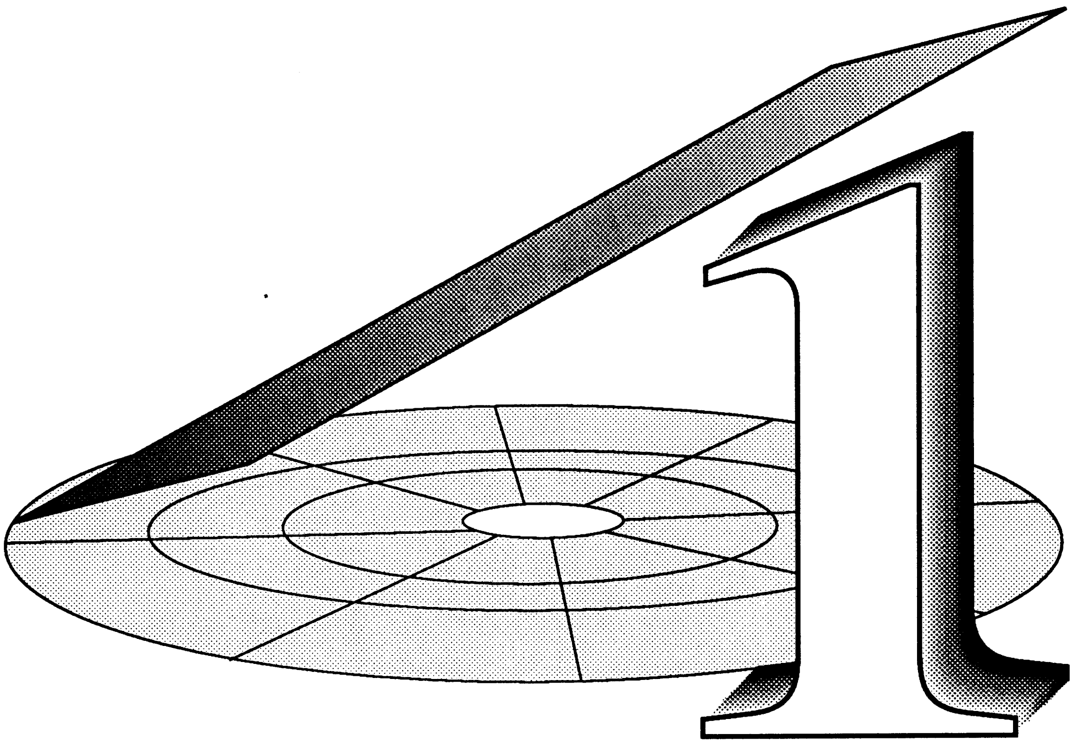
Optimize Utility Set 1

Documentation



JWT Enterprises
5755 Lockwood Blvd.
Youngstown, OH 44512

Optimize Utility Set 1

A Product of JWT Enterprises

INTRODUCTION

Efficient operation of the OS-9 operating system's disk routines rests upon the amount of fragmentation in a disk's files and the amount of blank entries in a disk's directories. The two utilities contained in this optimize set, *optimize* and *inq*, both deal with fragmentation and the "padding" of directories with blank entries (see the *Background* section for more information about padding). This manual assumes a basic familiarity with the OS-9 Operating System. For more help, see the *Getting Started* section at the end of the manual for additional information.

COPYRIGHT NOTICE

DISCLAIMER

Every effort has been made to insure the reliability and accuracy of this manual and the accompanying software. *JWT Enterprises* does not assume any liability for any damage incurred though the use of the documentation or software. The product is sold on an as-is basis, and *JWT Enterprises* is not responsible for any damage done to the product except that which is covered in the limited warranty.

LIMITED WARRANTY

The user must be a registered owner of the product in order to take advantage of the warranty. *JWT Enterprises* warrants the product from defects in production or damage due to transportation for a period of sixty (60) days after the date of purchase. This warranty is limited to the replacement of the product at the expense of *JWT Enterprises* during this period at the discretion of *JWT Enterprises*. This warranty excludes software defects and defects caused by tampering, negligence, and abuse of the product.

BACKGROUND INFORMATION

The following information is a more detailed description of directory structures, file structures, and file fragmentation. For further reference, the RBF Manager section of the *OS-9 Technical Reference* manual contains the actual tables and technical data for the more advanced user. This information intends to familiarize the user with the basics of these topics.

Directory Structures:

Each OS-9 directory can actually be thought of as a file with 32-byte entries for each file and sub-directory contained in the directory. The directory has a file descriptor sector and segments just as a regular file has, and directory entries and file entries are stored the same way in a directory. The only difference between a file and a directory is that the "D" attribute is set in the directory's attributes.

Each 32-byte entry is divided into two parts; the first 29 bytes contain the name of the file or directory, and the last 3 bytes contain the disk sector number of the file or directory's descriptor sector[1]. The last character of the filename always has the value 128 added to it—in other words, the most significant bit of the last character is always set.

The first two entries in any directory are always ".." and ".". The first entry, "..", refers to the *parent directory*. This parent directory is the directory that contains the entry for the directory in question. By keeping a record of how to backtrack though a directory tree, OS-9 can easily let you use pathlists such as ".../CMDS" and "../SYS", where OS-9 would move up two directories from your current location and then search for the CMDS directory. The "." entry refers back to the same directory and exists so that the current directory's descriptor sector can be found easily.

If an entry has a value of zero in the first character, that entry is unused. Whenever a new file is created, the first unused entry is filled. Likewise, when a file entry is being searched for, OS-9 begins with the first entry and continues to the last unless the entry is found. If there are a large number of empty, "padded" entries at the beginning of the directory, OS-9 must look through each of those, taking up extra time whenever the directory is searched.

File Structures:

Each file on the system consists of two parts: the file descriptor sector and the file's data segments. The file descriptor sector holds information such as the file's attributes (directory, single-user, public read, write, and execute, owner read write, and execute), user ID of owner, last modification date, link count, size, creation date, and segment list. The segment list can contain a maximum of 48 5-byte entries. Each entry has two parts: the beginning sector number of the file—the first 3 bytes, and the size of the segment in sectors—the last 2 bytes.

[1] see *File Structures* for more information about the descriptor sector

When a file is stored on a disk, it need not be stored in a contiguous, or unbroken sequence of sectors. The first part of a file may reside at one location on a disk, and the last part may be on a totally different part of the disk. Each of these "parts" is called a segment, and each segment is recorded, in order, in the files's segment list. As noted above, each entry in the list contains the start location of the segment and the length of the segment. A new segment may be created whenever OS-9 attempts to write past the end of the file. OS-9 first attempts to extend the last segment, if at all possible; otherwise, a new segment is created. When a file has more than one segment, OS-9 takes longer to access the file. Whenever a read is done from a different segment, OS-9 must refer back to the segment list to find the location of the new segment. It also takes time to move the disk drive's head to the file descriptor sector to examine the segment list and then to the new segment.

Theory of Operation:

The *optimize* program attempts to speed disk accesses in three ways: first, to convert each file into a single-segment file, to move unused directory entries to the bottom of the directory, and to "float" directories above files in each directory.

As mentioned above, single-segment files are accessed much quicker than multi-segment files because the disk head will always be near the file's contents. By moving the unused directory entries to the bottom of the directory, they will never be searched before a file is found. The last option, moving directories to the top of a directory, aids during pathlist searches. Whenever OS-9 receives a pathlist that does not refer to a file in the current directory (i.e. ".../CMDS/PROG/testprog" versus "testprog"), OS-9 must move through the directory structure to get to the file. If directory entries are at the top of a directory, OS-9 will not have to search through normal files before it finds the directory. Since directories are searched for more often than files, is makes sense to keep the sub-directories at the top of each directory.

Optimize will keep each file's status information (attributes, times, owner ID, etc.) intact while defragmenting. Note that *optimize* will not defragment directories. Another small quirk in the operating system is the fact the OS-9 allocates segments in multiples of 512K. If a file is larger than 512K, an optimize run will still leave the file with more than one segment.

OPTIMIZE UTILITY

Usage and Options:

The *optimize* utility actually modifies any disk or hard disk in order to speed disk accesses. Note that any RBF device, such as floppy disk drives, hard drives, and RAM disks, may be used with *optimize.* Any of the three optimization options outlined above may be performed in part or combination. In addition, only a particular directory's files and/or sub-directories may be optimized rather than the entire disk. Following is the *optimize* help display:

```
          OPTIMIZE - optimize disk storage

          Usage - optimize [opts] [dirname]

             opt=-? Help text
                 -c Don't compact directories
                 -d Do float directories
                 -f Don't defragment files
                 -l=n Set fragmentation limit to n
                 -s Suppress status messages
                 -x Don't check subdirectories
```

Options may be used in any combination, either individually (ex. "-s -x -d") or together (ex. "-sxd"). The directory name may appear anywhere: before, between, or at the end of the options. Note that when an entire disk it to be specified, simply use the root directory (ex. /d0, /r0, /h0, /d1). For instance, if an entire disk in drive /d0 is to be optimized, the command would look something like this:

```
   OS9: optimize /d0
```

To only optimize the CMDS directory in /d0, use:

```
   OS9: optimize /d0/CMDS
```

By default, files are defragmented and directories are compacted. In fact, it is necessary to compact the directories in order for the defragmentation algorithm to work; therefore, it is not possible to defragment files and not compact directories. Note that the compacting of a directory will not be visible to the user in any way and is highly advised whenever a directory is optimized.

Normally, directories are not floated to the top so that directory order will not be changed. Sometimes it is not desirable to have directories at the top of a directory, so it is left up to the user to decide when and where directories will be floated.

The *fragmentation limit* option controls which files, if any, will be defragmented. This option sets the number of segments a file can have and not be defragmented. If, for example, the fragmentation limit is set to 3, only files with more than 3 segments will be defragmented. Normally, the limit is set to 1; however, a user may set this to any reasonable value in order to perform a partial defragmentation run or for other technical reasons. For most optimize runs, it is a good idea to leave the limit at 1. Note that a fragmentation limit of 0 is possible—in this case, *every* file, even files that already have only one segment, will be defragmented. Note that this is an *extremely* time consuming and unnecessary process and could increase the time of an average run by a factor of 15 or more.

Throughout an optimize run, many status messages are displayed for the user's convenience. Certain situations, such as background execution, may require that these messages not be displayed. The *suppress status message* option will eliminate all messages which are normally sent to the standard output path, but will still send error messages which are output to the standard error path. Note that it is recommended that *optimize* has exclusive access to the directory and sub-directories begin used. Since *optimize* directly modifies the directories and files, it could conflict with other programs with a resultant loss of data.

The last option, *don't check subdirectories*, will scan only the files in a directory for optimization and ignore any subdirectories. Note that the initial directory will still be compacted and/or floated.

Note that *optimize* should not be used while running another process which might use any files on the disk that you are optimizing. A program reading or writing a file might interfere with the defragmentation process resulting in loss of data.

Warnings and Errors:

The following errors may be encountered while using the *optimize* module:

Initial Directory Error - could not open directory specified by the user
Device Name Error - could not determine device name for specified directory
Device Error - could not open device for specified directory
Disk Error - could not read device for specified directory
Directory Error - could not open sub-directory
I/O Error - disk input/output error
Deleting Error - trouble deleting temporary file 'o_'

In addition, two warnings may be seen which will not cause the operation of *optimize* to terminate but should be known to the user:

Could not defrag 'file' due to lack of disk space - in order to defragment a file, a duplicate, unfragmented copy is formed on the disk. If there is not enough free space to accommodate the entire file, the file is not defragmented.

Could not defrag 'file' due to existence of 'o_' in directory - A temporary file, 'o_', is repeatedly created during the defragmentation process. If a file with the name of 'o_' already exists in a directory, no files can be defragmented in that directory.

INQ UTILITY

Usage and Options:

The *inq* utility will allow you to determine the extent of fragmentation on your disk. The utility will recognize files and directories with no segments (empty files), one segment, and more than one segment. Other information is also collected and is explained below. There are no options available for the utility, and the only parameter needed is a pathlist to a directory or disk. To check an entire disk, use the root directory of the disk that you want to inspect (such as /d0, /h0, etc.). You may also use a subdirectory on a disk; if you do, only the files and directories in that directory will be included in the search. To check a disk in drive /d0, use:

```
OS9: inq /d0
```

To check only the CMDS directory in /d0, use:

```
OS9: inq /d0/CMDS
```

Here is an example listing of an inquiry executed on a hard disk:

```
INQ:  (dir /dd)

Total files:                2310
Files with no segments:     10      (0.43%)
Files with one segment:     2300    (99.57%)
Files with multi-segments:  0       (0.00%)
Average segments per file:  1.00
Maximum segments per file:  1

Total directories:          320     (8 sublevels)
Dirs with no segments:      0       (0.00%)
Dirs with one segment:      319     (99.69%)
Dirs with multi-segments:   1       (0.31%)
Average segments per dir:   1.00
Maximum segments per dir:   2

Average segments per unit:  1.00

Average padding per dir:    0.00
```

In addition to the number of non-, one-, and multi-segmented files and directories, the total number of files and directories is given, as well as average segments and the maximum segments per file or directory. Note that the average does not include files and directories that do not have any segments. *Average segments per unit* describes the average segments for files and directories together. *Average padding per directory* describes the average number of blank entries before the last valid entry in each directory.

Warnings and Errors:

The following errors may be encountered while using the *inq* module:

Initial Directory Error	- could not open directory specified by the user
Device Name Error	- could not determine device name for specified directory
Device Error	- could not open device for specified directory
Directory Error	- could not open sub-directory

Performance:

The *optimize* module uses an efficient algorithm to speed the defragmentation process. The following figures were obtained while optimizing a hard drive containing 12 megabytes of data contained in approximately 2,400 files.

Type	Time reference
No defragmentation[2]	12 minutes
Minimal defragmentation[3]	19 minutes
Complete defragmentation[4]	5 hours, 43 minutes

Using the same hard drive containing 12 megabytes of data contained in 2,400 files, it took approximately 6 minutes to complete an *inq* check.

[2] Optimization run with defragmentation disabled. Only directory compaction was active.

[3] Approximately twenty to forty files required defragmentation.

[4] Forced with a fragmentation limit of zero. Highly unusual under normal conditions.

Burke & Burke *Repack* utility

Overview:

The Burke & Burke *repack* utility is also a disk optimization utility but functions differently than the *optimize* utility. *Repack* takes each file's descriptor and segments and attempts to "pack" them as close to the beginning of the disk as possible. This increases disk performance because the drive head does not have to move as far between different files. Unfortunately, the repacking process frequently takes from twelve to twenty-four hours or more, which puts a significant strain on a hard drive. This is due to the fact that *repack* must scan the entire disk's contents between each pass.

Comparison:

Repack and *optimize* both attempt to optimize a disk but use two different methods, each of which is good alone or when used together. The first difference is that *repack* does not actually defragment files, but only attempts to pack all segments together as close as possible to shorten the physical distance that the drive head has to move between files. *Optimize* will actually defragment files into one segment, although it does not try to cram all of the files into any particular portion of the disk. Another small difference is the fact that *repack* will only work on drives formatted as one sector per cluster; *optimize* will work on any type of disk drive.

Suggestions:

If you own both *repack* and *optimize*, it would be beneficial to use them both together. However, it would be even better if you would use them as follows:

The first time that you use *optimize*, you will probably have many files that will be defragmented. Because of this, it may be best to execute *optimize* first, then use *repack* to consolidate the files near the beginning of the disk, and then re-optimize the disk. The reason for the second optimization is that often the *repack* utility will actually split a file into two segments in order to pack it with the other files. Running *optimize* a second time will allow *repack* to move the files together but still eliminate any segmented files afterwards.

On subsequent optimizations, it should only be necessary to run *optimize*, and will only take around fifteen minutes on a hard drive. It might be a good idea to optimize your hard disk every week if you use it often or do a lot of programming or disk-intensive work, or every month for an occasional user. A repack will only be necessary every few months, in which case you should run *repack* and then *optimize* to get the best results.

GETTING STARTED

Backup:

The first thing you should do when you receive any new program is to make a backup copy. By working with a backup, the original can be stored in a safe place in case anything happens to the backup. To make a complete backup of the *Optimize Utility Set 1* package, it is only necessary to copy the *optimize* and *inq* modules contained on the original disk. One common procedure for making backups is to format a new disk and then use the OS-9 *backup* command to copy the entire master disk to the new disk. The only problem with this method is that the new disk must be formatted identically to the master disk (i.e. If the master disk is a 40-track double sided disk, the new disk must be formatted identically, not, for example, as an 80-track single sided disk). The optimize modules come on a 40-track single sided disk, and this backup method may be used to make a copy. However, it is much easier to simply format a new disk and then copy each file to the new disk. To format a disk, insert a new disk into drive /d0, and type:

```
OS9: format /d0
```

OS-9 will automatically complete the format, asking you only for a disk name at one point. After this disk is formatted, you can then copy the files. If you have a two drive system, put the master disk in drive /d0, and the new disk in drive /d1. Type the following to copy both files:

```
OS9: copy /d0/optimize /d1/optimize
OS9: copy /d0/inq /d1/inq
```

If you have only one drive, use the single-drive modifier with the copy command:

```
OS9: copy /d0/optimize /d0/optimize -s #32k
OS9: copy /d0/inq /d0/inq -s #32k
```

You can then put away your master disk in a safe place and use your new backup.

Use:

To actually use the modules, you must either load them into memory or have them in your *current execution directory*. Loading them into memory is convenient if you are going to optimize a large number of disks, but they will remain in memory until unlinked, taking up valuable memory space from anything else that you might be doing.

If you have a hard drive, it might be wise to copy both modules into your hard drive's CMDS directory. In this way, they will always be instantly available. If you use a floppy-drive only system, it might be easiest to simply insert your backup copy into drive /d0 when you need it and change the execution directory to drive /d0:

```
OS9: chx /d0
```

The other alternative is to type a complete pathlist to the optimize program instead of changing the execution directory. Simply type:

```
OS9: /d0/optimize /dd
```

If you have a single drive system, you will have to load the module into memory before using it. This is because OS-9 reads the module into memory from disk every time you use it. If the module must be read from the program disk, you cannot optimize or check another disk. Type:

```
OS9: load /d0/optimize /d0/inq
```

while the program disk is in drive /d0. Now the programs will remain in memory for your use until you unlink them:

```
OS9: unlink optimize; unlink inq
```

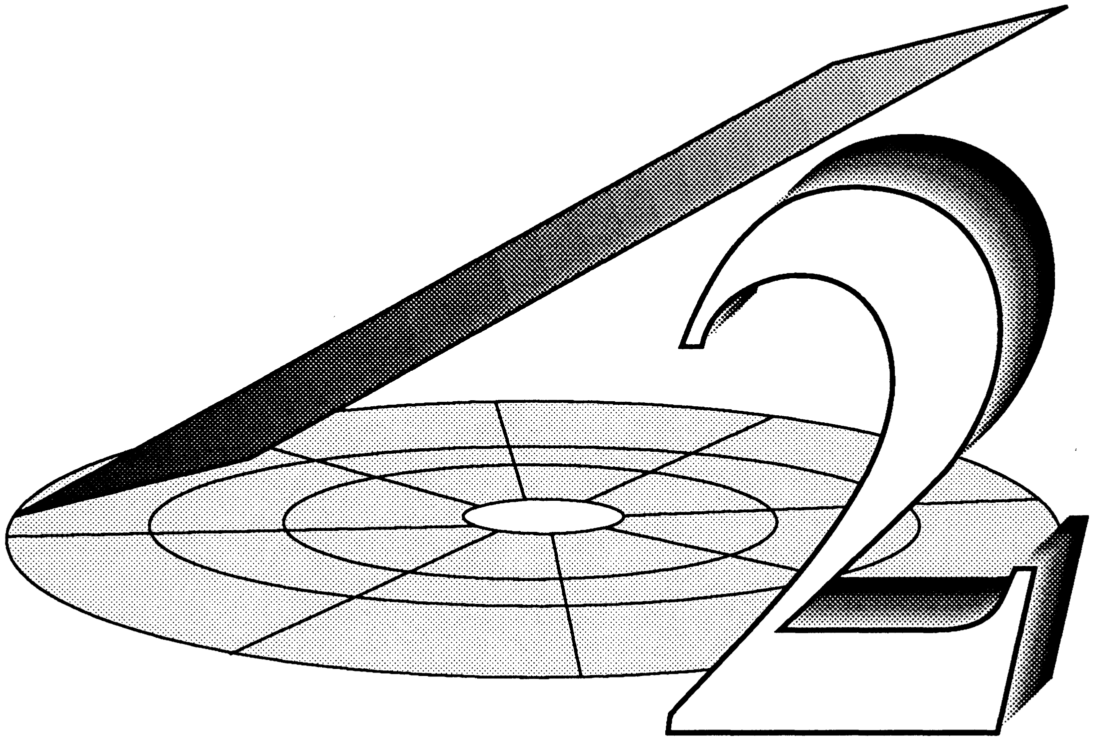
If you have any problems or errors, try consulting your OS-9 manual. An error will usually provide you with enough information to quickly correct the problem. If you have any problems that you cannot find a solution to, do not hesitate to write to *JWT Enterprises* with your problem. Be sure to include a description of your setup and conditions of your setup at the time that the problem occurred.

THANK YOU

Thank you for purchasing the *Optimize Utility Set 1* package. We hope that it will help to improve the performance of your disks. Its companion package, the *Optimize Utility Set 2*, contains two other utilities to assure that the file structure of your disks in intact. The first will check all directories and subdirectories to make sure that the hierarchal structure is intact. It is possible for child directories to not point back to the parents or even themselves, and this directory utility will recognize and correct any problems. The other utility will check the disk to make sure that the disk allocation map is intact, that no space is reserved for nonexistent files, and that all files are correctly registered in the map. If a file is not registered, it is possible and probably that OS-9 will write over parts of the file, destroying the information in the file. And, space reserved for nonexistent files cannot be used for other files, even though there is nothing stored in that area. To solve any problems, the utility will recognize and correct any discrepancies in the disk allocation map. It is recommended but not necessary that these utilities both be used on a disk before an optimization run. If there are any problems with the directories or disk structure, it is possible that *optimize* might write over files or become caught in a loop because of invalid directory pointers. Though these kinds of errors are very infrequent, it is a good idea to keep these problems in check.

Optimize Utility Set 2

Documentation



JWT Enterprises
5755 Lockwood Blvd.
Youngstown, OH 44512

Optimize Utility Set 2

A Product of JWT Enterprises

INTRODUCTION

This *Utility Set* is designed to compliment the *Optimize Utility Set 1* by verifying a disk's integrity to avoid problems with the *Optimize Utility Set 1* and the normal operation of the disk. Two utilities are contained in this package: *dircheck* and *damcheck*. *Dircheck* will check and correct any problems with the directory structure on a disk. *Damcheck* is similar in operation to *dircheck*, except that it works with the disk's disk allocation map to account for every file on the disk. This manual assumes a basic familiarity with the OS-9 Operating System. For more help, see the *Getting Started* section at the end of the manual for additional information.

COPYRIGHT NOTICE

DISCLAIMER

Every effort has been made to insure the reliability and accuracy of this manual and the accompanying software. *JWT Enterprises* does not assume any liability for any damage incurred though the use of the documentation or software. The product is sold on an as-is basis, and *JWT Enterprises* is not responsible for any damage done to the product except that which is covered in the limited warranty.

LIMITED WARRANTY

The user must be a registered owner of the product in order to take advantage of the warranty. *JWT Enterprises* warrants the product from defects in production or damage due to transportation for a period of sixty (60) days after the date of purchase. This warranty is limited to the replacement of the product at the expense of *JWT Enterprises* during this period at the discretion of *JWT Enterprises*. This warranty excludes software defects and defects caused by tampering, negligence, and abuse of the product.

BACKGROUND INFORMATION

The following information is a more detailed description of directory structures, file structures, and file fragmentation. For further reference, the RBF Manager section of the *OS-9 Technical Reference* manual contains the actual tables and technical data for the more advanced user. This information intends to familiarize the user with the basics of the above mentioned topics.

Directory Structures:

Each OS-9 directory can actually be thought of as a file with 32-byte entries for each file and sub-directory contained in the directory. The directory has a file descriptor sector and segments just as a regular file has, and directory entries and file entries are stored the same way in a directory. The only difference between a file and a directory is that the "D" attribute is set in the directory's attributes.

Each 32-byte entry is divided into two parts; the first 29 bytes contain the name of the file or directory, and the last 3 bytes contain the disk sector number of the file or directory's descriptor sector[1]. The last character of the filename always has the value 128 added to it—in other words, the most significant bit of the last character is always set.

The first two entries in any directory are always ".." and ".". The first entry, "..", refers to the *parent directory*. This parent directory is the directory that contains the entry for the directory in question. By keeping a record of how to backtrack though a directory tree, OS-9 can easily let you use pathlists such as ".../CMDS" and "../SYS", where OS-9 would move up a directory from your current location and then search for the CMDS directory. The "." entry refers back to the same directory and exists so that the current directory's descriptor sector can be found easily.

If an entry has a value of zero in the first character, that entry is unused. Whenever a new file is created, the first unused entry is filled. Likewise, when a file entry is being searched for, OS-9 begins with the first entry and continues to the last unless the entry is found. If there are a large number of empty, "padded" entries at the beginning of the directory, OS-9 must look through each of those, taking up extra time each time the directory is searched.

File Structures:

Each file on the system consists of two parts: the file descriptor sector and the file's data segments. The file descriptor sector holds information such as the file's attributes (directory, single-user, public read, write, and execute, owner read write, and execute), user ID of owner, last modification date, link count, size, creation date, and segment list. The segment list can contain a maximum of 48 5-byte entries. Each entry has two parts: the beginning sector number of the file—the first 3 bytes, and the size of the segment in sectors—the last 2 bytes.

[1] see *File Structures* for more information about the descriptor sector

When a file is stored on a disk, it need not be stored in a contiguous, or unbroken sequence of sectors. The first part of a file may reside at one location on a disk, and the last part may be on a totally different part of the disk. Each of these "parts" is called a segment, and each segment is recorded, in order, in the file's segment list. As noted above, each entry in the list contains the start location of the segment and the length of the segment. A new segment may be created whenever OS-9 attempts to write past the end of the file. OS-9 first attempts to extend the last segment, if at all possible; otherwise, a new segment is created. When a file has more than one segment, OS-9 takes longer to access the file. Whenever a read is done from a different segment, OS-9 must refer back to the segment list to find the location of the new segment. It also takes time to move the disk drive's head to the file descriptor sector to examine the segment list and then to the new segment.

Disk Allocation Map:

The Disk Allocation Map consists of one or more sectors starting at Logical Sector Number one on each disk that record which portions of the disk contain data and which parts are empty. Each bit in the map corresponds to one cluster, starting at Logical Sector Number zero. Except for large hard disks, each cluster is equal in size to a sector. If the bit is on, that sector is used; likewise, if the bit is off, the sector is unused and available. The length of the map is dictated by the total number of sectors on the device and is established when the disk is formatted.

Normally OS-9 takes care of maintaining the DAM, although it is possible to destroy the integrity of the map by turning off the computer while files are open. Using the file descriptor sectors of every file on the disk, it is possible to reconstruct a map by scanning each file and setting the appropriate bits in the map.

Problems:

Both the directory structure and the DAM can become corrupted due to a number of things including system lock-ups, power failures, rampant programs, and many other errors. These do not occur often, but can cause serious problems that get progressively worse as time goes on and more accesses and changes are made. Directory tree errors will most often show up as Error 214 or 253 or an unexpected directory being accessed. DAM errors can range from areas of the disk becoming unusable to files being overwritten because the DAM indicates that the space containing the file is supposedly free space. OS-9 cannot automatically detect these problems and they will not show up (especially in the case of the DAM) until damage is done. Whenever a problem such as those outlined above occurs, it is a good idea to run *dircheck* and *damcheck* on any device that was being used at the time. Catching a problem as quickly as possible will prevent widespread damage to your data.

Dircheck Utility

Usage and Options:

The *dircheck* utility allows you to check the integrity of the directory structure on any device or portion of a device. Basically, *dircheck* will make sure that the parent and current pointers in each directory point to the correct directory and correct any problems as an option. The *dircheck* help display is as follows:

```
          DIRCHECK - check directory structure

          Usage - dircheck [opts] [dirname]

             opt=-? Help text
                 -c Correct any errors
                 -l Don't list errors
                 -r Include root directory
```

Options may be specified in any combination, either individually (ex. "-c -l -r") or together (ex. "-clr"). The directory name may appear anywhere: before, between, or at the end of the options. Note that when an entire disk it to be specified, simply use the root directory (ex. /d0, /r0, /h0, /d1). For instance, if an entire directory structure on drive /d0 is to be checked, the command would look something like this:

```
   OS9: dircheck /d0
```

By default, *dircheck* will only report any errors that it finds, but will not correct them. By using the '-c' modifier, every error found will be corrected. If error reports are not required, the '-l' modifier will disable the output. Note that '-c' and '-l' in combination effectively do nothing, even though *dircheck* will still scan the entire directory structure.

The one loophole in the directory structure is the root directory of each disk. Normally, the '.' and '..' entries both point to the root directory. OS-9 knows where this directory is by keeping its starting sector number in a special area of the first sector on the disk. By default, *dircheck* does not check the validity of the root directory's pointers because the algorithm scans only the directory structure and not the first sector of the disk. However, the '-r' option will invoke another routine that will check the root directory's pointers. Note that if the '-r' option is used with a directory that is not the root directory, errors will be reported because the specified directory's pointers will not match up with the root directory's pointers. In conclusion, use the '-r' option whenever you are specifying the root directory of a disk.

Warnings and Errors:

The following errors may be encountered while using the *dircheck* module:

Initial Directory Error - could not open directory specified by the user
Device Name Error - could not determine device name for specified directory
Device Error - could not open device for specified directory

In addition, non-fatal errors are reported which will not cause the operation of *dircheck* to terminate:

Directory Error 'directory' - if there is an error opening or accessing a directory, that particular directory will not be checked and/or fixed. All other directories will be processed except for any subdirectories in the directory with the error.

Root Directory Error - if there is an error opening or accessing the root directory, it will not be checked and/or fixed. However, this utility is isolated from the main check and other subdirectories may be processed unless a regular *Directory Error* is reported.

The following errors are reported due to flaws in the directory structure and are fixed if the '-c' option is used with *dircheck*:

Parent Link Error 'directory' - the directory does not properly point to its parent directory. For example, a 'cd ..' while in this directory will not return you to the proper parent.

Self Link Error 'directory' - the directory does not properly point to itself. For example, a 'dir .' will not give a directory of the current directory.

Root Parent Link Error - The root directory's '..' entry should point to itself; however, it does not.

Root Self Link Error - The root directory's '.' entry does not point to itself.

DAMCHECK UTILITY

Usage and Options:

The *damcheck* utility compares the disk allocation map with the current file structure on the disk to make sure that all files in the file structure are properly logged in the disk allocation map and that all sectors that are allocated in the map are also in the file structure. In addition, media errors (I/O errors) on the disk can be found and compensated for during the process. Other options include listing any files that are located in questionable sectors. The help command for the *damcheck* utility is as follows:

```
DAMCHECK - disk allocation integrity check

Usage - damcheck [opts] devname

   opt=-? Help text
       -c Correct DAM
       -e Don't check/account for bad sector errors
       -l Don't list damcheck messages
       -p List pathlists of any questionable files
       -s List numbers of any questionable sectors
       -w=dir Specify workfile directory
```

By default, *damcheck* does not correct the disk allocation map of the device unless the '-c' option is specified. This insures that the disk will not be modified in any way unless the user explicitly specifies it.

Another default is the checking for bad media errors. Any sector that is unreadable due to a disk aberration should normally be mapped out of the allocation map so that there is no attempt to write a file to the damaged sector. If for some reason you do not wish to account for media errors, specify the '-e' option.

Unless the '-l' option is given, *damcheck* will periodically report about what it is doing. These stages are: "Initializing check files", explained below; "Cross-referencing files", formulating a correct allocation map; "Checking for media errors", checking and recording bad sectors; "Writing corrections to DAM", if appropriate; "Questionable sectors", if applicable and explained below; "Questionable files", if applicable and also explained below. Note that all error messages are sent to the standard error path regardless.

The '-p' and '-s' options allow *damcheck* to report exactly what structure errors and warnings it has detected. The '-s' option will list any sector that is in question and report the problem next to the sector number. Note that this information, as well as that for the '-p' option, will be displayed even if the '-l' option is given. The '-p' option will display the filepath for every sector that has been found to be in error (except for the first error listed below). By using the sector numbers obtained by the '-s' and '-p' options, it is possible to link the problems with the proper files in case the files need to be checked for damage and/or copied to a safe location. Note that the '-s' option can substantially increase the running time and the '-p' option can double it.

There are ten possible errors/warnings, each with different meanings:

Mapped out but not in file structure - This sector is mapped out of the disk allocation map, so that it cannot be written over by another file. However, no file on the disk uses this sector and no media error has been detected. There is nothing threatening about this particular error, but it can be fixed easily using the '-c' option as it is simply taking up space that could otherwise be used.

Used in a file, not mapped out of disk - This is almost the opposite of the above problem, in that this sector contains part of a file but is *not* marked as being in use. This means that any disk write could possible use this sector and destroy the original contents of the file. If this condition has occurred while writes were made, it is possible that the file may already be corrupted. It would be a good idea to use the '-p' option and match this sector number with the file it belongs to, and check this file for any damage. If there is any damage, it is irreversible and the file should either be deleted if it is executable code or salvaged if it is text. Note that the '-c' option will repair this problem and can be done before the file is checked.

Used in multiple files - This presents a very bad problem, in that it indicates that two separate files claim ownership of the same sector. The best method to follow is finding the two (or more) files' pathlists using the '-p' options and matching the sector error to any file corresponding to the same sector. Copy each of these files to a safe location (it can even be on the same disk as long as no other errors are present or all errors have been fixed with the '-c' option). Delete both of the affected files, and the disk will be back to normal (the sector will be "deleted" twice, but it will still be cleared in the allocation map). Note that it is important to check the two files, as there is a very good probability that one or both of the file files is corrupted, although there is a possibility that both files may be unharmed.

Used in multiple files, not mapped out of disk - This is basically a combination of the above two problems. Follow the procedure for "Used in multiple files", but keep in mind that the sector may have been overwritten and corrupted both files.

Bad sector, not mapped out of disk - This sector has been found to have a media error, making it unusable. However, it is not mapped out of the disk, and a write operation may try to use this sector as storage. There is no damage to any files; however, the '-c' option should be used to map this sector out of the disk so that no attempts will be made to use this sector in the future.

NOTE: Bad sector mapped out of disk - This sector has a media error, is not used in any files, and is properly mapped out of disk. Rather than an error, this is a simple warning to inform you of the disk error. The '-c' option would not apply here, as the sector is already mapped out of the disk.

Bad sector, used in a file - This sector has been found to be bad and used in a file. This could be caused by a sector being corrupted in storage or due to age. To be completely effective, the file should first be deleted, then *damcheck* should be run with the '-c' option to map this sector off of the disk. Note that it might not be possible to reconstruct the file because the error may be encountered in the middle of the file, although the error may be beyond the end of the file.

Bad sector, used in a file, not mapped out of disk - This is essentially the same as the last error, although the possibility of it being overwritten is of no consequence, because the sector is bad. However, this should be mapped out of the disk so future writes will not use the sector. Follow the last error's procedure for possible recovery of the file.

Bad sector, used in multiple files - Again, follow the procedure for multiple file allocation errors, taking into account that the error may be in the middle of the files making recovery impossible. After following the procedure for multiple files, use the '-c' option to correct the disk allocation map.

Bad sector, used in multiple files, not mapped out of disk - Again, follow the bad sector, multiple files procedure.

The last option is the '-w' option. Four or five workfiles, or checkfiles, are created in the default data directory unless the '-w' option specifies a replacement. It is okay to locate these files on the disk being checked, but in any case, there should be enough space for the check files, or an error will result.

It is essential that *damcheck* is the only process using the disk while the check is completed. If any file is modified on the disk, the check files will no longer be accurate and random errors may be reported.

In the event that there is a major problem with the file structure, it is possible that the program may take longer that normal and you may wish to terminate it prematurely. The program may be stopped at any time with the BREAK key unless *damcheck* is writing to the DAM.

Damcheck should not be used on boot disks due to the fact that there are certain sectors that are intentionally mapped out of the disk. Although it is possible to check the disk, there will be approximately eighteen sectors that are "allocated but not in file structure" and should not be corrected.

Warnings and Errors:

The following errors may be encountered while using the *damcheck* module:

No Device Specified - *damcheck* needs a device in order to operate
Initial Directory Error - could not open directory specified by the user
Device Name Error - could not determine device name for specified directory
Device Error - could not open device for specified directory
Not 1 sector per cluster - *damcheck* only works on one s.p.c. disks
Clearfile Error - the clear checkfile could not be created
NAfile Error - the n/a checkfile could not be created
Dupfile Error - the duplicate checkfile could not be created
Errfile Error - the error checkfile could not be created
Workfile Error - the work checkfile could not be created
Checkfile Write Error - a checkfile had a write fault, possibly media full
Checkfile Read Error - a checkfile had a read fault
Error Reading DAM - error while reading DAM off of disk
Error Writing DAM - error while writing corrected DAM to disk
Directory Error 'directory' - could not open/access sub-directory; note that this error will not halt execution of the program and could result in "sector allocated but not in file structure" because the files in this directory could not be checked.

Performance:

The following figures were obtained while optimizing a 30-megabyte hard drive containing 14 megabytes of data contained in approximately 2,800 files and 400 directories.

Type	Time reference
Dircheck	11 minutes
Damcheck[2]	37 minutes
Damcheck w/media check	1 hour, 3 minutes
Damcheck w/media check, questionable sectors	1 hour, 45 minutes
Damcheck w/media check, questionable sectors, questionable files	2 hours, 45 minutes

[2] All *damcheck* tests were conducted with the work files on a ramdisk

GETTING STARTED

Backup:

The first thing you should do when you receive any new program is to make a backup copy. By working with a backup, the original can be stored in a safe place in case anything happens to the backup. To make a complete backup of the *Optimize Utility Set 2* package, it is only necessary to copy the *dircheck* and *damcheck* modules contained on the original disk. One common procedure for making backups is to format a new disk and then use the OS-9 *backup* command to copy the entire master disk to the new disk. The only problem with this method is that the new disk must be formatted identically to the master disk (i.e. If the master disk is a 40-track double sided disk, the new disk must be formatted identically, not, for example, as an 80-track single sided disk). The optimize modules come on a 40-track single sided disk, and this backup method may be used to make a copy. However, it is much easier to simply format a new disk and then copy each file to the new disk. To format a disk, insert a new disk into drive /d0, and type:

```
OS9: format /d0
```

OS-9 will automatically complete the format, asking you only for a disk name at one point. After this disk is formatted, you can then copy the files. If you have a two drive system, put the master disk in drive /d0, and the new disk in drive /d1. Type the following to copy both files:

```
OS9: copy /d0/dircheck /d1/dircheck
OS9: copy /d0/damcheck /d1/damcheck
```

If you have only one drive, use the single-drive modifier with the copy command:

```
OS9: copy /d0/dircheck /d1/dircheck -s #32k
OS9: copy /d0/damcheck /d1/damcheck -s #32k
```

You can then put away your master disk in a safe place and use your new backup.

Use:

To actually use the modules, you must either load them into memory or have them in your *current execution directory*. Loading them into memory is convenient if you are going to optimize a large number of disks, but they will remain in memory until unlinked, taking up valuable memory space from anything else that you might be doing.

If you have a hard drive, it might be wise to copy both modules into your hard drive's CMDS directory. In this way, they will always be instantly available. If you use a floppy-drive only system, it might be easiest to simply insert your backup copy into drive /d0 when you need it and change the execution directory to drive /d0:

```
OS9: chx /d0
```

The other alternative is to type a complete pathlist to the program instead of changing the execution directory. Simply type:

```
   OS9: /d0/dircheck /dd
```

If you have a single drive system, you will have to load the module into memory before using it. This is because OS-9 reads the module into memory from disk every time you use it. If the module must be read from the program disk, you cannot optimize or check another disk. Type:

```
   OS9: load /d0/dircheck /d0/damcheck
```

while the program disk is in drive /d0. Now the programs will remain in memory for your use until you unlink them:

```
   OS9: unlink dircheck; unlink damcheck
```

If you have any problems or errors, try consulting your OS-9 manual. An error will usually provide you with enough information to quickly correct the problem. If you have any problems that you cannot find a solution to, do not hesitate to write to *JWT Enterprises* with your problem. Be sure to include a description of your setup and conditions of your setup at the time that the problem occurred.

THANK YOU

Thank you for purchasing the *Optimize Utility Set 2* package. We hope that it will help to improve the performance of your disks. Its companion package, the *Optimize Utility Set 1*, contains the actual optimization utility and any inquiry utility that can determine the extent of fragmentation on a disk. Using the two sets together provides a unique package that will maintain the integrity of your file system and improve performance at the same time.