About the Disk Driver

The Disk Driver is a specialized device driver in ROM that you use for storing and retrieving information on Macintosh floppy and hard drives. That information is stored in 512K-byte sectors consisting of an address mark that shows the position of the sector on the disk and a data mark that holds the data for that sector.

Consecutive sectors are organized into tracks--each of which represents a ring of constant radius around the disk.

Applications read and write data in sector-sized chunks so it must specify reads and writes in multiples of 512. The Disk Driver calculates which sector to access. Your application also specifies where on the disk the data should be read or written by providing a positioning mode and a positioning offset. You can read and write data: at the current sector; from a position relative to the current sector; or from a position relative to the beginning of the first sector on the disk. The positioning mode constants are defined as:

fsAtMark	=	0	current sector
fsFromStart	=	1	relative to first sector
fsFromMark	=	3	relative to current sector

Remember that relative positions are always given in multiples of 512.

On top of the 512 bytes of address and data information in each sector, there are also 12 bytes of file tags for floppy drives and 20 bytes of file tags for hard drives. They are designed to let you rebuild files when their directory information (or other file-access data) has been destroyed. Whenever the Disk Driver reads a sector from a disk, it places the sector's file tags in a file tags buffer. Whenever a sector's file tags are written to the buffer, the previous tag information is overwritten. Going the other way, whenever the Disk Driver writes a sector to the disk, it takes the 12 or 20 bytes (depending on the kind of disk drive involved) in the buffer and writes them to the disk along with the sector's address and data information.

The Disk Driver disables interrupts during disk access. It uses the time while interrupts are disabled to store any serial data received from the modem port. Later it passes the data to the Serial Driver. The result is that you can receive messages over communications links without generating hardware overrun errors.

The Disk Driver opens automatically when the system starts up. It obtains space in the system heap for variables, installs entries in the drive queue for every drive on the computer and installs a task in the vertical retrace queue. Writing data to a disk involves calling **Device Manager**'s write procedure and passing the following parameters: floppy drive reference number (either single- or double-sided) -5; hard drive reference number -2; logical (not hard-coded) drive number in consecutive, sequential order starting at 1 and running until all the drives have been numbered; a positioning mode to show where on the disk the new information goes; a positioning offset in multiples of 512; a buffer with the data you want to write; and the number of bytes to be written.

In the event of an error, one of the following codes will be returned;

noErr	(0)	No error
nsDrvErr	(-56)	No such drive
paramErr	(-50)	Bad positioning information
wPrErr	(-44)	Volume is physically locked
firstdskErr	(-84)	Low-level disk error
through lastDskErr		...lastDskErr is range of disk err codes as follows:

sectNFErr	(-81)	Can't find sector
seekErr	(-80)	Drive error
spdAdjErr	(-79)	Can't correctly adjust disk speed
twoSideErr	(-78)	Tried to read side two in a single-sided drive
initIWMErr	(-77)	Can't initialize disk controller chip
tk0BadErr	(-76)	Can't find track 0
cantStepErr	(-75)	Drive error
wrUnderrun	(-74)	Write underrun occurred
badDBtSlp	(-73)	Bad data mark
badDCksum	(-72)	Bad data mark
noDtaMkErr	(-71)	Can't find data mark
badBtSlpErr	(-70)	Bad address mark
badCksmErr	(-69)	Bad address mark
dataVerErr	(-68)	Read-verify failed
noAdrMkErr	(-67)	Can't find an address mark
noNybErr	(-66)	Disk is probably blank
offLinErr	(-65)	No disk in drive
noDrive	(-64)	Drive not connected

Reading data from a disk involves making a **Device Manager** read call and passing the same parameters as above, except that you give it the buffer you want to read into(including the number of bytes it contains) instead of a buffer to write.

In the event of an error, one of the following codes will be returned:

noErr	(0)	No error
nsDrvErr	(-56)	No such drive
paramErr	(-50)	Bad positioning information
firstdskErr	(-84)	Low-level disk error
through lastDskErr		...lastDskErr is range of disk err codes as follows:

sectNFErr	(-81)	Can't find sector
seekErr	(-80)	Drive error
spdAdjErr	(-79)	Can't correctly adjust disk speed
twoSideErr	(-78)	Tried to read side two in a single-sided drive
initIWMErr	(-77)	Can't initialize disk controller chip
tk0BadErr	(-76)	Can't find track 0
cantStepErr	(-75)	Drive error
wrUnderrun	(-74)	Write underrun occurred
badDBtSlp	(-73)	Bad data mark
badDCksum	(-72)	Bad data mark
noDtaMkErr	(-71)	Can't find data mark
badBtSlpErr	(-70)	Bad address mark
badCksmErr	(-69)	Bad address mark
dataVerErr	(-68)	Read-verify failed
noAdrMkErr	(-67)	Can't find an address mark
noNybErr	(-66)	Disk is probably blank
offLinErr	(-65)	No disk in drive
noDrive	(-64)	Drive not connected

Make a read call immediately after a write cal to verify that the information was transferred correctly. A read-verify operation is much the same as a standard read call except that it contains a constant (rdVerify = 64) in the positioning mode. Additionally, a dataVerErr (-68) code does not match, will be returned if an data mismatch occurs.