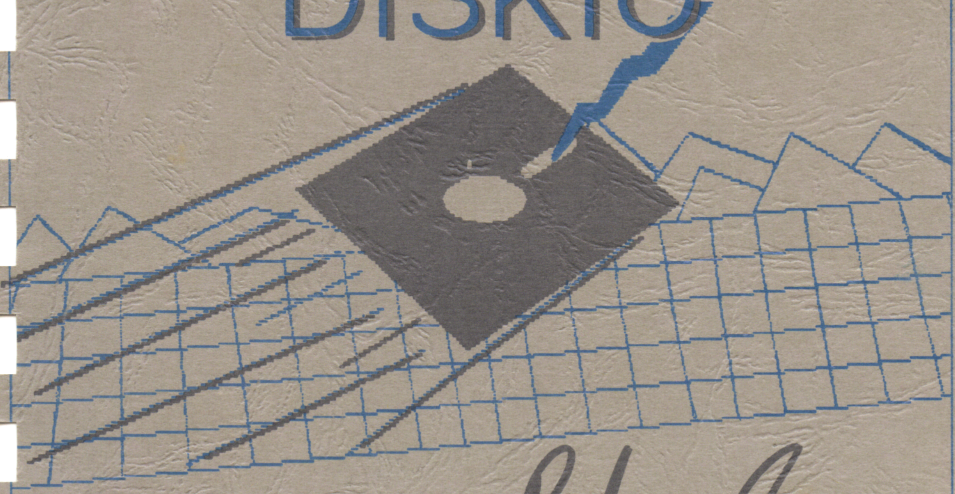
DISKIO

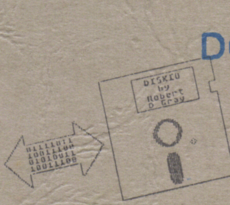


By: Robert Gray

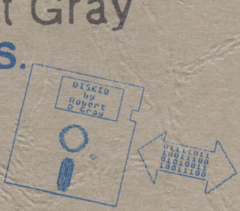
$32.95

Robert Gray

DON'T TAKE CHANCES.



DISKO can help you
get your data,
back on the right track



TM

Omega Soft

DISKIO
BY
Robert D. Gray

Omega Soft
Po Box 139
Harrells, NC 28444

(919) 532-2359

Manual version 1.1

ACKNOWLEDGEMENTS

I would like to thank David young of CDY consulting and Charles Marslett, the author of MYDOS, for their technical assistance and support during this project. If not for their help DISKIO would not be a reality.

Thank you,
Robert Gray

WARNING:

TABLE OF CONTENTS

APPENDICES

INTRODUCTION

DISKIO is a program which allows the user to do practically anything to nearly any kind of disk. It is compatible with single, double, and enhanced density drives. Double sided, 8 inch and hard drives as well. It will also use the MYDOS, DOS 2.5, and 8K OMNIMON/OMNIVIEW ramdisk handlers to access practically any ramdisk available for the 8-bit Atari computers. Both single and double density disks can be examined in hexadecimal, character, or internal screen character mode. The hexadecimal data from a double density sector is all displayed on a single screen instead of flipping back and forth between the first 128 bytes and the last 128 bytes of the sector.

Since DISKIO is a binary file it can be copied to the ramdisk and loaded almost instantly from DOS. If during a programming project DOS was used to delete the wrong file, DISKIO could, with as little as three key presses, undelete it and return the user to productivity in less than thirty seconds.

The experienced programmer, as well as the inexperienced will appreciate the straight forward "just enter the number" method for changing link sectors, file numbers, byte counts, file start sectors, and lengths. Everything that the computer, instead of the user, can remember is taken advantage of. Never again be just a little unsure about which byte does what in a load vector, DISKIO will always remember these little things for you

DISKIO has some very powerful and useful file commands. It is able to fully reconstruct an erased, scrambled, or destroyed directory recovering most if not all of the lost files. The trace command identifies binary file load vectors, as well as basic save file parameters.

Almost all of the standard DOS commands as well as a few new ones are available on the directory screen. Two drives can be made to continuously, without prompting, format new disk. The quick format command clears a disk in a matter of seconds instead of taking the time for the complete format process. Subdirectories are also supported. There is also a command to automatically correct the infamous "File Number Mismatch" error.

The write menu command creates a binary file menu on a blank disk which will load files very quickly without DOS. This menu will be considered public domain.

DISKIO is a very useful and complete disk access program it pulls together into one package a lot of utilities that never seem handy when they are needed.

THE MAIN SCREEN

The main screen displays the data contained in a single or double density sector. This data can be seen in either hexadecimal, character, or internal screen character mode. DISKIO is able to put the entire contents of a double density sector in hexadecimal on a single screen. This is the reason for the alternating stripes of normal and inverse pairs of numbers.

The sides and top of the sector data field contain decimal (left) and hexadecimal (right) referance numbers to make finding a specific byte in the data field a simple matter.

The number of the sector currently being accessed is displayed in decimal and hex in the upper left corner of the screen. The three link bytes are also decoded and the link sector, file number, and byte count are all displayed in decimal and hex. (NOTE: This information is meaningless when viewing a boot sector or sequential sector. See appendix 1) When accessing a disk with more the 1023 sectors the file number information is omitted. The reason for this is double sided disks, etc., use 16 bit file links (see appendix 3). If the sector or link number exceeds 999 the it will be displayed in decimal only. The source and destination refer to which drive will be read from and written to respectively. These can both be the same drive or different drives from 1 to 8 (8 specifies Ramdisk). The disk in these drives must both be the same density. The density is indicated by "S" for single and "D" for double. The sequence/link mode is displayed in the upper right corner and is toggled by COTROL-L command. When in link mode the next sector command (+) follows the file links. Also when in this mode the rotate bytes command (</>) leaves the three link bytes alone. When using the search command (S) in link mode, the three link bytes are not examined in the search since it is unlikely that you will be searching for a string that involves link bytes.

MAIN SCREEN COMMANDS

E - Exit to DOS this command returns to DOS if verified by a "Y". If DOS is not in memory use of this command will send the Atari into funny byte land.

? - Help screen displays a short command summary of all the main screen commands.

R - Read sector asks for a sector number then reads and displays the selected sector.

W - Write sector allows the data currently on the screen to be written to the current sector by pressing "Y". The data can be saved to a different sector without affecting the current sector on the disk by simply entering the new sector number. The write can be aborted by pressing "N".

P - Print screen prints the data currently on the screen. Notice the print command does not waste your paper or time by printing the entire screen, only the relevent data. No menus or blank lines are printed.

T - Hex/char toggle changes the data field from hexadecimal to character and vise - versa. If when toggling to the character screen the OPTION key is held down, the character data will be displayed as internal screen characters instead of ATASCII. It can then be changed with no conversions, by using the change "C" command, and rewritten directly to the disk. This is very useful when looking for strings which you suspect are poked rather than printed, to the screen.

D - Directory screen displays the first sector of the root directory (sector 361) with the directory commands. See the directory screen section of this manual.

S - Search for a string up to eight bytes long. In hex mode enter the numbers to be searched for in hex, with no $ signs, and with a space separating each pair: (xx xx xx ...). In actual ATASCII characters, not hex numbers.

EX: HEX - "00 01 02 03 04 05 06 07"
char - "2STRINGS"

C - Change bytes in a sector. To move the cursor in this mode, use the CONTROL - arrow keys. To cause the CONTROL - arrow keys to print, first press ESCAPE key. In hex mode do not worry about the rows of normal and inverse charecters, just keep typing, DISKIO will handle it. When finished changing a sector, hold the OPTION key and press "W". This will allow the new data to be written to the disk. To abort the change, hold the OPTION key and press any other key.

L - Link modify allows changes to the link sector, file number, and byte count by simply typing the new number. If any of these do not need to be changed, simply enter an "N" at that question. This allows the lnk bytes to be changed without getting into any math.

(NOTE: When accessing a disk with more than 1023 sectors, file numbers are not used, therefore that quistion will not be asked.)

CTRL-L - Seq/Link toggle In SEQ mode the +/- keys will read the next or last sector in numerical order. In LINK mode the + key will follow the file links. The - key will read the previous sector in numerical order.

F - File Maintenance Screen File trace and recovery commands. See the File Maintenance section of this manual.

M - Disk Map reads sector 360, displays the number free sectors, and the sector map of the disk. Unused sectors are represented by ones while sectors that have been allocated for use by a file or system information are represented by zeros. Notice the map of the formatted, empty disk shows the boot map, and directory sectors as allocated, or unavailable sectors. The map command displays only sectors 1 - 719 on high capacity disks. Holding the OPTION key while selecting this command will cause the map screen to be echoed to the printer

O - Drive Options allows the user to set which drive or drives will be origin and destination, (read from and written to). DISKIO will access drive numbers 1-8. The disk in the origin and destination must always be the same density. The default density is single. Always use the "O" before accessing a double density disk, and again will display as single density, simply because they are single density (see appendix 1). When the "O" command is selected the origin drive number is asked for. If 1-7 is selected the density (single, double or enhanced) will be entered. The next question is weather or not the drive is a hard disk. If it is not a hard disk, the next selection is single or double density, and remember the origin and destination must be the same density.)

Then number of tracks will be selected.
ANSWER: 1 for 35 tracks/side
2 for 40 tracks/side
3 for 77 tracks/side (8 inch drive)
4 for 80 tracks/side

Then the drive read/write head step rate:

8 inch	5 1/4 inch
0 for 6 ms	3 ms

1 for 12 ms	6 ms
2 for 20 ms	10 ms
3 for 30 ms	15 ms

Finally the destination drive number will be asked for and the same set of questions will be repeated.

If the selected drive is "8" for either the origin or destination then it is assumed to be a ramdisk. DISKIO supports the ramdisk handlers contained in MYDOS, 8K OMNIMON, OMNIVIEW, and DOS 2.5. This allows its use with a 130XE, AXLON ramdisk, or any of the available upgrades for the 800 or 800XL. Next is whether or not the ramdisk has more than 720 sectors. The 130XE does not, so respond with a "N" This answer will determine if the file links are expected to contain a file number (see appendix 3). Note Ramdisks are always considered single density and remember the orgin and destination must be the same density.

Excluding the questions involving ramdisk handlers (which expects a M, O, or D) the inverse character in the available selections for each question is default. So if drive 1 were to be the origin and destination, single density, single sided, 40 tracks/side, and have a head step rate of 12 ms, then the "O" command could be selected and RETURN held down all the way through the menu.

Z - Fill one sector, or range of sectors with zero or any other number. If the default of zero is desired then type "Y", any other number (0-255) may be entered in decimal or hex.

K - Copy Sectors from origin to destination, even if they are the same disk. This is a little slow because only one sector is copied at a time, but it gets the job done.

X - Print File Sectors in character and hex format. Given the starting sector of a file, this command automatically prints each sector to the printer, in character and hex format up to and including the EOF sector.

+/- - Next/Previous sector. When in link mode the next (+) command follows the links (see CNTL - L command).

</> - Rotate Bytes within a sector. When in link mode the three link bytes are left out of the rotation. This command is useful when adding bytes to the beginning of a sector.

EX: When generating a new file with the File maintenance "G" command, the new file name will probably not be in the right position in the directory (file number mismatch see appendix 1). Read the first sector of the file and determine the proper file number. The read the directory sector conatalning the new file name and rotate it to the proper file number position. (This could also be done with the directory screen correct file number "C" command) for another example of the use of this command see appendix 8.

$/#/% -Two Byte Number Base Conversion. First select the base to convert to, then enter the number to be converted. For example to convert decimal 65535 to binary, type : % then type #65535 RETURN (NOTE : when converting from binary to decimal or hex you must enter 16 binary elements with no spaces (with preceding zero).

```
EX: $
   %0000100000000000 RETURN
   $0800
```

DIRECTORY SCREEN

There are eight directory sectors, each containing eight filenames For each filename there is a starting sector, number of sectors (the length of the file), and a status byte. (See appendix 1-5) Each of these 64 possible files have a file number, 0-63. This number is not saved on the disk, but is determined by the location in the directory (the first is 0, the second is 1, ...). This file number should match the file number in the link bytes of each sector in the file. When these numbers do not match a "file number mismatch" error is generated. (This can be corrected using the Directory Screen "C" command.) The file number, status byte, number of sectors, starting sector, and directory screen. If a sector number exceeds 999 it will be displayed in decimal only. All of the directory commands that pretain to a specific file handle file selection the same way. After selecting the command, use any key to move the pointers to the desired filename, then press RETURN. To abort this command press "N".

1

DIRECTORY SCREEN COMMANDS

D - Delete a file from the disk. The file is traced and the sectors it uses are deallocated, or made available for use. on the sector map. The number of free sectors on the disk are changed, to reflect the space left by the deleted file. Then the status byte, in the directory, is changed to an inverse heart (deleted file status). As far as DOS is concerned the file is gone and the space it occupied is available for use. The next file saved to the disk will possibly overwrite the data that was just deleted.

U - Undelete a previously deleted file. As seen in the delete a file explanation, the delete command does not destroy any of the file's data, it merely makes the file space available for use. This is also true for the DOS delete command. By reversing the steps mentioned above, a deleted file may be brought back, or undeleted. This, of course, may not work if another file has been saved to the disk since the delete was executed. But it is still worth a try. This function can also be used to repair the VTOC.

R - Rename like the DOS command, simply renames a file. Enter the new name in the standard eight letter filename, a period, then a three letter extension (if desired). This rename command does not care what a file is named, lower case , inverse letters, CONTROL characters, or whatever may be used. DO BE AWARE that DOS will not recognize these strange filenames! (But the DISKIO binary file menu will. This could be used to allow files to be run, but not copied from a disk.)

L - Locks a file by changing the status byte to a locked file status using the appropiate status byte for the given file type (See Appendix 5).

X - Unlocks a file by changing the status byte to an unlocked file status using the appropiate status byte for the given file type (See Appendix 5).

F - Formats the destination origin or both. The disk will be formatted in the density which the drives are currently set. To format the origin press 'O'. To format the destination press 'D'. If "B" is pressed, DISKIO asks for a disk to be inserted in the origin drive. When RETURN is pressed it immediately asks for a disk to be inserted in the destination drive and begins formatting the origin. When the origin disk is finished, DISKIO asks for a new disk in the origin drive, then automatically format's the destination disk without any prompting. The origin and destination obviously cannot be in the same drive, and once again must be in the same density. The number of disks formatted is displayed after each disk format. This process continues automatically until the OPTION key is held down. This is a real time saver when formatting a box of disks.

(NOTE: This command will format single and double density 720 sector disks,and enhanced density disks. The format commands will not correctly format double sided disks, 8 inch disks, hard disks, or ramdisks.)

Q - Quick Format a new disk that has been previously formatted, but may be full of useless files in the actual formatting process, the drive essentially tells a brand new disk where and how an Atari computer expects data to be stored. Once this is done, it is not necssary to repeat the entire procedure. This command does only what is necessary to clear the disk, just like a new formatted disk, and does it in only a few seconds. See note in Format command explanation.

C - Correct File Number traces a file selected from the directory and makes the link bytes of the file number match the location of the directory entry. Remember the file number is not stored in the directory but is determined by the filenames location (i.e.the first file is number 0, the second is number 1, etc.). This command will repair the DOS 2 "File Number Mismatch" error (#164). The "C" command will not do any thing on a double sided or high capacity disk because the files on these disks do not use file numbers.

A - Access Subdirectory. Select the directory name just like a filename,and it will display exactly like the root directory. Once in a subdirectory, another may be accessed the same way. This may continue to an unlimited number of directories. To return the root directory "E"xit, then reenter the "D"irectory screen. A subdirectory appears as a filename with ATASCII "club" ($10), status byte or an ATASCII "0"($30) if locked. The length should always be 8 sectors, and the start the start sector is where the directory starts on the disk. Subdirectories are identical in structure to the root directory (the one from sector 361 to 368) they just appear somewhere other than sector 361 on the disk subdirectories are a way to put more than the maximum 64 files on a 10 megabyte hard disk. All of the directory screen commands work on subdirectory files just as they would on the files in the root directory.

+ - Next directory sector, and eight filenames.

- - Previous directory sector, and eight previous filenames.

P - Print this sector to printer.

S - Change the start sector (in the directory) of any file.

EX: If the start sector is damaged, this command will repair it, but it is useful in other ways. If for example, changes to the variable name table in a basic save file are to be made without modifying the first sector. The first sector can be copied to an empty sector and the changes made. The link will still point to the second sector of the file. Use this command to make the new sector the first sector in the file. The original first sector will remain intact. Essentially this file will have two different starting sectors.

M - Modify the number of sectors (in the directory) of any file. This command, used with the start sector command can be used to add any number of sectors to the start or end of a file

E - Exits to Main Screen.

FILE MAINTENANCE SCREEN

The majority of the commands on this screen involve examining and recovering files. If you are not very comfortable with Atari file structure, read appendices 7 and 8.

FILE MAINTENANCE COMMAND SUMMARY

T - Trace a file, attempts to determine the file's type and display it's load vectors. After telling DISKIO the start sector of the file to be traced, the option is given to just trace the file without trying to determine the file type. This is done by pressing "Y", and may be used when tracing a known text or data file. If any other key is pressed, it reads in the file type

If it is a basic save file, (see appendix 8) the page zero tokenized file parameters are displayed, with the 256 byte offset added as it would be on the disk, and the option to continue or abort the trace is given.

If a binary file, (see appendix 7), is being traced the load vectors and the sectors where they occur will be displayed. In the case of special init and run vectors, the address of the vector will also be displayed

If the first 2 bytes are not $FF $FF then "T"race assumes it is a text file and traces with no vectors.

The entire file trace process may be echoed to the printer with the option key.

B - Boot disk vectors from sector one (see appendix 1). this may be echoed to the printer with the OPTION key.

C - Check for bad sectors in any range. Any bad sector found will be displayed along with its error code. This may be echoed to the printer with the option key. When the search of the given range is complete, DISKIO gives the option to lock these sectors out (allocate) in the disk map and create a new file in the directory named "BADSECTO.RS". The length of this file will correspond to the number of bad sectors found on the disk and the number of free sectors will be decremented accordingly. This often allows the remaining area on the disk to be used, provided none of the boot, map, or directory sectors are bad.

M - Modify VTOC bit when selected, this command asks whether the sector is to be allocated (used) or deallocated (freed) in the sector map (see appendix 6). At this point a return will abort the command. The sector number to be changed is entered and DISKIO handles the rest. This command may be used when adding (or removing) a sector to (or from) a file. It may also be used to protect a sector from DOS in wwhich information such as a serial number or some piracy protection scheme could be hidden. If there is confusion on what this command does, format a blank disk and print its map (using the main screen "M" command). Then allocate or deallocate a sector and see what happens to it on the map screen.

R - Repair Directory attempts to recover the files from the disk on which the directory has been scrambled or damaged. Once this command has been selected, put the disk in the origin drive and press RETURN. DISKIO will now clear the entire directory and sector map. Since sector four is the first sector after the boot sectors, DISKIO will assume a file starts there. This file will be traced and a filename, DISKIORD.G00 - DISKIORD.G63 will be generated in the directory. It will then continue to trace every file or file fragment on the disk and generate new filenames until the directory is full or the entire disk has been checked. The sector map is then modified to reflect these new files

Each file must then be loaded, identified and renamed Any file that has been deleted will be returned even if it has been partially overwriten. These file fragments can send the repair directory command to strange places. DISKIO will assign any file with sectors that link to the boot, map, directory, nonexistent sectors, or themselves with a file length of 999 sectors, Obviously a file on a DOS 2 disk cannot have 999 sectors, so any definitely damaged files found by this command takes a while to complete its job and creates a bit of file juggling, but is definitely preferable to losing an entire disk of important files.

It is recommended that you use DOS to copy any known good files to another disk before using this command. This will save some work when identifying and renaming the files

If the directory sectors were physically destroyed, either from magnetism, wear, or maybe coffee spill, sectors 1-359 and 169-720 may be copied to another disk and the directory reconstructed there.

This command is currently limited to single and double density 720 sector disk. Subdirectories will confuse the issue, so any recoverable files in a subdirectory should be copied to another diskand then preferably fill the subdirectory sectors with $FF's to make sure it is not mistaken as a file

G - Generate file allows files, possibly missed by the "R" command, to be recovered. If the first sector of the file can be found using the sector map as a clue, DISKIO will create a file named "DISKIORD.G64". The file number will not necessarily be right but can be repaired with the directory screen "C" command.

When adding a sector to the front of a file (see appendix 8) this command could be useful. Pick any free sector on the disk as the new start sector and change its link to the new start sector. Once again, use the directory screen "C" command to correct the file number, or see the example for the Main Screen </> command.

W - Write boot, binary file menu creates a boot program on a formated single or double sided disk which will display and load up to 12 binary load files without DOS. Since there is no waiting for DOS to, the disk menu will boot in just a few seconds. The only files that will come up on the menu are the ones that are locked.

Not only does the boot menu come up more quickly than DOS, due to the way it handles the file load, it loads and starts the files more quickly than DOS "L" command.

This menu will be considered to be public domain DISKIO ITSELF IS COPYRIGHTED AND IS NOT IN THE PUBLIC DOMAIN

Not only does the boot menu come up more quickly than DOS, due to the way it handles the file load, it loads and starts the files more quickly than DOS "L" command.

This menu will be considered to be public domain DISKIO ITSELF IS COPYRIGHTED AND IS NOT IN THE PUBLIC DOMAIN.

ATARI DISK STRUCTURE
(appendix 1)

The-Atari disk drive, during the formatting process, sets a blank disk up with 40 tracks (concentric circles) of 18 sectors each, or 720 sectors numbered 1-720. If the disk is formatted in single density each of these sectors are 128 bytes long. If double density, sectors 4-720 are 256 bytes long and sectors 1-3 (the boot sectors) are 128 bytes long (single density). (The reason for this is when the computer is booted it assumes the disk boot sectors will be single density. The code in the three boot sectors must inform the computer that the rest of the disk is double density.) DOS then clears the boot and directory sectors and sets up the map or VTOC sector. Through an oversight, DOS was written to access 720 sectors, 0-719, leaving sector 720 out. Since the drive is not designed to access sector zero, there are only 719 sectors available to DOS. Subtracting the three boot sectors, eight directory sectors, and one map sector there are 707 free sectors available for file storage on a standard Atari DOS formatted disk.

The sectors on a boot disk are read sequentially so all 128 bytes are considered data. The sectors that make up a file are 'linked' together so DOS can arrange the files on the disk wherever free sectors are available. In other words, the sectors in a file may not follow numerical sequence. The last three bytes of each sector in a file contain the link data, or the number of the next sector in the file. This linking continues until a sector is read that links to sector zero, signifying the EOF (end of file). These three bytes also contain the file number which must match the number of the filename in the directory. If a file number in any sector does not match, a 'File Number Mismatch' error occurs. For example, all sectors in the file numbered zero must have a zero in their file number bits of the link bytes.

The very last byte in a file sector contains the byte count of that sector, or how many bytes in this sector are really data. This is usually 125 for single density and 253 for double density; each being three short of a full sector to make allowance for the link bytes.

FILE LINK BYTES

Byte: 125(253) 000000xx First six bits are file #, two lower

bits are the upper two bits of the ten bit link sector (this is why DOS may only access 1023 sectors.)

126(254) Link sector low byte
127(255) Byte count

ENHANCED DENSITY DISK STRUCTURE
(appendix 2)

Enhanced density disks formatted by DOS 2.5 on a 1050 disk drive have a structure similar to, but slightly extended beyond the DOS 2 disk. The disk is single density with 128 bytes per sector,but as opposed to having 40 tracks of 18 sectors each, it has 40 tracks of 28 sectors each. This gives a possible 1040 sectors per disk. Enhanced density disks do use the DOS 2 six bit number leaving only ten bits for the file link information. A ten bit file link can only access 1023 sectors, so this minus three boot, eight directory, and one map sector leaves 1011 sectors free for file storage. The directory is set up the same as a DOS 2 directory, except that files containing any sectors above 719 are given a special status byte (see appendix 5). This is what generates the brackets around these files in a DOS 2.5 directory listing, and this is what excludes these files from a DOS 2 directory listing. The map in sector 360 is identical to a DOS 2 map except the sector count reflects the larger storage space. An extended map is contained in sector 1024 (one sector above the file storage area). This map is offset by 16 bytes from the map in 360 (byte number 0 in the extended map is byte number 16 in the map in sector 360. In other words bit seven of byte zero in the extended map represents sector #48. Bit seven of byte $54 will be zero locking out sector 720. Bit zero of byte $79 represents sector 1023. Bytes $7A and $7B contain a count of free sectors above 720. This is initialized to $012F (#303), which, when added to 720 gives the available

1023 total sectors on an enhanced density disk. The 16 sectors between 1025 and 1040 are unused and could be used to hide information such as an alternate directory or copy protection scheme.

HIGH CAPACITY DISK STRUCTURE

(appendix 3)

Double sided disks, 8 inch disks, hard disks, etc., have basically the same structure as a standard DOS 2 disk. There are really only two significant differences. First the six bit file number is completely dropped, leaving 16 bits for file sector links. This gives 65535 allowable sectors on a disk. The other difference is the VTOC or sector map The maximum number of sectors which could be represented in a single, double density sector are 1967 (including the inaccessable sector zero). Disks containing more sectors than this must contain multiple map sectors. The first map sector is always 360 where the first byte (if greater than two) minus two equals the number of VTOC or map sectors (see appendix 6). The second map sector will be 359, the third 358 and so on until enough map sectors exist to account for all of the available sectors on the disk.

HIGH CAPACITY LINK BYTES

Byte: 125(253) Link sector high byte
126(254) Link sector low byte
127(255) Byte count

BOOT SECTORS

(appendix 4)

The boot sectors (1-3) are always single density (128 bytes). Sector 1 is the first sector read when the computer is booted up. The first byte should always be zero. The second tells how many sectors to load during the boot process. These sectors will be loaded consecutively starting with sector one. Since this is only one byte, the most that can be loaded in a single stage boot is 255 sectors. The third and fourth bytes of sector one are the address where the load, beginning with byte zero of sector one, will start. The fifth and sixth bytes are the address where the computer will jump after the first stage load is complete.

If more than 255 sectors are needed or the disk is double density, a multi-stage boot is used. The code loaded in the first stage is responsible for loading the rest of the data.

BOOT DISK LOAD VECTOR

Byte: 0 - is always zero
1 - number of sectors to load
2 - load address low byte
3 - load address high byte
4 - start address low byte
5 - start address high byte

DIRECTORY SECTORS
(appendix 5)

The directory sectors (361-368) are each capable of containing 8 filenames, their starting sectors, length in sectors, and status byte. This is also true of double density disks. The last 128 bytes of a double density directory sector are left alone. The status byte tells the condition of the file.

VARIOUS ATARI FILE STATUS BYTES

HEART(atascii) - empty directory entry
INVERSE HEART(atascii) - deleted file
'b' - DOS 2 locked file
'B' - DOS 2 unlocked file
'f' - MYDOS locked file (high capacity disk)
'F' - MYDOS unlocked file (high capacity disk)
'#' - DOS 2.5 locked file which exists above sector 720
(atascii 3 symbol) - DOS 2.5 unlocked file which exists above sector 720
'0' - ($30) locked subdirectory
(atascii club symbol) - unlocked subdirectory

STATUS BYTE BIT DESIGNATION

BIT:0 - SET - currently open file
1 - CLR - DOS 1 / SET - DOS 2
2 - SET - 16 bit file links (high capacity disk)
3 - CLR - always
4 - SET - subdirectory
5 - CLR - unlocked / SET - locked
6 - CLR - always
7 - SET - deleted file

When a file is deleted by DOS no data is actually erased The status byte is set to keep the filename from being seen by DOS and the disk map is changed to allow the space occupied by the file to be overwritten.

DIRECTORY BYTE STRUCTURE

Byte: 0 - Status byte
1 - File length low byte
2 - File length high byte
3 - File start sector low byte
4 - File start sector high byte
5-15 - Filename

MAP SECTOR(S) (VTOC)

(appendix 6)

The map sector (360) contains the number of free sectors on the disk in the fourth and fifth bytes. Starting in the lower four bits of the eleventh byte and continuing through the one hundredth byte is the sector map. (On a high capacity disk this may continue for any number of sectors (see appendix 3).) Each bit in these bytes represents a sector 0-719, sector zero being bit four of byte eleven and sector 719 being bit zero of byte $53. Bits set to ones represents available, or free sectors, and bits set to zeros represent occupied sectors. This map is how DOS finds a blank space on the disk for a new file.

The first byte of the map in sector 360 determines what type of disk is being used.

0 - DOS 1
1 - DOS 1
2 - DOS 2 (707 or 708 free sectors)
3 - disk has more than 1024 sectors
4 - 2 VTOC sectors
5 - 3 VTOC sectors
6 - ETC...

For more information on multiple VTOC sectors see appendix 3

BINARY FILE STRUCTURE
(appendix 7)

Most binary files load in several segments. At the beginning of each segment is a set of bytes called a load vector. The reason the files are broken into segments, instead of just loading into the starting address and continuing until the EOF is found, is so the programmer will be able to put his code anywhere in memory at any time by simply defining a new orgin in the source code. In other words, everytime an origin is encountered in the source code the current load segment is terminated and a new load vector is created representing the location of the new origin.

The first load vector in a file is six bytes long The first two bytes of the first vector (and file) must both be a $FF (255) for the DOS 'L' command to load the file. The third and fourth bytes in the vector contan the address where the first byte of the segment (the start of the program) is to load. The fifth and sixth bytes are the end address of this segment The start and end addresses are used to compute the number of bytes to load in this segment, not including the six vector bytes Starting with the seventh byte, the file is loaded into memory until the number of bytes called for in the load vector is reached. At this point if the EOF has not been reached, another load vector should be encountered All of the load vectors except the first are 4 bytes long. The first two bytes in these vectors contain a new start address and the next two bytes contain the new end address. These are also used to compute the number of bytes in the segment, or number of bytes to the next segment.

The init and run vectors are two special cases. These look like $E2, $02, $E3, $02 (init) and $E0, $02, $E1, $02 (run). They mean the next bytes (after the vector) will be loaded into locations $2E2 and $2E3 for the init and $2E0 and $2E1 for run. These locations are checked after the loading of each segment. If $2E2 and $2E3 do not contain zeros, then this is assumed to be an init address of the file. DOS then JSR's to this address before continuing the load. When a RTS is executed, DOS zeros $2E2 and $2E3 and the load continues with the next vector if the EOF has not been reached. If $2E0 and $2E1 do not contain zeros, DOS will continue loading until the EOF is reached, at which time it will JSR to that address.

BINARY LOAD VECTOR BYTES

Byte: 0 - $FF if first vector in file
1 - $FF if first vector in file
2 - Start address of segment low byte
3 - Start address high byte
4 - End address of segment low byte
5 - End address high byte

BASIC SAVE FILE STRUCTURE
(appendix 8)

The first 14 bytes of a file created by the BASIC 'SAVE' command contain various pointers which are loaded into zero page memory starting at location $80. Each of these pointers is an offset from byte number 15 in the file (the first byte after the set of pointers). During the save process, Basic moves these seven two-byte pointers from their zero page locations to a temporary area to be saved, subtracting the value of the first pointer from itself and each of the other six. The first pointer, $80 and $81, contains the length of a 256 byte, multi-purpose buffer. This buffer is not saved and its pointer always contains a 256. During the execution of the Save command, the first pointer is substracted from itself, therefore, the first two bytes of the file will always be zeros. Two zeros in the first two bytes of a file is how BASIC recognizes a saved file. This also explains why the other six pointers in the displacement list are actually their real value +256.

These pointers, starting at the second one, represent: the start of the variable Name Table, the end of the Variable Name Table, the start of the Variable Value Table, the start of the Statement Table, the start of the Current Statement Table, and the Displacement to the end of the file.

The start of the Variable Name Table pointer ($82, $83) usually contains 256, since the VNT normally starts in the 15th byte (displacement 256-256=0). This table contains all the variable names in the program with each name's last character displayed in inverse.

The end of the VNT ($84, $85) points to the last byte in the VNT from byte #15.

The start of the Variable Value Table is pointed to by ($86, $87). This table contains the value of each variable at the time the program was saved

The start of the Statement Table is pointed to by ($88, $89). This table contains the line numbers, statement tokens, offsets into the VNT to represent variables, and other data used to make up the program itself.

The start of the Current Statement Table is pointed to by ($8A, $8B). This is not used by the save/load process. These are the bytes used to create a "run only" file, done by adding this line to the end of your program - "32000 POKE PEEK (138) + PEEK (139) * 256 + 2,0: SAVE "D:FILENAME>EXT": NEW", then in immediate mode type G.32000

Displacement to the end of the file is at ($8C, $8D). This number minus 256 is the length of the program.

This information can be used to do several things. Variable names in the VNT on the disk can be changed as long as you inverse the last character in each name. Once the variables are changed in the VNT they are changed automatically throughout the program. If the VNT is filled with return characters, the program will still run and list, but the listing will be very difficult to read. Another method is to fill the VNT with a single inverse character, for example, and inverse A. This will also make the listing difficult to decipher.

If you are trying to list a file modified in this way you can fill the VNT with different variable names (inverse A,B,C...). If the VNT is not long enough, or you do not want to use single character variable names, it is possible to lengthen the VNT on the disk. This is done by adding a sector to the front of the file by changing the start and length in the directory. (NOTE: The FMS 'G' command could also be used to generate a new file with the new starting sector.) Set the link bytes of the new first sector accordingly, then copy the original first sector to the new first sector. In link mode rotate the bytes in the original first sector to the left until the first byte of the VNT is in byte zero (15 times), then subtract that number (15) from the byte count. Now the new first sector, from byte 15 to the link bytes can be used for variable names. The only thing left to be done is to change the zero page pointers in the file to reflect the extra length (all but the current statement Pointer which must be lleft alone). The pointers must be exact or the computer will lock up.

MAC/65 SAVE FILE STRUCTURE
(appendix 9)

Like BASIC, the MAC/65 also tokenizes source files when the "SAVE" command is used Tokenizing is a method of saving files in a format that is easier for the computer to reload, making these files load faster and occupy less space on the disk. The first four bytes of a MAC/65 saved file make up a header similar to a binary file. The first two should always be $FE, This indentifies the file. The first two should always be $FE, this indentifies the file as MAC/65 source file. If they are not $FE then the MAC/65 will give a "file type error #23" when loading the file. The next two bytes are the length, in bytes, of the file excluding the header. So this length is given by:

# of sectors -1 * 125 [or 253] bytes/ sector + byte count of last sector - 4 bytes for the header

The number of sectors - 1 is the length of the file minus the last sector (since the last sector is unlikely to be full). Remember the length bytes will be in the low byte/high byte order on the disk. The length bytes will cause a file to crash if they are not exactly correct. If these length bytes are greater than they should be the file will appear to load but will not list. It will be as though the file was not loaded at all. If the file appears to load ok, but when listed the last line that comes up has some trash in it (this may or may not be the actual last line of your file) try deleting the line. If the length bytes, are less than the actual number of bytes in the file the computer will probably lock up. Re-boot, calculate, and fix the length bytes. The calculations are not as complex as they look, try them on a known good file for practice.

The first byte following the four byte header is the low byte of the two byte line number of the first line in the file.The line numbers in the source, for example line 10 would be $0A $00 in the file. The next byte in each tokenized line is the length of that line from the first byte of the line number to the end of this line (the byte before the next line number). The rest of this line (up to the next line number) is the actual source code in tokenized form. Labels, equate names, strings, and comments show up as recognizable ATAsCII in the file. These are good clues to look for when trying to reconstruct a damaged file. Here are two ways to salvage a damaged MAC/65 source file. First, if possilbe, always copy your damaged file to a work disk before attempting to repair it. If only a few bytes in a source file are damaged the file will load, but when listed it will only list to the point in the file where the damage is. During its detokenizing process (when listing) the MAC/65 attempts to interpret the bytes it gets the only way it knows how. If there are bytes there that don't belong, the listing will go from seeensible source to a line number around 20 zillion and will probably end in an error #22- overflow. Note the last complete line number, convert it into hex and search the file for it (remember they are in low byte/high byte order). Looking at this lines length byte, find the last byte in that line. The next byte should be the low byte of the line number of the damaged line. Look for the next two byte pair that could be a line number, keeping in mind that if the line numbers were incremented by 10, then the number of the next hopefully good line will be some multiple of 10 and greater than the line number before the trashed line. Now comes the magical part. Count the number of bytes between the end of the last good line and the line number bytes of the first good line after the trash. If this isn't very many ('many' is relative to just how bad you don't want to have to retype this file) then this trashed area can be replaced with a comment line. A tokenized comment line takes the following form.

EX: 0010 ; (note one space between the line number and ;)
and in hex tokenized form:

```
0A 00 05 58 3B
  |    |  |  └── ATASCII
  |    |  └───── TOKEN
  |    └──────── LINE LENGTH
  └───────────── LINE NUMBER
```

These bytes could be put into the trashed area of the file (with the line number bytes changed to wherever the line needs to go) and this would replace five bytes of the garbage with a comment line. The trick is to fill the rest of the trashed area by simply putting more $3Bs at the end of the line then change the line length to reflect this. For example, if there were nine bytes of garbage between line 10 and line 30 (obviously line 20) then these bytes would be inserted:

```
14 00 09 58 3B 3B 3B 3B 3B
  |    └──────────────── NEW LENGTH
  └───────────────────── LINE NUMBER
```

The file could then be loaded and Ine 20 would list as

```
0020 :::::
```

which could then be changed to whatever it was before the crash.

Do remember the ATARI's 120 character line length limit when using this technique. It would be best to break a large damaed area into several lines several lines 100 bytes long then catch the odd number left with a shorter line.

When the damage occupies several sectors it is best to recover the file using another method. Armed with a few basic concepts and DISKIO one will be able to recover at least part of a file whether the damaged sectors are at the beginning, in the middle, or near the end of the file. Since there is no reference from one line to another in the tokenized file several lines (bytes in sectors) may be removed from anywhere in the file directly on the disk. The only thing this requires is changing the link bytes in the file sectors to redirect DOS around the bad sectors. The file length bytes in the header will have to be corrected as before to reflect the lost lines. Obviously there must be no line fragments in the file or the de-tokenizer will get confused. It is unlikely that the last good line before the bad sectors will end exactly in the last byte of its sector. In other words the line may start near the end of the last good sector and carry over to the first bytes of the first bad sector. If this bad sector is removed then so is the last of the line. This is not as big a problem as it seems because DOS does pay attention to the byte count of a sector and loads into memory only as many as it is told.

In the case of a file with sector damage at the end, search the last good sector for the last line (remember the 3 link bytes), look for the line number and use the line length byte to determine whether or not the line carries into the next sector (it proably will). Count the number of bytes this line occupies (again, don't count the link bytes) and subtract this number from the sector byte count (see the "L" command). This will become the last sector of the file. So change the link byte to the zero (See the "L" command). When DOS loads it only the bytes up to the line fragment will be recognized Now the file byte count must be adjusted as discussed before

If the damage is in the first sectors of the file, the header will have to be recreated. In this case there will probably be a line fragment at the beginning of the first good fragment. Here's the tricky part, the line data in the sector must be moved towards the first of the sector ("</>" command). Be sure DISKIO is in link mode ("CTRL-L" command) when rotating these bytes. The idea is to move the line number of the first complete line, after the fragment, to the first of the sector then adjust the byte count so DOS will not see the fragment bytes which are now at the end of the sector. Replaceing the header can be done two ways. If there is room the first line could be moved to start in the fifth byte of the sector and the header could then be put in the first four bytes of the sector. The other way would be to add a sector to the beginning of the file, and use the "G" command to create a new directory entry recognizing this new first sector (as explained in appendix #8). Change the link bytes of this new sector to point to the, now second sector in the file. The header would be put in this sector and the sector byte count would be four.

In the case of damaged sectors in the middle of the file, parts of both of these techniques would be shortened (byte count) and it would be made to link to the first good sector after the
amage ("L" command). The bytes in this sector would then be rotated ("</>" command) and its byte count charged, to get rid of any fregment. Remember to correct the file length bytes.

Always back up a file after recovering it before doing anything else. It would be a shame to work to repair a file only to have it damaged futher by some software critter before getting to even look at the fruites of your labor.

DISKIO USERS NOTES

Any time DISKIO asks for a number, it can be entered in one of three ways: as a decimal number with no '#' sign, as a decimal number with a '#' sign, or as a hexadecimal number preceded by a '$' sign. The latter is helpful if you are already familiar with, and prefer using, hexadecimal notation or just wish to become more familiar with it.

Hold the RETURN key down when starting DISKIO to skip the title screen.

If using the Option Command to set the drive parameters to single density, single sided (a normal disk), just hold the RETURN key down all the way through the menu. This will select the default parameters.

Anywhere a character in a group of selections is in inverse that will be the default taken if RETURN is pressed.

DISKIO
COMMAND SUMMARY

MAIN SCREEN

E - Exit to DOS
? - Help Screen
R - Read sector (entered in Hex or Dec.)
W - Write sector
P - Print Screen
T - Hex / char. / internal char. values (if OPTION is held) toggle
D - Directory Screen
S - Search for up to 8 bytes if in hex mode or a string of up to 8 characters if in character mode
C - Change bytes in sector.
L - Link modify allows changes to the link sector, file number and byte count.
CTRL-L - Seq/Link toggle determines whether the "+" command moves to the next linked file sector.
F - File maintenance screen.
M - Disk Map of sectors 1-719 (OPTION - print)
O - Drive options selects the origin and dest. drive config
Z - Fill one sector, or range of sectors, with any number from 0-255
K - Copy sectors from origin to destination.
X - Print file sectors origin to destination.
+/- - Next/Previous sector.
</> - Rotate bytes within a sector Excludes link bytes.
$/#/% - Two byte number base conversion. First press the symbol of the base to be converted. Then enter the number to be converted, with it's base.

DIRECTORY SCREEN

D - Delete a file from the disk
U - Undelete a deleted file.
R - Rename a file.
L - Lock a file.
X - Unlock a file.
F - Format origin, destination, or both.
Q - Quick format.
+ - Next directory sector.
- - Prevous directory sector.
P - Print this sector.
S - Change the start sector of a file.
C - Check file number in the file links.
A - Access subdirectory.
E - Exit to main screen.

FILE MAINTENANCE SCREEN

T - Trace file and identify vectors (OPTION - print)
B - Boot disk vectors (OPTION - print)
C - Check for bad sectors and optionally lock them out.
M - Modify VTOC sectors bit uses or free a sector in the map.
R - Repair directory by first clearing the directory and map sectors, and then reconstructing each "possible" file according to the map.
G - Generate a new file in the directory and a map by knowing it's starting sector.
W - Write boot, binary file menu which loads locked binary files without DOS. Loads about 20% faster than DOS.
E - Exits to main Screen.

ATTENTION

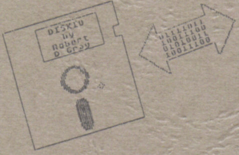
PROGRAMMERS

If you have written a program for the ATARI ST or ATARI 8-bit computers and would like to have it published we at Omega Soft would like to take a look at it. We are always looking for good quality software to add to our line of ATARI products. If you would like more information on submitting programs for review please give us a call and we will help you in any way we can.

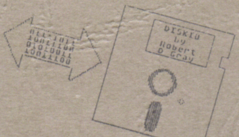
Omega Soft
Attn. Program Dept.
Po Box 139
Harrells, NC 28444

(919) 532-2359

INDEX



DISKIO



This program allows you to do practically anything to nearly any kind of disk. It is compatable with single, double, and enhanced density. Double or single sided, 8 inch or hard drives as well. also supports ramdisk.

DISKIO is for the experienced programer as well as the inexperienced. Change link sectors, file numbers, byte counts, etc. in just seconds.

Reconstruct an erased, scrambled or destroyed directory recovering most, if not all of the lost files. Even trace through files to find out where the problem is.

DISKIO allows you to check a range of sectors or even an entire disk for bad sectors and then gives the option to lock out all bad sectors.

All standard DOS commands as well as a few new ones are available on the directory screen.

If you ever delete the wrong file from your disk now you can undelete it.

Tired of the old "File number mismatch error 164" DISKIO will correct this error and restore your data.

OM1272

Omega Soft™

Po Box 139
Harrells, NC 28444

(919) 532-2359