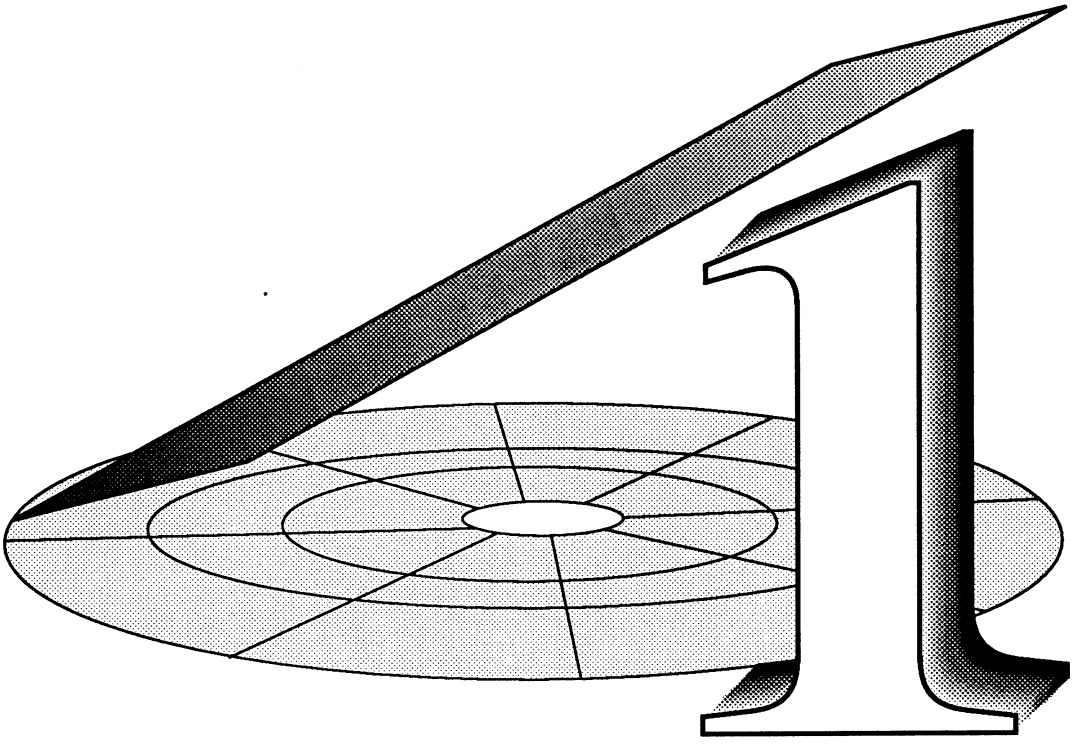
Optimize Utility Set 1

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.