Wordmark Systems

MyDOS 4.50

Technical User Guide

For Atari Home Computers

By
Charles Marslett &
Robert Puff

MYDOS Version 4 Technical User Guide
Revision 4.50
for Atari Home Computers

Charles Marslett
2705 Pinewood Dr.
Garland, TX 75042
CIS: 73317,3662
UseNet: CHASM@KILLER.DALLAS.TX.US

and

Robert Puff
Suite 222
2117 Buffalo Rd.
Rochester, NY 14624
GEnie: BOB.PUFF

IV. THE MENU COMMANDS: The MEM.SAV feature

Whenever the DOS menu is entered, MYDOS will load DUP.SYS from drive 1. If DUP.SYS cannot be found on drive 1, it will look for it on drive 2, 3, etc... until it finds it.

Loading DUP.SYS will overwrite a good chunk of lower memory, and will wipe out the data in most languages. Here's where MEM.SAV comes into play. When this feature is enabled (by going to the menu and pressing "N" and [RETURN], then "B" to exit back), the memory that DUP.SYS would overwrite is saved to the file "MEM.SAV" on drive 1 (may be drive 8 if you use one of the RAMBOOT programs on the disk). It then loads the DUP.SYS. When you press "B" to return, or if you use the "N" command of the menu, MYDOS will load back in the MEM.SAV, thus restoring the overwritten memory. The result is your program remains intact.

V. FILE MANAGER FUNCTIONS PROVIDED THROUGH CIO

MYDOS supports all CIO calls supported by ATARI DOS 2, with the following modifications to the OPEN (Function code 3) and the FORMAT (Function code 254) functions. Three additional CIO functions have been added: MAKE DIRECTORY (Function code 34 & 42), SET DIRECTORY (Function code 41) and LOAD MEMORY (Function code 39 & 40).

The OPEN function in ATARI DOS 2 does not use the data provided in the AUX2 byte, but in MYDOS when the AUX1 byte is 8 (the file is opened for creation or replacement), the AUX2 byte contains two flags that control the file format, and whether it will be created locked or not.

If AUX2 bit 2 is set, the file will be written in MYDOS 4 format, and may contain sectors beyond absolute sector 1023. Such a file may not (easily) be read by DOSes other than MYDOS. This is the only format used with high capacity disks. If AUX2 bit 5 is set, the file will be written with the "LOCKED" bit in the directory set initially. This is provided for use by multi-tasking functions (such as a print spooler, sequential file pre-reading function or other enhancements one might want to make to the standard OS or DOS provided functions).

The FORMAT function in ATARI DOS 2 does not provide for any variations to the standard disk usage: in MYDOS the contents of the AUX1 and AUX2 bytes are used to specify the number of sectors on the disk being formatted, and whether the disk needs to be formatted by the controller as well as needing directory initialization. Bit 7 of AUX1 is set to skip the physical formatting of the entire disk surface when it is not required, and bits 6-0 of AUX1 and all of AUX2 are used to specify the number of sectors on the disk being created (if all 15 bits are zero, the disk is assumed to be the size defined by the drive configuration). This permits formatting a single sided diskette on double sided drives, for example. Be careful when using

this feature.

To load (and possibly execute) a program file, MYDOS provides the CIO function 39 call. CIO function 40 will do the same; it was included for compatibility with programs written for SpartaDOS. From BASIC you can load and execute a program by executing the line: XIO 39, #3, 4, 0, "D:MYPROG.OBJ". Any inactive IOCB can be used, and if AUX1=4 both the INIT and the RUN entries will be executed. If AUX1=5, the RUN entry will be executed; if AUX1=6, the INIT entry will be executed; and if AUX1=7, the file will be loaded without executing either entry point. Any other values of AUX1 will return an error code and do nothing.

Another XIO call, XIO 34, has been added to create a directory. CIO function 42 performs exactly the same thing, and has been provided for compatibility with programs written for SpartaDOS. When a directory is created, the name used must not match any existing file or directory in its parent (for example if the directory to be created is named "D1:TEST>BUGS", there can be no other directory in the main directory named "TEST" nor a file named "TEST" there.

From BASIC, the XIO 34 call is "XIO 34, #iocb, 8, 0, dirname", where "iocb" is any available unit number, and "dirname" is the name of the new directory (it does not end with a trailing ":" or ">").

The final function added to those provided by ATARI DOS 2 is XIO 41, to define the default directory. The default directory is that which will be searched for a file if the file name begins with "D:". In ATARI DOS 2 this default directory is always "D1:" but in MYDOS, the default directory can be any root or subdirectory on any disk in the system. The buffer address passed to CIO in the XIO 41 call is the address of a string that contains the default directory name, terminated with either an end of line ($9B) or a null byte ($00). The directory will be accessed before returning to the calling program so that an error in specifying the directory will be reported as early as possible.

VI. CIO FUNCTION CODES PROVIDED BY MYDOS 4.50

Function code 3, OPEN

The open function uses the buffer address to point to an ATASCII string terminated with a non-alphanumeric character or wildcard. This string is the name of the file to be accessed or created. A good terminator for this string is either a null ($00) or an end of line ($9B).

The AUX1 byte defines the usage of the file: 4 for input, 6 for directory data reading, 8 for creating/replacing output, 9 for creating/appending output and 12 for input/update (without extension). The AUX2 byte is used when a file is replaced or created, and contains two significant bits: bit 2 set causes the MYDOS format to be used even if the diskette is a 40 track single sided diskette;

and bit 6 set results in the file being LOCKed initially without and additional CIO call. For input, update or directory access AUX2 is ignored, and the length is always ignored. In normal use, AUX2 is set to zero emulating ATARI DOS 2 usage.

MYDOS does not leave partially full sectors when appending to a file. This has two positive effects on programs which open files in append mode: the open will fail if the file cannot be appended to rather than the close (in ATARI DOS), and the file size will not change if a file appended to is copied to another disk (in ATARI DOS it may grow smaller).

Function code 5, GET RECORD

The get record function reads a line of data into a buffer, the buffer being defined by its starting address and length. The line is defined as the data bytes in the file up to an end of line character ($9B) or until the buffer is full, whichever occurs first. The line is also terminated if the end of the file is read. All record I/O is buffered in MYDOS, so record transfers are necessarily slower than unbuffered I/O.

No other fields of the IOCB are referenced or needed. Note that the ATARI ROM OS supports single byte I/O through the accumulator if the buffer length is set to 0. In this case, GET RECORD and GET CHARACTERS function exactly the same way.

Function code 7, GET CHARACTERS

The get characters function reads a fixed number of bytes from a file into a buffer, the buffer being defined by its address and length (two 16-bit numbers in the IOCB). The only cases where the buffer is not always filled is if the end of the file is read, or the file cannot be read withut error. As is the case with get record calls, a single byte may be read into the accumulator by setting the length field to zero. A get character CIO call will perform unbuffered I/O if the buffer is longer than 256 bytes (ATARI DOS 2 sets a similar threshold at 128 bytes). For this reason a single long input is considerably faster than several short ones.

Only the buffer address and length in the IOCB are used by the get characters function.

Function code 9, PUT RECORD

The put record command will write a single line to an output file: the line defined by the starting address of the buffer and either the length of the buffer if no end of line ($9B) bytes are encountered, or the first end-of-line byte. Only the buffer address and length in the IOCB are used in this command.

Function code 11, PUT CHARACTERS

The put characters command will write the contents of a buffer defined by its address and length (in the IOCB) to a file opened for output. Unless an error occurs, the entire buffer is always written to the file unless the write is to an output/update file and the end of the file is reached or the write is to an output/append or create file and the disk has filled. Only the buffer address and length fields in the IOCB are used when the put character function is used. The single byte put character (using the A register as data) is supported by setting the length bytes to 0.

Function code 12, CLOSE A FILE

To terminate use of a file (and for an output file, to write the incomplete buffer to the disk) the IOCB used to access the file should be closed. This is done by setting the function code in the IOCB to 12 and calling CIO. The close function does not use any of the data in the IOCB for any purpose whatsoever.

Function code 13, READ STATUS

The read status command is issued to an unopened IOCB, with the buffer address that of a file name string. If the file is not present, that error condition is returned, if it is locked, that error condition is returned; otherwise, a normal completion code is returned. Only the function code and the buffer address in the IOCB are needed.

Function code 32, RENAME A FILE

The rename function is passed a character string (pointed to by the buffer address in the IOCB), the first part of the string being a file name identifying the file or files to be renamed. Following a single invalid character (one invalid in the file name, that is) a simple file name must also be present: this second file name cannot include any drive or directory names. An example, using a comma as the invalid character, is "D2:TEST>PGMS>A.OUT,ZCPY" which will change the string needed to access the file "D2:TEST>PGMS>A.OUT" to "D2:TEST>PGMS>ZCPY" -- Note that only the last file name (if subdirectories are used) can be changed; to change "PGMS" to "MLPROGS", the buffer must contain "D2:TEST>PGMS,MLPROGS".

Wild card characters should appear only in the part of the file name following the last ":" or ">", and their effect is best described by an example. The string "D2:TEST:*.*,*.XYZ" will rename all the files in the TEST directory, making each extension ".XYZ". If the directory had the files "ATEST.BAS", "LOG", and "REPORT.XYZ" in it, the result would be a directory with "ATEST.XYZ", "LOG.XYZ" and "REPORT.XYZ" in it.

Function code 33, DELETE A FILE

The delete function removes any files that match the file name string pointed to by the buffer address in the IOCB. Files locked will not be deleted, so must be unlocked before being removed, and subdirectories that are not empty (that have a file in them) cannot be deleted. If either case is attempted, the corresponding error code is returned. Otherwise, the files are removed and their data areas are returned to the free space on the disk. Files that have been deleted may be "undeleted" by various utility programs ONLY if the data has not been overwritten. If you write to the disk after you have just deleted a file on that disk, chances are that you will not be able to recover the file.

Function code 34, MAKE DIRECTORY

The make directory function will create a new subdirectory on a disk (it is not used to create the first directory, that is the "root directory" identified by the drive specification "D1:", for example). It is called through CIO by storing the address of the new directory's name in the IOCB buffer address and setting up AUX1 and AUX2 as for an open call (see Function code 3), normally AUX1=8 and AUX2=0. This function has no effect on the current default directory; and if it is desired to make the newly created directory the default one, the program must make a set directory call (Function code 41) following the make directory call (the order is very important, because the default directory cannot be set to a nonexistent directory). CIO function code 42 may also be used to access this function; the parameters are the same.

Function code 35, LOCK FILE

A file can be "locked" so that it may not be modified or deleted inadvertently, by calling CIO with the lock function. The buffer address is used to point to a file name string that identifies the files on the disk to be locked. The only file modification that can be performed on a locked file is to unlock it. The lock function can be requested for a file already locked, and it will return no error (unlike other file modification calls to CIO), but the status of the file will not have been changed.

Function code 36, UNLOCK FILE

The unlock function is identical to the lock function except that is re-enables the modification or deletion of an unlocked file. A file that is not locked can be unlocked with no error returned and no change in the file's status.

Function code 37, POINT TO POSITION IN FILE

The point function is passed the 3-byte disk address to be positioned to in the twelfth through fourteenth bytes of the IOCB. On return, the next byte read from that IOCB will be the one that was read or written next after the corresponding note function was executed. A point call to CIO can only be made if the file can be used for input: that is, if it is opened for input or update processing. The first two bytes of the disk address are a sector number (in low byte / high byte format) and the third is the byte (offset) within the sector.

If a file is being appended to (opened with AUX1=9), a point function call made before closing the file may return an unexpected error (this cannot happen with the note function, however).

A problem can occur if the file being pointed to is in the last half of a 16 Megabyte disk: Atari BASICs do not allow sector number to be greater than 32767. A solution is to use the following BASIC substitute for the POINT statement (with attention paid to the fact the the two AUX bytes must match the two used to open the file):

```
OPEN #K,AUX1,AUX2,"D5:BIGFILE"
. . .
NOTE #K,SECTOR,BYTE
. . .
POKE 844+16*K,ASC(CHR$(SECTOR)):POKE 845+16*K,INT(SECTOR/256)
POKE 846+16*K,BYTE: XIO 37,#K,AUX1,AUX2,"D:"
```

Function code 38, NOTE POSITION IN FILE

The note function returns in the twelfth through fourteenth bytes of the IOCB a 3-byte disk address that may be used at a later time to reposition the file using the point function. The note function can be used on files open for input, output, update or appending. The three bytes returned are the low byte of the sector address, the high byte of the sector address, and the byte (offset) within the sector, in that order.

Function code 39, LOAD MEMORY

The load memory function takes a file formatted in the ATARI DOS 2 executable program format (generated by the "K" command, by the assembler/editor cartridge, by AMAC or MAC65, or by any of several compilers for the ATARI computers), and loads its contents into memory as specified in the file. No offset control is provided and no part of memory is protected from the loading process. The initialization and execution addresses (if any) can be individually enabled and disabled, to permit loading and patching a program then writing it back to the disk for normal use.

To load a program into memory, the address of the file name string is stored into the buffer address, and a value of 4, 5, 6 or 7

is stored into the AUX1 field. If AUX1 is 4, both the initialization routines and the run address are executed after closing the IOCB, but before returning to the calling program. If AUX1 is 5, the initialization routines are disabled, but the program will be run. If AUX1 is 6, the initialization routines will be run, but the program execute address will be loaded and ignored. If AUX1 is 7, the text of the program will be loaded into memory, but no other activity will be performed. CIO function code 40 performs the exact same function as this (39).

Function code 41, SET DEFAULT DIRECTORY

The set directory command will use the contents of the buffer as a file name and open the specified file, determining if that file is a valid directory. If so, it will become the new default directory. File names of the form "D:..." will be assumed to be in the default directory (which may be on any disk in the system and may be either the root directory of that disk or a subdirectory).

Only the buffer address and the function code are significant when setting the default directory.

Function code 254, FORMAT A DISKETTE

The format function uses the contents of the buffer pointed to by the buffer address to identify the drive containing the diskette to be formatted. If both AUX1 and AUX2 are zero, the disk is formatted according to the capacity data in the system control table defined using the "O" command. If AUX2 bit 7 is set to 1, the format operation is skipped and an empty file system is written to the diskette. (This assumes the disk is preformatted.) The remaining 15 bits of AUX1 and AUX2 are used as a 15 bit number to specify the number of sectors available on the disk (permitting the use of the last few sectors of a disk outside the file system if desired). You may format a disk in enhanced density (MYDOS compatible - not DOS 2.5 type format) by setting AUX1 to 1, and AUX2 to 0.

VII. DISK STRUCTURES SUPPORTING MYDOS 4.50

MYDOS uses the first three sectors of a disk to hold some disk information and the initial boot program if the drive contains DOS.SYS and DUP.SYS. Sector $168 (and sectors $167, $166, $165, etc., if the disk is formatted as a higher capacity disk not compatible with ATARI DOS 2) is used to hold a bit map of available sectors and several flag byte identifying the default format of files on the disk. Sectors $169 through $170 contain main disk directory data, identifying the files on the disk, their sizes and their starting sector number.

Note that this usage, when the diskette is a 719 sector volume declared to be DOS 2 compatible, is in fact exactly the same as ATARI

DOS 2 would make of the disk. The default single sided format differs only in that sector 720 is not left out of the file system in MYDOS but is used to provide 708 free sectors on an empty diskette rather than 707. The only significant change made when the high capacity format is chosen are that enough sectors before sector $168 are allocated to assign a bit for each sector that may be allocated for a file or for use by the system (VTOC sectors). The high capacity disk directory may be read by ATARI DOS 2, but the data in the files can only be accessed if it falls in the first 1023 sectors of the disk and then only if the file number checking code in DOS 2 is disabled. This format allows MYDOS to support accessing disks of up to 65,535 sectors of 256 bytes each (approximately 16 Mbytes).

Compatibility with DOS 2.0 is further reduced if subdirectories are used: to ATARI DOS, the subdirectories will appear to be simple files with unreadable contents. The subdirectory's files will not be accessible and the subdirectory can be damaged if it is written to (even by appending). For this reason disks sold to the general public, exchanged with friends, and so forth, should not contain subdirectories unless there is reason to require that the disk be used with MYDOS. A further problem with exchanging diskettes is that there are many different formats are used by vendors of double sided disk systems for the ATARI. For this reason, double sided disks not only require both computers use MYDOS, but also require that they use the same disk system (PERCOM, SWP, Astra, Supra or whatever).

VIII. MYDOS 4 MEMORY MAP

The MYDOS 4.50 disk operating system occupies the area from $0700 to $1EE9 at all times, and when the menu is active, it also occupies the area from $294A to $4331. In addition, the first 16 bytes of the floating point workspace ($D4 - $E3) are used by MYDOS at that time. Unlike ATARI DOS 2, MYDOS utility program (DUP.SYS) also calls the floating point ROM entry points. The nonresident part of MYDOS 4.50 starts loading at $294A, reserving the area from $1EE9 to $2949 for disk buffers and drivers. Allocating three disk buffers leaves approximately 2500 bytes for resident drivers that will not be overwritten by the nonresident portion of DOS (contained in DUP.SYS).

IX. CUSTOMIZING A SYSTEM DISK

RAMdisk Configurations

The RAMdisk driver included in MYDOS 4.50 is already configured for the Atari 130XE computer and uses its banked 64K bank of memory for the RAMdisk providing 499 free (single density, 128 byte) sectors. The "O" command provides an easy way to alter the operation of the RAMdisk driver for other common banked memory systems. Most memory upgrades for the 800XL and 130XE use the same mapping address (the PORTB pins of the PIA chip in the computer). A 128K RAMdisk can be used in an Atari 130XE using the last unused pin of that port with no tradeoff (selecting the 64K bank is done with bit 6 of PORTB). If

you have such a system, enter a "2" for the page sequence, or just use the default sequence (answer "Y" to the question "Use default page sequence...").

If, instead of adding one or two rows of 64K memory chips, the enhancement replaces the entire memory of the computer with a single bank of 256K memory chips, then the banked memory is a total of 192K and 4 bits of the port must be used to select the memory bank. Often the bits used are bits 0 and 1 (as in the 130XE) along with bit 6 (as in the expansion above) and bit 5 (used in the 130XE to control banking screen memory). Programs that bank screen memory (a very odd proposition because of the difficulty of obtaining a useful sharing of the banked memory page bits between the screen memory and the program) will not work with this enhancement. This is the approach used in the Newell Industries 256K upgrade for the Atari 800XL.

If the enhancement is done externally or to an Atari 800 (with its internal expansion slots), a new dedicated register may be used to map the 16K pages. The Axlon RAMPOWER 128 card for the Atari 800 works this way. In such a system, the pages are selected by writing a page number to the mapping address and no sharing of the 8 bit byte is necessary. The address of the mapping register is entered explicitly and page sequence "5" is the proper sequence.

The page sequence table coded into MYDOS is actually one 32 byte sequence table of numbers to be stored in the mapping register. MYDOS 4.50 has a feature that allows you to skip all this configuration stuff, if the upgrade is XE compatible. In the configuration, MYDOS will create a custom page sequence for your memory upgrade, and update its pointers. If you choose to enter our own page sequence, the number of 16K pages determines the number of bytes to be used. You may use one of the four built-in page sequences by entering a single sequence number:

Seq. No.	Page Values	OR Value
0	E3, E7, EB, EF, C3, C7, CB, CF, 83, 87, 8B, 8F, A3, A7, AB, AF	00
1	C3, C7, CB, CF, 83, 87, 8B, 8F, E3, E7, EB, EF, A3, A7, AB, AF	00
2	A3, A7, AB, AF, C3, C7, CB, CF, E3, E7, EB, EF, 83, 87, 8B, 8F	00
5	00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F	FF

As an example, if you want to use BASIC/XE in extended mode (or use a program that uses the standard XE banks: pages E3, E7, EB, and EF), and you have a RAMBO XL upgrade, set your memory size to 128K, and use the page sequence of:

```
1
or
C3,C7,CB,CF,83,87,8B,8F,00
```

This will configure the RAMdisk to use only that part of the banked memory not used by BASIC/XE.

The file RAMBOOT.M65, the MAC/65 assembler source code for which is in the file RAMBOOT.AUT, is an AUTORUN.SYS file that simulates the operation of Atari DOS 2.5 and its RAMdisk. It "formats" the RAMdisk and copies DUP.SYS to it, as well as setting the RAMdisk unit number and the unit used to access the DUP.SYS and MEM.SAV files to 8.

By modifying the code in the source file and creating a modified AUTORUN.SYS file, the drive used to save MEM.SAV and fetch DUP.SYS can be modified, other files than just DUP.SYS can be copied to the RAMdisk when the system is booted, or any other operation could be performed that you find useful.

Number of Files Open at Once

The number of files that may be simultaneously open is set with the same byte as in ATARI DOS 2: location $0709 (decimal 1801). This byte contains a number from 1 to 16 setting the number of disk files that may be open at the same time. Normally it is set to 3, the smallest number that supports all the functions in the MYDOS menu. Specifically, a copy from one disk file to another requires three open disk files. The value in the distributed version of MYDOS 4.50 is three. To permit more or fewer files open, use the "O" command followed by a RETURN. To permanently change the maximum number of files, use the "H" command to write a modified MYDOS system to a disk. Each file that may be open at one time requires the allocation of a 256 byte buffer so setting this value to 7 (instead of 3) will cause MYDOS to be 1024 bytes longer than before and the programs loaded must begin no lower than $22E9 (instead of $1EE9). In corresponding fashion, by setting the value to 1, a BBS program can be loaded in with 512 bytes of additional memory if only one disk file is ever open at one time (commonly true of bulletin board programs).

Controlling the Disk Drives Accessed by MYDOS

Like ATARI DOS 2, MYDOS automatically identifies the disk drives that are present when booted up initially and any time it is reinitialized (some programs do this on exiting to the DOS, and it is always done if the RESET key is pressed). MYDOS 4.50 is distributed with drives 1 and 2 configured, all others are omitted in order to speed up the booting process. To modify the maximum configuration MYDOS will use, invoke the "O" command for each drive to be added to (or removed from) the system. Pressing the RESET key will then use this value to redefine the system to configure the drive(s). To permanently change the maximum drive configuration, use the "H" command, writing a new copy of MYDOS back to the system disk.

Selecting or Disabling Write-with-Verify

MYDOS 4.50 is distributed with the write-with-verify disabled. Most drives are very reliable, and will never give you a problem. If

however, something happens (such as dust, a scratch in the oxide coating, or some other problem that may have arisen since the diskette was formatted), the error might not be detected. In short, if you are working with something that you want to be absolutely positive that is perfectly saved, enable the verify. This will cause the drive to read back each sector after it has been written, taking about three times the time normally taken on a write without verify.

The byte at location $0779 (1913 decimal) controls all write operations to the disk. If the value "poked" into it is $57 (87 decimal), then all writes will be read back to verify the action was successful. If the value "poked" into $0779 is $50 (80 decimal) then writes will be assumed successful, and will be performed in about one third the usual time. Note that this address is not the same as in MYDOS 4.0 and 4.1 (where it was $0770 or 1904 decimal). This byte is defined, along with the count of the number of buffers to be allocated when the file manager is initialized, whenever the "O" command is invoked with no drive specified (only a RETURN is entered in response to the drive number query). To permanently alter it, rewrite MYDOS back to the disk using the "H" command after changing it.

X. DISK DRIVE INTERFACE (via SIO)

The physical disk drives and diskettes are external to the ATARI home computers and the ones supported by MYDOS 4 are normally attached to the "serial interface connector" on the right side or back of the computer. High capacity or "hard" disks may also be connected to the parallel port of 800XL and 130XE computers. The software in the operating system (OS ROMs) to access the devices attached to either connector is call the "serial I/O driver" or SIO for short. The MYDOS disk operating system uses this lower level driver to pass all commands and information to and from the physical disk drive. Several commands were defined by ATARI to communicate with the 810 disk drive and most vendors of high performance disk systems for the Atari have adopted a slightly extended version of this set of commands. MYDOS will operate with any disk system that supports the original 810 set, but the full set of commands is required to support all the functions. An additional function necessary to perform automatic density selection is that the drive should automatically identify the density of a diskette inserted in it if the first operation is a read of sector 1 (this is necessary if the drive is to boot either a double or single density diskette).

The minimum set of disk drive functions to support MYDOS (or ATARI DOS 2 for that matter) are:

Device	Unit	Command	Direction	Byte Ct.	Aux Bytes	Function
$31	Drive#	$21	From Drive	128/256	0	FORMAT DISK
$31	Drive#	$50	To Drive	128/256	1 to 720	WRITE(no vfy)
$31	Drive#	$52	From Drive	128/256	1 to 720	READ
$31	Drive#	$53	From Drive	4	0	READ STATUS
$31	Drive#	$57	To Drive	128/256	1 to 720	WRITE(verify)

An additional command to format a disk in enhanced density is:

Device	Unit	Command	Direction	Byte Ct.	Aux Bytes	Function
$31	Drive#	$22	From Drive	128	0	FORMAT DISK

The byte count is always 128 for a single density drive, and is 128 for the first three sectors (1, 2, and 3) of a double density drive. All other sectors on a double density drive are 256 bytes long.

The FORMAT function is always called with a sector number in the range of 4 to 720. It expects 128 bytes from a single density drive and 256 bytes from a double density drive.

The first byte returned by the READ STATUS command is expected to indicate the sector size -- if bit 5 is a 1 (bit 7 is the sign bit) then the sectors are large (256 bytes), otherwise, they are small (128 bytes).

The auxiliary bytes are treated as an address to a sector on the diskette, and range from 1 to 720 (when in DOS 2 compatible mode) or from 1 to 65,535 (when accessing large capacity disk drives).

The additional functions used to configure disk drives dynamically are:

Device	Unit	Command	Direction	Byte Ct.	Aux Bytes	Function
$31	Drive#	$4E	From Drive	12	1 to 720	READ CFG.
$31	Drive#	$4F	To Drive	12	1 to 720	WRITE CFG.

These commands are used to configure the drives identified as configurable when the computer is booted: if there is a possibility that a drive does not support these functions, it should be defined as not configurable (such as the Atari 810). These commands are used by the "P" command, permitting reconfiguration of a disk drive on demand - to format a diskette, for example. (To format a disk on an Indus drive, issue the "P" command to manually change the density on the drive, then issue the "I" command).

The individual bytes transferred by these commands are defined as follows:

- byte 0: Tracks per side (40 for a standard disk drive)
- byte 1: Disk Drive Step Rate (as defined by Western Digital)
- byte 2: Sectors/Track -- high byte (usually 0)
- byte 3: Sectors/Track -- low byte(18 for standard diskettes)
- byte 4: Side Code (0=single sided, 1=double sided)
- byte 5: Disk Type Code --
  - bit 2: 0=single density, 1=double density
  - bit 1: 0=5 1/4 inch diskette, 1=8 inch diskette drive
- byte 6: High byte of Bytes/Sector (0 for single density)
- byte 7: Low byte of Bytes/Sector (128 for single density)
- byte 8: Translation control
  - bit 7: 1=40 trk. disk I/O on an 80 trk. drive

bit 6: Always 1 (to indicate drive present)
bit 1: 1=Handle sectors 1, 2, and 3 as full size sectors
bit 0: 1=Sectors number 0-17 (for example), not 1-18
bytes 9-11 are not used by MYDOS (see the drive documentation as to how they are to be set -- usually zeroes)

MYDOS 4.50 (unlike version 3 of MYDOS) always issues a read configuration command before writing the configuration to the drive and the contents of bytes 9-11 are written exactly as they were previously read (so they will be unchanged).

An additional change in the usage of this command occurs when a high capacity drive (hard disk) is configured. The configuration data for such a drive is very complex and is usually built into the drive controller or written to a "magic" location on the disk. To support partitioning of very large drives (larger than 16 Megabytes), MYDOS issues a write configuration command with the number of sectors per track set to number of sectors on the disk (as defined in the "O" command) and the number of tracks set to 1. All high capacity disks are large sector drives (using 256 byte sectors).

XI. RAMDISK INTERFACE

The driver built into MYDOS is intended to eliminate the need for a "driver" to use common RAMdisks. The required characteristics of the hardware can be most easily described by explaining what is done to access a "sector" of information in the extended RAM.

(1) The sector number is divided by 128, and the remainder is then multiplied by 128 and added to 16384 to get the starting address of the sector in memory (it will be between $4000 and $7F00).
(2) The quotient is used to index into a page table with one entry for each 16K that can be mapped into the memory area from $4000 to $7FFF.
(3) The value from the page table is "AND"ed with the contents of the mapping register, and rewritten to the mapping register.
(4) The data is moved to(from) the area addressed above from(to) the sector buffers at the high end of MYDOS.
(5) The mapping register is restored to its non-mapping state by "OR"ing the restore value with the mapping register and rewriting the result to the mapping register.

Note that this design forces the RAMdisk to be single density and no larger than 4 megabytes (256 pages of 16384 bytes each). Out of that, MYDOS can only accommodate 1 megabyte, because its table is only 64 bytes long.

As you can see, the parameters are the mapping register address ($CFFF for Axlon boards and $D301 for the Atari 130XE), the value "OR"ed into the register to reset the system back to normal ($FF for

the Axlon and $00 for the Atari 130XE types), and the actual map values. These values are determined by first identifying the bits in the mapping register to be left unchanged and setting them to "1" in each of the register values. Second, the remaining bits are filled in with all the legal combinations of mapping bits. The values for the Newell Industries 256K upgrade (which uses the 130XE mapping, more or less) are given here as an example -- future versions of this board and other memory expansion products are not necessarily going to use the same design.

Bits: 7	6	5	4	3	2	1	0	
1	x	x	x	x	x	1	1	First, set bits 7, 1 & 0 in all the mapping values
1	0	0	0	0	0	1	1	These are the 12 (of 32)
1	0	0	0	0	1	1	1	
1	0	0	0	1	0	1	1	
1	0	0	0	1	1	1	1	
1	1	0	0	0	0	1	1	
1	1	0	0	0	1	1	1	
1	1	0	0	1	0	1	1	
1	1	0	0	1	1	1	1	
1	1	1	0	0	0	1	1	
1	1	1	0	0	1	1	1	
1	1	1	0	1	0	1	1	
1	1	1	0	1	1	1	1	

Lastly, since the mapping register at $D301 can be read as well as written, it can be left exactly as it was before we used it by "OR"ing the initial value with zero (leaving it unchanged). The sequence is then: 83, 87, 8B, 8F, C3, C7, CB, CF, E3, E7, EB, EF, 0.

XIII. ERROR CODES AND THEIR SOURCES

```
  3   Last byte of file read, next read will return EOF (MYDOS)
128   Break Abort (OS ROMs)
129   IOCB already open (OS ROMs)
130   No such device defined in the system (OS ROMs)
131   Write-only IOCB, cannot read (OS ROMs)
132   Invalid command (OS ROMs)
133   Device or File not open (OS ROMs)
134   Invalid IOCB reference (OS ROMs)
135   Read-only IOCB, cannot write (OS ROMs)
136   Attempt to read past end of file (MYDOS)
137   Truncated record (OS ROMs)
138   Device Timeout (OS ROMs)
139   Device NAK (serial bus failure, OS ROMs)
141   Cursor out of range for graphics mode (OS ROMs)
142   Data frame overrun (serial bus failure, OS ROMs)
143   Data frame checksum error (serial bus failure, OS ROMs)
144   Device I/O error (in peripheral hardware, OS ROMs)
146   Function not provided by handler (OS ROMs)
147   Insufficient RAM for graphics mode selected (OS ROMs)
160   Invalid Unit/Drive Number, zero or greater than 7 (both
      MYDOS and OS ROMs)
161   No sector buffer available, too many open files (MYDOS)
162   Disk full, cannot allocate space for output file (MYDOS)
163*  Write protected or system error - disk is not readable (MYDOS)
164   File number in link does not match the file's directory
      location (MYDOS)
165   Invalid file name (MYDOS)
166   Byte not within file, invalid POINT request (MYDOS)
167   File locked, cannot be altered (MYDOS)
168   Invalid IOCB (MYDOS and OS ROMs)
169   Directory full, cannot create a 65-th entry in a directory
      -- entries may be used for "lost" as well as real
      files (MYDOS)
170   File not in directory, cannot be opened for input (MYDOS)
171   IOCB not open (MYDOS and OS ROMs)
172*  File or directory of same name already exists in parent
      directory, cannot create (MYDOS)
173   Bad diskette or drive, cannot format diskette (MYDOS)
174*  Directory not in parent directory (MYDOS)
175*  Directory not empty, cannot delete (MYDOS)
180*  Not a binary file
181*  Invalid address range for loading a binary file, END<BEGIN
(MYDOS)

* -- New error codes, not present or different in Atari DOS 2.
```

Most error codes are identical to those returned from ATARI DOS 2; the differences result from the expanded capabilities of MYDOS. Specifically, Error 164, indicating a file number mis-match, only occurs if the file is written in DOS 2 or DOS 2.5 format. Errors 180 and 181 can only occur when XIO 39 (or 40) is invoked to load a file.

Errors 172 and 175 apply to creating and deleting directories and have no ATARI DOS 2 equivalent; error 174 applies to accessing files in subdirectories, so it also has no ATARI DOS 2 equivalent. Error code 173 serves the same function as it did in ATARI DOS 2, but is returned more often (to identify bad diskettes more reliably).