Analyze/Disk_ Structure Utility Manual

Order Number: AA–LA39A–TE

**April 1988**

This document describes how to use the Analyze/Disk_Structure Utility on VMS operating systems. The Analyze/Disk_Structure Utility was formerly called the Verify Utility.

| | |
|---|---|
| **Revision/Update Information:** | This document supersedes the *VAX/VMS Verify Utility Reference Manual*, Version 4.0. |
| **Software Version:** | VMS Version 5.0 |

---

**HOW TO ORDER ADDITIONAL DOCUMENTATION**
**DIRECT MAIL ORDERS**

## Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

---

™ PostScript is a trademark of Adobe Systems, Inc.

# Contents

# Contents

# Preface

## Intended Audience

This document is intended for system managers who need to check the readability and validity of Files-11 disk volumes. System managers can use the information in this document to detect and repair errors and inconsistencies.

## Document Structure

This document consists of the following four sections:

- Description—Provides a description of the Analyze/Disk_Structure Utility (ANALYZE/DISK_STRUCTURE).

- Usage Summary—Outlines the following ANALYZE/DISK_STRUCTURE information:

  -Invoking the utility
  -Exiting from the utility
  -Directing output
  -Restrictions or privileges required

- Qualifiers—Describes ANALYZE/DISK_STRUCTURE qualifiers, including format, parameters, and examples.

- Appendixes—Provides supplemental information.

## Associated Documents

The *VMS System Messages and Recovery Procedures Reference Manual* provides a complete description of each message generated during an ANALYZE /DISK_STRUCTURE session and describes the appropriate responses to those messages.

The *Guide to Maintaining a VMS System* provides additional information about the Analyze/Disk_Structure Utility.

# Preface

## Conventions

| Convention | Meaning |
|---|---|
| RET | In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.) |
| CTRL/C | A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box. |
| $ SHOW TIME<br>05-JUN-1988 11:55:22 | In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red. |
| $ TYPE MYFILE.DAT<br>.<br>.<br>.<br>. | In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown. |
| input-file, . . . | In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted. |
| [logical-name] | Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.) |
| quotation marks<br>apostrophes | The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark. |

# New and Changed Features

For VMS Version 5.0, ANALYZE/DISK_STRUCTURE does the following:

- Performs a more thorough search of the directory hierarchy to locate and repair damaged files.

- Distinguishes between deleted files and invalid file headers.

- Correctly processes files cataloged in more than one directory.

- Sorts listings by logical block number. Listings organized in this manner allow you to easily identify all blocks allocated to multiple files.

- Provides you the option of changing the name of a directory file whose extension and version are not ".DIR;1" (ODS–2 volumes only).

- Verifies the contents of reserved file VOLSET.SYS for a volume set. This verification confirms that all members of a volume set are accurately reflected.

- Allows you to save the extension headers of files when their primary headers cannot be found.

- Includes the file specification from the primary header of a file in all messages. This enhancement ensures that the correct name of the file is displayed.

- Allows the following options for defective directory files:

  - If the contents of the directory are invalid, you can discard the defective portion of the directory block.

  - If the header attributes are invalid, you can now either clear the directory flag or delete the directory file.

# ANALYZE/DISK_STRUCTURE Description

Use the Analyze/Disk_Structure Utility on a regular basis to check disks for inconsistencies and errors, and to recover lost files. ANALYZE/DISK_STRUCTURE detects problems on On-Disk Structure (ODS) Level 1 or 2 Files–11 disks that were caused by hardware errors, system errors, and user errors.

By using ANALYZE/DISK_STRUCTURE to identify and delete lost files and files marked for deletion, you can reclaim disk space.

ANALYZE/DISK_STRUCTURE performs the verification of a volume or volume set in eight distinct stages. During these stages, ANALYZE/DISK_STRUCTURE collects information used in reporting errors or performing repairs. However, ANALYZE/DISK_STRUCTURE repairs volumes only when you specify the /REPAIR qualifier. For a complete description of each of the eight stages, and an annotated example of an ANALYZE/DISK_STRUCTURE session, refer to Appendix C.

## 1 Error Reporting and Repair

You can invoke the utility to operate in any of the following three modes:

- Error reporting with no repairs
- Error reporting with repairs
- User-controlled selective repairs

By default, ANALYZE/DISK_STRUCTURE reports errors, but does not make repairs. For example, use the following command to report all errors on device DBA1:

```
$ ANALYZE/DISK_STRUCTURE DBA1:
```

When you issue this command, ANALYZE/DISK_STRUCTURE runs through eight stages of data collection, then, by default, prints a list of all errors and lost files to your terminal. One type of problem that ANALYZE/DISK_STRUCTURE locates is an invalid directory backlink; a backlink is a pointer to the directory in which a file resides. If your disk has a file with an invalid directory backlink, ANALYZE/DISK_STRUCTURE displays the following message and the file specification to which the error applies:

```
%VERIFY-I-BACKLINK, incorrect directory back link [SYSEXE]SYSBOOT.EXE;1
```

To instruct ANALYZE/DISK_STRUCTURE to repair the errors that it detects, use the /REPAIR qualifier. For example, the following command reports and repairs all errors on the DBA1 device:

```
$ ANALYZE/DISK_STRUCTURE DBA1:/REPAIR
```

If you want to select which errors ANALYZE/DISK_STRUCTURE repairs, use both the /REPAIR and /CONFIRM qualifiers:

```
$ ANALYZE/DISK_STRUCTURE DBA1:/REPAIR/CONFIRM
```

# ANALYZE/DISK_STRUCTURE Description

When you issue this command, ANALYZE/DISK_STRUCTURE displays a description of each error and prompts you for confirmation before making a repair. For example, the previous command might produce the following messages and prompts:

```
%VERIFY-I-BACKLINK, incorrect directory back link [SYSO]SYSMAINT.DIR;1

Repair this error? (Y or N): Y

%VERIFY-I-BACKLINK, incorrect directory back link
[SYSEXE]SYSBOOT.EXE;1]

Repair this error? (Y or N): N
```

Consider running ANALYZE/DISK_STRUCTURE twice for each volume. First, invoke the utility to report all errors. Evaluate the errors and decide on an appropriate action. Then invoke the utility again with the /REPAIR qualifier to repair all errors, or with the /REPAIR and /CONFIRM qualifiers to repair selected errors.

For complete descriptions of all errors and recommended actions, refer to the *VMS System Messages and Recovery Procedures Reference Manual.*

## 1.1 Recovering Lost Files

A *lost file* is a file that is not linked to a directory. Under normal circumstances, files should not become lost. However, files occasionally become lost because of disk corruption, hardware problems, or user error. For example, in cleaning up files and directories, you might inadvertently delete directories that still point to files. When you delete a directory file (a file with the file type DIR) without first deleting its subordinate files, the files referred to by that directory become lost files. Though lost, these files remain on the disk and consume space.

When you run ANALYZE/DISK_STRUCTURE specifying the /REPAIR qualifier, the utility places lost files in SYSLOST.DIR.

For example, to report and repair all errors and lost files found on the device DDA0, issue the following command:

```
$ ANALYZE/DISK_STRUCTURE/REPAIR/CONFIRM DDAO:
```

If it discovers lost files on your disk, ANALYZE/DISK_STRUCTURE issues messages similar to those that follow:

```
%VERIFY-W-LOSTHEADER, file (16,1,1) []X.X;1
        not found in a directory
%VERIFY-W-LOSTHEADER, file (17,1,1) []Y.Y;1
        not found in a directory
%VERIFY-W-LOSTHEADER, file (18,1,1) []Z.Z;1
        not found in a directory
%VERIFY-W-LOSTHEADER, file (19,1,1) []X.X;2
        not found in a directory
%VERIFY-W-LOSTHEADER, file (20,1,1) []Y.Y;2
        not found in a directory
%VERIFY-W-LOSTHEADER, file (21,1,1) []Z.;1
        not found in a directory
%VERIFY-W-LOSTHEADER, file (22,1,1) []Z.;2
        not found in a directory
%VERIFY-W-LOSTHEADER, file (23,1,1) LOGIN.COM;163
        not found in a directory
%VERIFY-W-LOSTHEADER, file (24,1,1) MANYACL.COM;1
        not found in a directory
```

All lost files in this example are automatically moved to SYSLOST.DIR.

## 1.2 ANALYZE/DISK_STRUCTURE Output

By default, the Analyze/Disk_Structure Utility directs all output to your terminal. If you prefer, you can use the /LIST qualifier to generate a file containing the following information for each file on the disk:

- File identification (FID)

- File name

- Owner

- Errors associated with the file

To generate a disk usage accounting file, use the /USAGE qualifier. The first record of the file, called the identification record, contains a summary of disk and volume characteristics. The identification record is followed by a series of summary records; one summary record is created for each file on the disk. A summary record contains the owner, size, and name of the file.

For more information on the disk usage accounting file, see Appendix D.

# ANALYZE/DISK_STRUCTURE Usage Summary

The Analyze/Disk_Structure Utility checks the readability and validity of Files–11 Structure Level 1 and Structure Level 2 disk volumes, and reports errors and inconsistencies.

You can detect most classes of errors by invoking the utility once and using its defaults.

**FORMAT**

**ANALYZE/DISK_STRUCTURE** *device-name:[/qualifier]*

**PARAMETER**

*device-name*
Specifies the disk volume or volume set to be verified. If you specify a volume set, all volumes of the volume set must be mounted as Files–11 volumes. (For information on the Mount Utility, refer to the *VMS Mount Utility Manual.*)

**usage summary**

Use the following command to invoke the utility:

`$ ANALYZE/DISK_STRUCTURE device-name: /qualifiers`

You can terminate an ANALYZE/DISK_STRUCTURE session by entering CTRL/C or CTRL/Y while the utility executes. You cannot resume operation by using the DCL command CONTINUE.

By default, ANALYZE/DISK_STRUCTURE directs all output to your terminal. Use the /USAGE or /LIST qualifiers to direct output to a file.

**To invoke the Analyze/Disk_Structure Utility, you must have BYPASS privilege.**

**Do not use ANALYZE/DISK_STRUCTURE on a disk that is currently being used for other file operations. This can produce error messages that incorrectly indicate severe file damage.**

## ANALYZE/DISK_STRUCTURE
## QUALIFIERS

# /[NO]CONFIRM

Determines whether the Analyze/Disk_Structure Utility prompts you to confirm each repair. If you respond with Y or YES, the utility performs the repair. Otherwise, the repair is not performed.

## FORMAT /[NO]CONFIRM

## DESCRIPTION

You can only use the /CONFIRM qualifier with the /REPAIR qualifier. The default is /NOCONFIRM.

## EXAMPLE

```
$ ANALYZE/DISK_STRUCTURE DBAO:/REPAIR/CONFIRM
%VERIFY-I-BACKLINK, incorrect directory back link [SYSO]SYSMAINT.DIR;1
Repair this error? (Y or N): Y
%VERIFY-I-BACKLINK, incorrect directory back link [SYSEXE]SYSBOOT.EXE;1
Repair this error? (Y or N): N
```

The command in this example causes the Analyze/Disk_Structure Utility to prompt you for confirmation before performing the indicated repair operation.

---

# /[NO]LIST[=filespec]

Determines whether the Analyze/Disk_Structure Utility produces a listing of the index file.

---

**FORMAT**    /LIST*[=filespec]*
/NOLIST

---

**DESCRIPTION**    If you specify /LIST, the utility produces a file that contains a listing of all FIDs, file names, and file owners. If you omit the file specification, the default is SYS$OUTPUT. If you include a file specification without a file type, the default type is LIS. You cannot use wildcard characters in the file specification.

The default is /NOLIST.

---

# EXAMPLE

```
$ ANALYZE/DISK_STRUCTURE DLA2:/LIST=INDEX
$ TYPE INDEX
Listing of index file on DLA2:
31-DEC-1988 20:54:42.22

(00000001,00001,001)   INDEXF.SYS;1
                                [1,1]
(00000002,00002,001)   BITMAP.SYS;1
                                [1,1]
(00000003,00003,001)   BADBLK.SYS;1
                                [1,1]
(00000004,00004,001)   000000.DIR;1
                                [1,1]
(00000005,00005,001)   CORIMG.SYS;1
                                [1,1]
  .
  .
  .
$
```

In this example, ANALYZE/DISK_STRUCTURE did not find errors on the device DLA2. Since the file INDEX was specified without a file type, the system assumes a default file type of LIS. The subsequent TYPE command displays the contents of the file INDEX.LIS.

# /[NO]READ_CHECK

Determines whether the Analyze/Disk—Structure Utility performs a read check of all allocated blocks on the specified disk. When the Analyze /Disk—Structure Utility performs a read check, it reads the disk twice; this ensures that it reads the disk correctly. The default is /NOREAD_CHECK.

## FORMAT          /[NO]READ_CHECK

## EXAMPLE

```
$ ANALYZE/DISK_STRUCTURE DMA1:/READ_CHECK
```

The command in this example directs ANALYZE/DISK—STRUCTURE to perform a read check on all allocated blocks on the device DMA1.

# /[NO]REPAIR

Determines whether the Analyze/Disk_Structure Utility repairs errors that are detected in the file structure of the specified device.

## FORMAT        /[NO]REPAIR

## DESCRIPTION

The Analyze/Disk_Structure Utility does not perform any repair operation unless you specify the /REPAIR qualifier. The file structure is software write-locked during a repair operation. The default is /NOREPAIR.

## EXAMPLE

```
$ ANALYZE/DISK_STRUCTURE DBA1:/REPAIR
```

The command in this example causes ANALYZE/DISK_STRUCTURE to perform a repair on all errors found in the file structure of device DBA1.

# ANALYZE/DISK_STRUCTURE
/USAGE[=filespec]

---

# /USAGE[=filespec]

Specifies that a disk usage accounting file should be produced, in addition to the other specified functions of the Analyze/Disk_Structure Utility.

---

## FORMAT /USAGE*[=filespec]*

---

## DESCRIPTION
If all or part of the file specification is omitted, ANALYZE/DISK_STRUCTURE assumes a default file specification of USAGE.DAT. The file is placed in the default directory [ACCOUNT].

---

## EXAMPLE

```
$ ANALYZE/DISK_STRUCTURE DBA1:/USAGE
$ DIRECTORY USAGE

Directory DISK$DEFAULT:[ACCOUNT]

USAGE.DAT;3

Total of 1 file.
```

The first command in this example causes ANALYZE/DISK_STRUCTURE to produce a disk usage accounting file. Since a file specification was not provided in the command line, ANALYZE/DISK_STRUCTURE uses both the default file name and directory [ACCOUNT]USAGE.DAT. The DIRECTORY command instructs the system to display all default information.

# A Supplemental Information—Files-11 Directory Structure

## A.1 File Identification (FID)

Each file on a Files-11 disk is identified by a unique, system-assigned file identification (FID) and a user-assigned alphanumeric name. The primary function of a Files-11 directory is to associate the user-assigned alphanumeric name of each file with the unique FID of the file. This association ensures that files present on a volume are retrievable by name.

The FID of a file consists of a set of three numbers. The first is the **file number** (num). The file system uses this number as an offset into the index file (reserved file INDEXF.SYS), which stores information for all files on a volume.

The second part of the FID is the **file sequence number** (seq), which represents the number of times a particular file number has been used. File numbers are allocated and deallocated as files are created and deleted. As a result, the file number alone cannot uniquely identify the file. By incrementing the sequence number each time a file number is used, the file system ensures that each file has a unique identification in INDEXF.SYS.

The third number in the FID is the **relative volume number** (RVN). This number indicates the volume (of a volume set) on which the file resides (ODS-2 only). If the volume set consists of a single volume, the RVN of all files on that volume is 1.

## A.2 Directory Hierarchy

The Files-11 ODS-1 structure supports a two-level directory hierarchy. Each user identification code (UIC) is associated with a user file directory (UFD). Each UFD is included in the master file directory (MFD) of the volume.

The VMS implementation of the Files-11 ODS-2 structure is a multilevel directory hierarchy. The top level of the directory structure is the master file directory (MFD). The MFD of a volume is always named [000000]. The MFD contains pointers to all top-level directories, including itself. The first level below the MFD is the user file directory (UFD). Levels below the UFD are called sub-directories.

Since directories are files, directories can contain files that are also directories. By nesting directories, users can construct directory hierarchies of up to eight levels deep (including the master file directory).

In a volume set, the MFD for all of the user directories on the volume set is located on relative volume 1. The entries of this MFD point to directories located on any volume in the set. These directories in turn point to files and subdirectories on any volume in the set. The MFD of any remaining volume in the set includes only the names of the reserved files for that volume.

# B Supplemental Information—Reserved Files

The Files–11 structure incorporates a set of nondeletable reserved files that are created when a volume or volume set is initialized. These files control the organization of a Files–11 disk.

ANALYZE/DISK_STRUCTURE rebuilds specific Files–11 reserved files and compares these files with their old versions. The utility reports and repairs (if you specified the /REPAIR qualifier) any discrepancies found during these comparisons.

Because ANALYZE/DISK_STRUCTURE uses reserved files, you may find it helpful to become acquainted with the names and functions of these files. The following sections discuss the reserved files that ANALYZE/DISK_STRUCTURE uses.

## B.1 INDEXF.SYS

INDEXF.SYS is a large extendable file made up of several sections. These sections provide the operating system with the information necessary to identify a Files–11 volume, initially access that volume, and locate all the files on that volume (including INDEXF.SYS itself).

ANALYZE/DISK_STRUCTURE is primarily concerned with the following three sections of INDEXF.SYS:

| | |
|---|---|
| Home Block | The home block identifies the volume as Files–11, establishes the characteristics of the volume, and serves as the entry point into the file structure of the volume. The total number of files that can be stored on a volume at any given time and the size of the bitmap of INDEXF.SYS are determined by fields in the home block. |
| Index File Bitmap | The index file bitmap is a bit string that controls the allocation of file headers. The index file bitmap contains one bit for each header. If the bit is set, then that file number (header) is in use; if the bit is clear, the header is not in use and may be assigned to a newly created file or extension header. |
| File Headers | The majority of INDEXF.SYS consists of file headers. The file header is a fixed-length block of fields that provide all the information required to identify and locate the contents and extents of a particular file. Note that a file header can also be an extension header. |

# Supplemental Information—Reserved Files
## B.1 INDEXF.SYS

---

## B.1.1    File Headers

Because they represent the current state of file storage on a volume, file headers are of particular interest to ANALYZE/DISK_STRUCTURE. Each file on a Files–11 disk (INDEXF.SYS included) is identified and located by a primary header (and extension headers, if required) in INDEXF.SYS.

Each fixed-length header contains both constant and variable-length data. This data is stored in one of the following six areas:

| | |
|---|---|
| Header | This area contains the header identification, the file number and its sequence number, the protection code for the file, and offsets to the other file header areas. |
| Ident | This area contains the identification and accounting data for the file (for example, the name of the file, its creation date and time, and backup date and time). |
| Map | This area contains a list of retrieval pointers that map the virtual blocks of the file to the logical blocks of the volume. Each pointer describes one group of consecutively numbered logical blocks that is allocated to the file. Retrieval pointers are arranged in the order of the virtual blocks they represent. |
| Access Control List | An optional area that contains ACL-related information. |
| Reserved | This area is reserved for use by special applications. |
| End Checksum | The last two bytes of the file header contain a 16-bit additive checksum of the preceding 255 words of the file header. The checksum helps verify that the block is a valid file header. |

A set of contiguous clusters is known as an extent. The size of an extent varies according to the number of contiguous clusters. For example, assume a file requires 1000 blocks of storage, and the file system finds a set of 800 contiguous blocks and a set of 200 contiguous blocks. The file would then be stored in two extents: one consisting of 800 blocks, the other of 200.

The **primary header** of a file points to the first extent of that file and to as many extents as can be stored in the map area of the primary header. When the number of extents required to contain a file exceeds the map area available in the primary header, or the ACL is too large to fit in the primary header, the file is allocated an **extension header**. Extension headers contain all the constant data of the primary header as well as the variable data (in the header map area and access control list) that specifies the locations of the extents to which the extension header points.

ANALYZE/DISK_STRUCTURE confirms the validity of a file by working its way down the list of primary and extension headers of the file. During this process, ANALYZE/DISK checks the validity of the file header, the chain of pointers to all extension headers, the retrieval pointers in all headers, and the attributes of the file.

## B.2 Master File Directory

As previously mentioned, the master file directory (MFD) is the root of the directory structure of the volume. It contains all reserved files, and all top-level user file directories. The file name for the MFD is 000000.DIR;1.

ANALYZE/DISK_STRUCTURE verifies all files contained in the directory structure by making comparisons to INDEXF.SYS. Any file found in INDEXF.SYS that is not traceable through the directory structure is termed lost. ANALYZE/DISK_STRUCTURE places lost files in the directory SYSLOST.DIR if you specified /REPAIR in the command.

## B.3 BITMAP.SYS

The storage bitmap file is a contiguous file that the file system uses to keep track of the available space on a volume. It consists of a storage control block (SCB) and the bitmap itself.

The SCB contains summary information about the volume (cluster factor, volume size, blocking factor, and so forth). Each bit in the bitmap represents an allocatable cluster on the volume. If a bit is set, the corresponding cluster is available for use. If a bit is clear, the cluster is not available.

During normal operation, the operating system moves portions of the bitmap in and out of cache memory. The state of each bit in memory is altered as clusters are allocated and deallocated. BITMAP.SYS is updated when the portion of the bitmap in cache is swapped back to disk. Since there is always a portion of the bitmap in cache, BITMAP.SYS never reflects the current state of allocated clusters on a disk (unless the disk is dismounted or write-locked).

One of the functions of ANALYZE/DISK_STRUCTURE is to build a current version of BITMAP.SYS from data extracted from INDEXF.SYS, so that BITMAP.SYS accurately reflects the status of free clusters on the disk.

## B.4 VOLSET.SYS

VOLSET.SYS is a reserved file that contains the name of the volume set, a list of labels for all the volumes in the set, and the attributes of each volume. ANALYZE/DISK_STRUCTURE uses this file to locate each volume in the set and confirm the attributes of each volume. Since all volume set information is stored in VOLSET.SYS on relative volume 1, ANALYZE/DISK_STRUCTURE ignores VOLSET.SYS on all other volumes.

## B.5 QUOTA.SYS

QUOTA.SYS is a reserved file that is used by the file system to keep track of the disk usage of each UIC on a volume. If you have enabled disk quota checking for a volume, the records of the file QUOTA.SYS contain all the UICs on the volume. The system constantly updates QUOTA.SYS to reflect the current disk usage, the maximum allowed disk usage, and the permitted overdraft for each UIC.

# Supplemental Information—Reserved Files
## B.5 QUOTA.SYS

During the course of its operations, ANALYZE/DISK_STRUCTURE creates
a version of QUOTA.SYS in memory that reflects the actual disk usage
for each UIC. This version is eventually compared to the disk version of
QUOTA.SYS. If ANALYZE/DISK_STRUCTURE detects any disparities in
disk usage, ANALYZE/DISK_STRUCTURE notifies you. If you invoked
ANALYZE/DISK_STRUCTURE with the /REPAIR qualifier, the disk version
of QUOTA.SYS is updated.

# C Supplemental Information—Stage Checks

## C.1 Stage Checks

ANALYZE/DISK_STRUCTURE performs the verification of a volume or volume set in eight distinct stages. During these stages, ANALYZE/DISK_STRUCTURE compiles information that is used in reporting errors and performing repairs.

Before ANALYZE/DISK_STRUCTURE can proceed with each stage, it must perform the following four initialization functions:

- Read the device name, validate access to the device, and save the device name

- Read the user-specified file names for the /LIST and /USAGE qualifiers, if specified, and open the files

- Assign all appropriate channels to the device being checked

- Write-lock the volume set to prevent simultaneous updates

The following sections describe the eight stages that ANALYZE/DISK_STRUCTURE goes through while verifying a disk. These descriptions assume that you specified the /REPAIR qualifier in the command. An annotated ANALYZE/DISK_STRUCTURE listing is included at the end of this appendix.

### Stage 1

In Stage 1, ANALYZE/DISK_STRUCTURE gathers various volume information (such as cluster size, volume labels, and the number of volumes in the set) from several reserved files, verifies the information for accuracy, reports all discrepancies, and corrects problems discovered during this stage.

ANALYZE/DISK_STRUCTURE identifies the volume and all the characteristics of that volume by using the parameters of the home block in INDEXF.SYS. When ANALYZE/DISK confirms this information, it builds a current version of VOLSET.SYS in memory and reads and verifies the status control block (SCB) of BITMAP.SYS.

ANALYZE/DISK_STRUCTURE then compares the volume-set attributes for the version of VOLSET.SYS in memory to the attributes listed in the version of VOLSET.SYS resident on the volume, reports discrepancies, and corrects errors.

### Stage 2

In Stage 2, ANALYZE/DISK_STRUCTURE copies the current version of QUOTA.SYS into working memory, and establishes the structure on which another QUOTA.SYS file is built during subsequent stages. In Stage 7, these copies are compared to each other, and inconsistencies are reported.

# Supplemental Information—Stage Checks

## C.1 Stage Checks

### Stage 3

Stage 3 checks consist of ANALYZE/DISK_STRUCTURE operations that use the reserved file INDEXF.SYS. During Stage 3, ANALYZE/DISK_STRUCTURE opens INDEXF.SYS, reads each file header, and completes the following steps:

- Validates each file's FID, and confirms that all files can be retrieved through the FID

- Validates the header and the revision date of each file

- Validates any extension headers of each file

- Confirms that each segment number reflects the proper sequence of extension headers

ANALYZE/DISK_STRUCTURE also does the following during Stage 3:

- Collects multiple references to one extension header and builds a map of all such references

- Determines the high block (HIBLK) and end-of-file block (EFBLK) record attributes and compares these values with the recorded values in INDEXF.SYS

- Checks the high-water mark (HIWATERMARK).

While performing these checks, ANALYZE/DISK_STRUCTURE builds several maps that it uses in subsequent stages. Table C-1 briefly describes each map built in Stage 3.

**Table C-1  Stage 3 Maps**

| Bitmap | Function |
|---|---|
| Valid file numbers | The current state of the bitmap for INDEXF.SYS |
| Lost file numbers | All the valid file numbers not yet found in a directory |
| Directory files | List of all directory files |
| Extension linkages | List of all FIDs referenced by extension linkages |
| Multiply-allocated clusters | List of all clusters that are referenced by more than one header |
| Allocated clusters | All allocated clusters on the volume (or volume set) |
| System map | The new storage bitmap |
| Valid file backlink | A map of all valid file backlinks |
| Invalid backlink | A map of all invalid backlinks |

### Stage 4

In Stage 4, ANALYZE/DISK_STRUCTURE builds a current version of BITMAP.SYS using the maps built during Stage 3. In addition, ANALYZE /DISK_STRUCTURE resolves all multiple references to extension headers

and corrects any discrepancies in the map sections of headers. In Stage 4, ANALYZE/DISK_STRUCTURE does the following:

- Copies BITMAP.SYS into working memory

- Compares the version of BITMAP.SYS built in Stage 3 with the copy of BITMAP.SYS just read into memory and corrects discrepancies

- Compares the corrected version of BITMAP.SYS with a map built from INDEXF.SYS

- Writes a corrected version of BITMAP.SYS to disk

- Resolves multiply-allocated clusters

- Rewrites header maps to reflect adjustments to multiply-allocated clusters

### Stage 5

In this stage, ANALYZE/DISK_STRUCTURE completes a pass of all entries in the invalid backlink map. ANALYZE/DISK_STRUCTURE searches the directory hierarchy of the volume to confirm that all files included in INDEXF.SYS are retrievable through the directory structure. In addition, ANALYZE/DISK_STRUCTURE identifies lost directories and attempts to reestablish valid backlinks to those directories.

In Stage 5, ANALYZE/DISK_STRUCTURE does the following:

- Confirms the locations of all directories listed in the directory map (compiled in Stage 3) and the subsequent files in those directories

- Enters all directories indicated as lost and locates a valid parent (if any)

- Places lost files in SYSLOST.DIR if you specified /REPAIR.

### Stage 6

Stage 6 is essentially a cleanup operation for lost file headers. Following Stage 5, ANALYZE/DISK_STRUCTURE is left with a list of files that are truly lost— files that have backlinks to nonexistent directories. These files were not traceable through the directory structure.

During Stage 3, ANALYZE/DISK_STRUCTURE compiled a map of backlinks for all files. As each file is validated, the corresponding flag bit in the map is cleared. As a result, all backlinks with set bits are invalid. During Stage 3, ANALYZE/DISK also compiled a list of lost files. ANALYZE/DISK_STRUCTURE uses both these lists in Stage 6. During this stage, ANALYZE/DISK_STRUCTURE does the following:

- Checks the backlink map to locate all files with invalid backlinks, then repairs backlinks

- Checks the lost file bitmap for lost files

- If you specified the /USAGE qualifier, creates a file that lists files

# Supplemental Information—Stage Checks
## C.1 Stage Checks

### Stage 7

In this stage, ANALYZE/DISK_STRUCTURE compares the values stored in the quota file built during Stage 2 with those stored in the reserved file QUOTA.SYS. During Stage 7, ANALYZE/DISK_STRUCTURE opens QUOTA.SYS and performs the following operations:

- Compares the block usage for each UIC listed in QUOTA.SYS to parallel statistics listed in the copy of QUOTA.SYS built in Stage 2.

- Modifies QUOTA.SYS such that values in QUOTA.SYS match values in the copy built in Stage 2.

- Closes QUOTA.SYS

### Stage 8

Throughout the first seven stages, ANALYZE/DISK_STRUCTURE places operations that cannot be performed during a particular stage on a deferred list. The list includes FIDs sorted by operation. In Stage 8, ANALYZE /DISK_STRUCTURE performs all operations stored on the deferred list. In Stage 8, ANALYZE/DISK_STRUCTURE does the following:

- Removes an FID from the deferred list, renames the file, and adds the file to SYSLOST.DIR or to a user-specified directory

- Updates QUOTA.SYS to reflect all additional blocks used by the UIC that received the lost file

- Updates VOLSET.SYS to correct inconsistencies discovered during previous ANALYZE/DISK_STRUCTURE stages

## C.2 Annotated Example

The following is an annotated example of an ANALYZE/DISK_STRUCTURE session. The command used to generate this example did not include the /REPAIR qualifier.

**Example C–1   ANALYZE/DISK_STRUCTURE—Annotated Example**

---

```
%VERIFY-I-BADHEADER, file (487,173,1) MAIL$0004008EEAEE0572.MAI;1 ❶
        invalid file header
%VERIFY-I-BADHEADER, file (531,112,1) MAIL$0004008EEFBB198B.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (589,104,1) MAIL$0004008EEAF199B9.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (604,157,1) MAIL$0004008EF12C3B28.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (674,247,1) MAIL$0004008EF6053C9B.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (688,41,1) MAIL$0004008EF608AFF4.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (689,135,1) MAIL$0004008EEE445A31.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (750,71,1) MAIL$0004008EEED19ADF.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (753,217,1) MAIL$0004008EE7C4A017.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (780,236,1) MAIL$0004008EF777ACA8.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (852,57,1) MAIL$0004008EF06C15F6.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (856,44,1) MAIL$0004008EE7D2520D.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1059,42,1) MAIL$0004008EEB045608.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1134,76,1) MAIL$0004008EE9EC806D.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1316,147,1) MAIL$0004008EEEDA734F.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1350,74,1) MAIL$0004008EE89BA8B0.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1351,64,1) MAIL$0004008EEB09B036.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1490,104,1) MAIL$0004008EE8B448B0.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1493,106,1) LASTNOTIC.NIL;1
        invalid file header
%VERIFY-I-BADHEADER, file (1548,204,1) MAIL$0004008EF7B4D1B8.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1613,61,1) MAIL$0004008EECEE4BA5.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1812,81,1) MAIL$0004008EE7DF05EC.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1848,26,1) MAIL$0004008EF78659B9.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1983,34119,1) MAIL$0004008EE7E49C13.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (1987,33907,1) REMIND.CAL;9
        invalid file header
%VERIFY-I-BADHEADER, file (2196,123,1) MAIL$0004008EE6FA2DC9.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (2372,125,1) MAIL$0004008EF06339F9.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (2569,67,1) MAIL$0004008EF2BF0C15.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (2605,72,1) MAIL$0004008EE856FC73.MAI;1
        invalid file header
```

---

**Example C–1 Cont'd. on next page**

# Supplemental Information—Stage Checks
## C.2 Annotated Example

### Example C-1 (Cont.) ANALYZE/DISK_STRUCTURE—Annotated Example

```
%VERIFY-I-BADHEADER, file (2616,70,1) MAIL$0004008EF063C04F.MAI;1
        invalid file header
%VERIFY-I-BADHEADER, file (2774,29818,1) LASTNOTIC.NIL;1
        invalid file header
%VERIFY-I-ALLOCCLR, blocks incorrectly marked allocated ❷
        LBN 442398 to 445538, RVN 1
%VERIFY-I-BADHEADER, file (487,0,1) MAIL$0004008EEAEE0572.MAI;1  ❸
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (487,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (531,0,1) MAIL$0004008EEFBB198B.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (531,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (589,0,1) MAIL$0004008EEAF199B9.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (589,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (604,0,1) MAIL$0004008EF12C3B28.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (604,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (674,0,1) MAIL$0004008EF6053C9B.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (674,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (688,0,1) MAIL$0004008EF608AFF4.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (688,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (689,0,1) MAIL$0004008EEE445A31.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (689,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (750,0,1) MAIL$0004008EEED19ADF.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (750,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (753,0,1) MAIL$0004008EE7C4A017.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (753,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (780,0,1) MAIL$0004008EF777ACA8.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (780,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (852,0,1) MAIL$0004008EF06C15F6.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (852,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (856,0,1) MAIL$0004008EE7D2520D.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (856,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1059,0,1) MAIL$0004008EEB045608.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1059,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1134,0,1) MAIL$0004008EE9EC806D.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1134,0,1)
        lost extension file header
```

### Example C-1 Cont'd. on next page

**Example C–1 (Cont.) ANALYZE/DISK_STRUCTURE—Annotated Example**

```
%VERIFY-I-BADHEADER, file (1316,0,1) MAIL$0004008EEEDA734F.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1316,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1350,0,1) MAIL$0004008EE89BA8B0.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1350,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1351,0,1) MAIL$0004008EEB09B036.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1351,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1490,0,1) MAIL$0004008EE8B448B0.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1490,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1493,0,1) LASTNOTIC.NIL;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1493,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1548,0,1) MAIL$0004008EF7B4D1B8.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1548,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1613,0,1) MAIL$0004008EECEE4BA5.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1613,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1812,0,1) MAIL$0004008EE7DF05EC.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1812,0,1)       '
        lost extension file header
%VERIFY-I-BADHEADER, file (1848,0,1) MAIL$0004008EF78659B9.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1848,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1983,0,1) MAIL$0004008EE7E49C13.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1983,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (1987,0,1) REMIND.CAL;9
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (1987,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (2196,0,1) MAIL$0004008EE6FA2DC9.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2196,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (2372,0,1) MAIL$0004008EF06339F9.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2372,0,1)
        lost extension file header
```

**Example C-1 (Cont.)  ANALYZE/DISK_STRUCTURE—Annotated Example**

```
%VERIFY-I-BADHEADER, file (2569,0,1) MAIL$0004008EF2BF0C15.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2569,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (2605,0,1) MAIL$0004008EE856FC73.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2605,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (2616,0,1) MAIL$0004008EF063C04F.MAI;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2616,0,1)
        lost extension file header
%VERIFY-I-BADHEADER, file (2774,0,1) LASTNOTIC.NIL;1
        invalid file header
%VERIFY-I-LOSTEXTHDR, file (2774,0,1)
        lost extension file header
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [ALLWAY]NOTES.LOG;25 ❹
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN.BOOTS]LOADER.OBJ;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN.BOOTS]SYSGEN.OBJ;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]MAIL_20600841.TMP;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]NETSERVER.LOG;181
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]NETSERVER.LOG;180
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]NETSERVER.LOG;179
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]NETSERVER.LOG;178
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BLAIN]NETSERVER.LOG;170
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BOEMUS.MAIL]MAIL$0004008EF94A72A0.MAI;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BOEMUS]NETSERVER.LOG;10
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [BOEMUS]UPDATE.LOG;1
%VERIFY-I-BACKLINK, incorrect directory back link [CALGON.GER]OBJ.DIR;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [CALGON]T.TMP;1
%VERIFY-I-BACKLINK, incorrect directory back link [CLABIN.BACKUP.TMPSRC]BACKDEF.SDL;1
%VERIFY-I-BACKLINK, incorrect directory back link [CLABIN.BACKUP.TMPSRC]COMMON.REQ;1
%VERIFY-I-BACKLINK, incorrect directory back link [CLABIN.BACKUP.TMPSRC]DUMMY.MSG;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [CLABIN.NMAIL]NMAIL.LOG;77
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [CLABIN.NMAIL]NMAIL.LOG;76
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DESIN.8800]2840HT86.GNC;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DESIN.8800]2840TP86.GNC;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.MAIL]MAIL$0004008EF94A79B3.MAI;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]MORT.OBJ;15
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;36
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;35
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;34
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;33
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;32
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;31
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [DOWNE.PRO]OUTPUT.LOG;30
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GAMBLE]CONFLICTS.LIS;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GAMBLE.DOC]SMP.LOCK;6
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GAMBLE]NETSERVER.LOG;5
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GAMBLE.NMAIL]NMAIL.LOG;22
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GAMBLE.NMAIL]NMAIL.LOG;21
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GILLEY.MAIL]MAIL$0004008EF94A7B70.MAI;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GILLEY]NETSERVER.LOG;657
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [GILLEY]NETSERVER.LOG;656
```

**Example C-1 Cont'd. on next page**

**Example C-1 (Cont.)  ANALYZE/DISK_STRUCTURE—Annotated Example**

```
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;33
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;32
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;31
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;30
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;29
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;28
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;27
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;26
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;25
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [HALL]2.LOG;24
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [NAMOLLY]NETSERVER.LOG;2
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [NAMOLLY]NETSERVER.LOG;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [RUSS]082654.LOG;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [SCHROEDER.LOGIN]NETSERVER.LOG;17
%VERIFY-I-BADDIR, directory [SYSLOST.BOOTS] has invalid format
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOEN]NETSERVER.LOG;374
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOEN]NETSERVER.LOG;373
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOEN]NETSERVER.LOG;367
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOMAS.MAIL]MAIL$0004008EF94D75EB.MAI;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOMAS.MAIL]MAIL$0004008EF955DDF3.MAI;1
%VERIFY-I-BADDIRENT, invalid file identification in directory entry [THOMAS.MAIL]MAIL$0004008EFD118B44.MAI;1
%VERIFY-I-LOSTSCAN, due to directory errors, lost files will not be entered ❺
%VERIFY-I-INCQUOTA, QUOTA.SYS indicates 69663 blocks used, actual use is 69740 blocks for [11,402] ❻
%VERIFY-I-INCQUOTA, QUOTA.SYS indicates 1764 blocks used, actual use is 1770 blocks for [12,12]
%VERIFY-I-INCQUOTA, QUOTA.SYS indicates 0 blocks used, actual use is 31 blocks for [11,720]
```

❶  ANALYZE/DISK_STRUCTURE has completed the first two stages, and is beginning Stage 3. Stage 1 involves collection and verification of various volume information. ANALYZE/DISK_STRUCTURE found no problems with volume information. In Stage 2, ANALYZE/DISK_STRUCTURE copies the current version of QUOTA.SYS to working memory, and builds the structure on which a new copy is built during subsequent stages. The first error message is produced by Stage 3. Stage 3 uses the reserved file INDEXF.SYS to locate a variety of file problems. Here, Stage 3 detects a number of invalid file headers. Note that the error message includes the FID and the file name.

❷  This error message is produced during Stage 4, during which ANALYZE /DISK_STRUCTURE builds a current version of BITMAP.SYS, resolves multiple references to extension headers, and corrects discrepancies in the map sections of headers. Here, ANALYZE/DISK_STRUCTURE has found that the specified logical blocks on the specified relative volume were marked allocated in the storage bit map, but were not allocated to a file.

❸  This message marks the beginning of Stage 5. Here, messages stating "lost extension file header" and "invalid file header" indicate that ANALYZE/DISK_STRUCTURE is performing a pass of all entries placed on the invalid backlink map. This map was created in Stage 3.

❹  This message marks the beginning of the second phase of Stage 5, in which ANALYZE/DISK_STRUCTURE confirms that all files in INDEX.SYS are retrievable through the directory structure. Here, the series of "invalid file identification . . . " messages indicates those directory entries that did not contain a valid file identification.

❺  This message is produced by Stage 6, which is essentially a cleanup phase for lost files. This message indicates that ANALYZE/DISK_STRUCTURE encountered errors during the directory scan that were reported in previous messages. As a result, the file is not entered in directory [SYSLOST].

# Supplemental Information—Stage Checks
## C.2 Annotated Example

**Example C-1 (Cont.)   ANALYZE/DISK_STRUCTURE—Annotated Example**

❻   Here, ANALYZE/DISK_STRUCTURE begins Stage 7, in which it compares values stored in the quota file built during Stage 2 with values in the reserved file QUOTA.SYS. The last three messages here indicate discrepancies between the two files.

Note that no messages were produced during Stage 8. During Stage 8, ANALYZE/DISK_STRUCTURE executes all operations placed on the deferred list, and if you specified /REPAIR, updates QUOTA.SYS and VOLSET.SYS as necessary.

# D Supplemental Information—Usage File

## D.1 The ANALYZE/DISK_STRUCTURE Usage File

When you specify the /USAGE qualifier, ANALYZE/DISK_STRUCTURE creates a disk usage accounting file. The first record of this file, the identification record, contains a summary of the disk and volume characteristics. The identification record is followed by many file summary records, one record for each file on the disk. Each file summary record contains the owner, size, and name of a file.

The identification record is characterized by the type code USG$K_IDENT in the USG$B_TYPE field of the record. Table D-1 contains a description of all the fields in this record.

**Table D-1   Identification Record Format**
**(Length USG$K_IDENT_LEN)**

| Field | Meaning |
|-------|---------|
| USG$L_SERIALNUM | Serial number of the volume. This is an octal longword value. |
| USG$T_STRUCNAM | Volume set name (if the volume is part of a volume set). For a Files–11 Structure Level 1 volume, this field contains binary zeros; for a Files–11 Structure Level 2 volume that is not part of a volume set, this field contains spaces. The length of this field is USG$S_STRUCNAME. |
| USG$T_VOLNAME | Volume name of relative volume 1. The length of this field is USG$S_VOLNAME. |
| USG$T_OWNERNAME | Volume owner name. The length of this field is USG$S_OWNERNAME. |
| USG$T_FORMAT | Volume format type. For a Files–11 Structure Level 1 volume, this field contains "DECFILE11A"; for a Files–11 Structure Level 2 volume, this field contains "DECFILE11B". The length of this field is USG$S_FORMAT. |
| USG$Q_TIME | Quadword system time when this usage file was created. The length of this field is USG$S_TIME. |

Each file summary record is characterized by the type code USG$K_FILE in the USG$B_TYPE field of the record. Table D-2 contains a description of all the fields in these records.

**Table D–2  File Record Format (Length USG$K_FILE_LEN)**

| Field | Meaning |
|-------|---------|
| USG$L_FILEOWNER | File owner UIC. This can be considered as a single longword value or as two word values (USG$W_UICMEMBER and USG$W_UICGROUP). |
| USG$W_UICMEMBER | The member field of the file owner UIC. This is an octal word value. |
| USG$W_UICGROUP | The group field of the file owner UIC. This is an octal word value. |
| USG$L_ALLOCATED | Number of blocks allocated to the file, including file headers. This is a decimal longword value. |
| USG$L_USED | Number of blocks used, up to and including the end-of-file block. This is a decimal longword value. |
| USG$W_DIR_LEN | Length of the directory string portion of USG$T_FILESPEC, including the brackets. This is a decimal word value. |
| USG$W_SPEC_LEN | Length of the complete file specification in USG$T_FILESPEC. This is a decimal word value. |
| USG$T_FILESPEC | File specification, in the following format: |
|  | [dir]nam.typ;ver |
|  | This field is of variable length. A file that has more than one directory entry is listed under the first file specification found. A lost file has an empty directory string "[ ]" and the file name is taken from the file header. In some cases this information does not exist; you must take this into consideration when you write application programs to process the usage file. The length of this field is USG$S_FILESPEC. |

The symbolic names referenced in both the identification and the file summary records are defined in the system definition macro $USGDEF. The length of the identification record is USG$K_IDENT_LEN. The length of a file summary record is USG$K_FILE_LEN.

# Index

# Index

# Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| I rate this manual's: | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page      Description

_____   _____

_____   _____

_____   _____

_____   _____

_____   _____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____   Dept. _____

Company _____   Date _____

Mailing Address _____

_____   Phone _____

**digital**™

# BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

# Reader's Comments

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

| I rate this manual's: | Excellent | Good | Fair | Poor |
|---|---|---|---|---|
| Accuracy (software works as manual says) | ☐ | ☐ | ☐ | ☐ |
| Completeness (enough information) | ☐ | ☐ | ☐ | ☐ |
| Clarity (easy to understand) | ☐ | ☐ | ☐ | ☐ |
| Organization (structure of subject matter) | ☐ | ☐ | ☐ | ☐ |
| Figures (useful) | ☐ | ☐ | ☐ | ☐ |
| Examples (useful) | ☐ | ☐ | ☐ | ☐ |
| Index (ability to find topic) | ☐ | ☐ | ☐ | ☐ |
| Page layout (easy to find information) | ☐ | ☐ | ☐ | ☐ |

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page      Description

_____    _____

_____    _____

_____    _____

_____    _____

_____    _____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

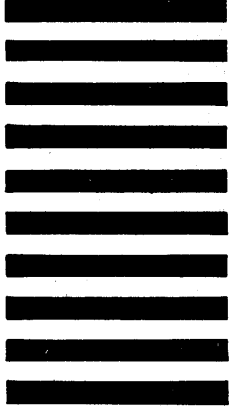I am using **Version** _____ of the software this manual describes.

Name/Title _____  Dept. _____

Company _____  Date _____

Mailing Address _____

_____  Phone _____

**d i g i t a l** ™

## BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987