13.

File System

The GEOS file system is based on the normal c64 DOS file system. A combination of two factors led to an augmentation of the basic structure: first, the c64 was not originally designed to be a disk computer, and second, the addition of the diskTurbo now makes it practical to read and write parts of a file as needed. Previously the slowness of the disk drive often meant that files were read in at the beginning of execution, and not written until exiting the program. If file writes had to be done in the middle of execution, a coffee break was usually warranted.

GEOS supports two different types of files. The first is similar to regular c64 files and is called a SEQUENTIAL* file. This type of file is made up of a chain of sectors on the disk. The first two bytes of each sector contain a track and sector pointer to the next sector on the disk, except for the last sector which contains $00 in the first byte to indicate that it is the last block, and an index to the last valid data byte in the sector in the second byte. The

* SEQUENTIAL stands for any non-VLIR file in GEOS, and should not be confused with the SEQ c64 file format. In fact USR, PRG and SEQ c64 files are all qualify as GEOS SEQUENTIAL file types.

second type of file is a new structure, called a Variable Length Indexed Record, or VLIR for short. An additional block, called a Header Block, is added to both VLIR SEQUENTIAL files. It contains an icon graphic for the file, as well as other data as discussed later.

To understand GEOS files, one must first understand the Commodore files on which they are based. I refer the reader to any of the several good disk drive books available. I use the Commodore 1541 (or 1571) User's Guide, and The Anatomy of the 1541 Disk Drive (from Abacus Software).

This chapter is divided into three sections. The first, for those already familiar with the 1541, is a brief refresher of the basic Commodore DOS. Second we present GEOS routines for opening and closing disks and dealing with directories and standard files. The final section is devoted to a detailed look at VLIR files.

The Foundation

A c64 disk is divided into 35 tracks. Each track is a narrow band around the disk. Track 1 is at the edge of the disk and track 35 is at the center. Each track is divided into sectors, which are also called blocks. The tracks near the outside edge of the disk are longer and therefore can contain more blocks than those near the center. The Block Distribution by Track table shows the number of sectors in each track.

Block Distribution by Track

Track Number	Range of Sectors	Total Sectors
1 to 17	0 to 20	21
18 to 24	0 to 18	19
25 to 30	0 to 17	18
31 to 35	0 to 16	17

Track 18, the directory track, is used to hold information about the individual files contained on the disk. Sector 0 on this track contains the Block Availability Map (BAM) and the Directory Header. The BAM contains 1 bit for every available block on the disk. The bits corresponding to blocks already allocated to files are set while the bits corresponding to free blocks are cleared. Before the BAM bits is a pointer to the first Directory Block, which is described later. The BAM format is unchanged by GEOS.

The Directory Header contains the disk name, an ID word (to tell different disks apart), and three new elements for GEOS, a GEOS ID string, a track/sector pointer to the Off Page Directory block, and a disk protection byte. The GEOS ID string is contained in an otherwise unused portion of the BAM/Directory Header Block. It identifies the disk as a GEOS disk and identifies the version number, which can be important for data compatability between present and future versions of GEOS. See the BAM Format/Directory Header table below. This string should not be confused with the GEOS Kernal ID and version string at $C000 as described in chapter 1.

The Off Page Directory Block is a new GEOS structure but has the same format as regular c64 Directory Blocks. Directory Blocks hold up to 8 Directory Entries. Each Directory Entry (also known as File Entry because it describes a file), contains information about one file. When a file is moved off the deskTop notepad onto the boarder, the file's Directory Entry is erased from its Directory Block and is copied to the Off Page Directory Block. A buffer in memory is also reserved to save information about each file on the boarder. The Off Page feature exists so that a file can be copied between disks on a one drive system. The Icon for an off page file will remain on the deskTop border when a new disk is opened and the deskTop set to display the contents of the new disk. The file can then be dragged to the notepad from the boarder, thus copying it to the new disk.

The disk protection byte is at byte 189 = OFF_GEOS_DTYPE in the Directory Header. This byte is normally 0, but may be set to 'P', to mark a disk as a Master Disk. GEOS Version 1.3 and beyond deskTops will not allow a Master Disk to be formatted, copied over, or have files deleted from the deskTop notePad. Files may still be moved to the border and deleted from there. This saves GEOS developers from having to replace application disks that have been formatted, or otherwise destoyed by user accident.

Here is the format of the BAM and Directory Header.

	BAM Format/Directory Header		
	Byte	**Contents**	**Definition**
	0	18	Track of first Directory Block. Always 18.
	1	1	Sector of first Directory Block. Always 1.
	2	65	ASCII char A indicating 1541 disk format.
	3		Ignored
BAM Track 1	4		Number of sectors in Track 1
	5		Track 1 BAM sectors 0-7
	6		Track 1, BAM for sectors 8-16
	7		Track 1, BAM for sectors 17-20
BAM Track 2	4		Number of sectors in Track 1
	5		Track 1 BAM sectors 0-7
	6		Track 1, BAM for sectors 8-16
	7		Track 1, BAM for sectors 17-20
...BAM for tracks 3-34...			
BAM Track 35	4		Number of sectors in Track 1
	5		Track 1 BAM sectors 0-7
	6		Track 1, BAM for sectors 8-16
	7		Track 1, BAM for sectors 17-20

DIRECTORY HEADER	144-159		Disk name, padded with CHR $A0
	160-161	$A0	Shifted spaces, CHR $A0
	162-163		Disk ID word
	154	$A0	Shifted spaces, CHR $A0
	165-166	$32,$41	ASCII of $2A, DOS version, & Disk format
	167-170	$A0	Shifted spaces, CHR $A0
	171-172		Tr/Sr of off page Directory Block
	173-188		GEOS ID string. "GEOS format V1.2"
	-189	0 ,'P'	'P' indicates protected MasterDisk
	190-255	0	Unused

The format of the Directory Block is shown below. The overall structure of a Directory Block is unchanged. The following table was taken from the c64 disk drive manual.

Directory Block Structure Dir. Blocks may appear on Track 18, Sectors 1 - 19	
0 - 1	Track and Sector of next Directory Block
2 - 31	**Directory Entry 1**
32 - 33	Unused
34 - 63	**Directory Entry 2**
32 - 33	Unused
66 - 95	**Directory Entry 3**
32 - 33	Unused
98 - 127	**Directory Entry 4**
32 - 33	Unused
130 - 159	**Directory Entry 5**
32 - 33	Unused
162 - 191	**Directory Entry 6**
32 - 33	Unused
194 - 223	**Directory Entry 7**
32 - 33	Unused
226 - 255	**Directory Entry 8**

Several unused bytes in each Directory Entry have been taken for use by GEOS. Bytes 1 and 2 point to the first data block in the file unless the file is a GEOS VLIR file. In this case these bytes point to the VLIR file's index table. Bytes 19 and 20 point to a new GEOS table, the File Header block as described below. Bytes 21 and 22 are used to convey the GEOS structure and type of the the file. The structure byte indicates how the data is organized on disk: 0 for SEQUENTIAL, or 1 for VLIR. The file type refers to what the file is used for, DATA, BASIC, APPLICATION and other types as listed in the table below. The SYSTEM_ BOOT file type should only be used by GEOS Boot and Kernal files themselves.

The TEMPORARY file type is for swap files. All files of type TEMPORARY are automatically deleted from any disk opened by the diskTop. The deskTop assumes they were left there by accident, usually when an application crashes and a swap file is left behind. When creating additional swap files, use the TEMPORARY file type and start the filename with the character PLAINTEXT. Example:

```
swapName: .byte PLAINTEXT,"My swap file",0
```

This will cause the file to print in plain text on the desk top and will prevent a user file with the same name to be accidentally removed when "My swap file" is created. Finally bytes 23 through 27 are used to hold a time and day stamp so that files may be dated.

	DIRECTORY ENTRY (Also Known as File Entry)
0	c64 File Type: 0=DELeted, 1=SEQuential, 2=PRoGram, 3=USeR, 4=RELative. Bit 6=Used as Write-Protect Bit.
1	Track and Sector of First Data Block in This File.
2	If the File Is VLIR then this Word Points to the Index Table Block.
3 ••• 18	16 Character File Name Padded with Shift Spaces $A0
19 - 20	Track and Sector of GEOS File Header (New Structure)
21	GEOS File Structure Type: 0=SEQuential, 1=VLIR
22	GEOS File Types: 0=NOT_GEOS, 1=BASIC, 2=ASSEMBLY, 3=DATA, 4=SYSTEM, 5=DESK_ACC, 6=APPLICATION, 7=APPL_DATA, 8=FONT, 9=PRINTER, 10=INPUT_DEVICE, 11=DISK_DEVICE, 12=SYSTEM_BOOT, 13=TEMPORARY (for Swap Files)
23	Date: Year Last Modified, Offset from 1900
24	Date: Month Last Modified (1-12)
25	Date: Day Last Modified (1-31)
26	Date: Hour Last Modified (0-23)
27	Date: Minute Last Modified (0-59)
28 - 29	Low Byte, High Byte: Number of Blocks (Sectors) in the File

Header Block

The GEOS File Header Block was created to hold the icon picture and other information that is handy for GEOS to have around. Something worth bringing attention to is that the File Header Block is pointed to by bytes 19 and 20 of the file's Directory Entry. Thus any c64 SEQUENTIAL file may have a header block. (Bytes 19 and 20 were previously used to point to the first side sector in a c64 DOS relative file, so these bytes are unused in a SEQUENTIAL file.) Bytes 0 and 1 in any block usually point to the next block in the file, or the offset to the last data byte in the last block. The Header Block is not a file, just an extra block associated with a file. Bytes 0 and 1 are set to 00,FF, to indicates that no blocks follow.

We follow the header block diagram below by a complete description of it contents.

GEOS File Header Structure

(128 bytes. New GEOS file. Pointed to by Directory Entry)

Byte No	Contents	Description
0 -1	00,FF	00 indicates this is the last block in the file. FF is the index to the last valid data byte in the block.
2	3	Width of icon in bytes, always 3
3	21	Height of file icon in lines, always 21.
4	$80 + 63	Bit Map data type. Top bit = 1 means the lower 7 bits contain the number of unique bytes which follow, i.e. 63. Always this value.
5 - 67	$FF,$FF,$FF ... $FF,$FF,$FF	Start of picture data total of 63 bytes used to define icon graphic End of picture data
68	$80 + PRG	C-64 file type, used when saving the file under GEOS PRG = 1 , SEQ = 2, USR = 3, REL = 4. Bit 6 = 1 Write Protected.
69	$02	GEOS file type: BASIC = 1, ASSEMBLY = 2, DATA = 3, SYSTEM = 4, DESK_ACC = 5, APPLICATION = 6, APPL_DATA = 7, FONT = 8, PRINTER = 9, INPUT_DEVICE = 10, DISK_DEVICE = 11, SYSTEM_BOOT=12, TEMPORARY=13
70	0	GEOS structure type, 1 = VLIR, 0 = SEQUENTIAL
71 - 72	FileStart	Start address in memory for loading the program
73 - 74	FileEnd	End address in memory for loading the program
75 - 76	InitProg	Address of initialization routine to call after loading the program
77 - 96	Filename 0,1,.,2,0,0,	20 byte ASCII application filename. Bytes 0-11 = the name padded with spaces; 12-15=version string "V1.3"; 16-20=0's
97 - 116	Parent Disk Author Name	If Data file, 20 byte ASCII filename of parent application's disk. If application program, holds name of software designer.
117 - 136	Parent Application	If Data file, 20 byte parent application filename. Bytes 0-11=name padded with spaces; 12-15=version string "V1.3"; 16-20=0's
137 - 159	Application	23 bytes for application use..
160 -255	Get Info	Used for the file menu option getInfo. String must be null terminated and null must be first char if string is empty.

Bytes 2 and 3 contain the width and height of the icon data that follows. File icons are always 3 bytes wide by21 scan lines high. The two dimension bytes precede the data because the internal routine used by GEOS to draw icons is a general routine for drawing any size icon and it expects the two bytes to be there. Bytes 4 through 67 contain the picture data for the icon in compacted bit-map format. Byte 4 is the bitmap format byte. There are three compacted bit-map formats. The second format as described in BitmapUp, is a straight uncompacted bit-map. To indicate this format, the format byte should be be within the range 128 to 220. The number of bytes in the bit-map is the value of this format byte minus 128. Since the value of the highest bit is 128, the lower 7 bits, up to a value of 92 indicate the number of bytes that follow.

The lowest 3 bits of byte 68 is the old c64 file type, PRG, SEQ, USR, or REL. Byte 69 is the GEOS file type. Presently there are 11 different GEOS file types. There may be additional file types added later, but these will most likely be application data files and will be lumped together under APPL_DATA. Byte 70 is the GEOS file structure type. This is either VLIR or SEQUENTIAL. (Remember, a SEQUENTIAL GEOS file is just a linked chain of disk blocks. It does not mean a c64 SEQ file.)

Bytes 71-72 is the starting address at which to load the file. Normally, GEOS will load a file starting at the address specified in bytes 71-72. Later we will see how an alternate address can be specified. This is sometimes useful for loading a data file into different places in memory. Bytes 73-74 contain the word length address of the last byte to be loaded from or saved to disk. This word serves several purposes. First, GEOS will compute the length of the file and determine if there is enough room to save it to disk. The end-of-file address variable should, therefore, should be updated by the application in the case of a data file that grows. Second, if the file is an ASSEMBLY or BASIC file, then GEOS uses the end-of-file address to determine if it will fit in memory without wiping out the GEOS diskTurbo code. If the file fits then it can be fast loaded, otherwise GEOS will use the normal slow c64 Kernal/BASIC routines.

If the file is a BASIC, ASSEMBLY, APPLICATION, or DESK_ACC, then it is as executable file. The deskTop will look at the word comprising bytes 75-76 for the address to start execution at after the file has been loaded. Usually this is the same as the start address for loading the file, but need not be.

The next 20 bytes store the Permanent Name String. Though there are 20 bytes allocated for this string, the last 4 bytes should always be null (0). This was done to maintain compatibility with other places filenames are stored, namely in the Directory Entry.

Bytes 0-11 are used for the file name and padded with spaces if necessary. Bytes 10 to 15 should be the version number of the file. We have developed the convention that version numbers follow the format: V1.0 where 'V' is just a capital ascii V followed by the major and minor version numbers separated by an ASCII period.

Some kind of permanent name for a file is necessary since the user can rename files at will. geoWrite needs to be able to tell, for example, that a geoWrite 1.0 data file is in fact a geoWrite data file, and that it is version 1.0, even if it is named "Suzy Wong at the Beach".

Following the Permanent Name String are two strings that can be used, in the case of a data file to get to the application used to create it. Like the Permanent Name String, the first 12 characters of each of these two strings store the name and the next four characters store a version number. The last four characters are not used. The first of the 2 strings, the Parent Application Disk name in bytes 97-112, contain as you might guess, the name of disk that contains the parent application. Presently this string is not used by GEOS applications.

When GEOS needs to locate an application it looks at the the Parent Application String in bytes 117-132 . When a user double clicks on a data file, GEOS will look at the Parent Application String and try to find a file of that name. If it cannot such a file on the current disk, it will ask the user to insert a disk containing an application file of that name, "Please insert a disk with geoWrite." When looking for an application, GEOS will only check the first 12 letters of the name, the filename, and will ignore the Version Number for the time being. GEOS assumes that the user will have inserted the version of the application he wants to use. In making this assumption, GEOS tacitly assumes that applications will be be downwardly compatible with data files created by earlier versions of the same application. This need not absolutely be the case as will be seen below.

When the application is loaded and begins executing, it should look at the Permanent Name String of the data file. Normally this string will be the same as the Parent Application Name with the exception that the version numbers may be different. Thus if you double click on a geoWrite V1.2 data file and insert a disk containing geoWrite V2.0, the deskTop, which doesn't compare version numbers, will load and start executing geoWrite 2.0. geoWrite will then look at the version number in the data file's Permanent Name String and determine if a conversion of data file formats needs to take place. If there were changes between the V1.2 and 2.0 versions of the data files then the data will have to be converted.

It is much more likely for the code of a program program to change – to fix bugs – that it is for the data file format to change. Data format version numbers then tend to leapfrog application numbers. For example, application X starts out with V1.0. After a month of beta test V1.1 is released. After 1 week of retail shipping a bug is found and a running production change to V1.2 is made and users with V1.1 are upgraded. Meanwhile the data file format is still V1.0; any version of the application can use it. Six months later V2.0 is released with greatly expanded capabilities and a new data format. The data Version Number should then change to V2.0, leapfrogging V1.1, and V1.2. This will indicate to V1.0 to V1.2 versions of the proram that they cannot read the new format. If the user has the newer version of the program than he should be using it and not an older version.

It is up to the application in its initialization code to look at the data file's version number and determine whether or not it can handle it, and if so whether or not the data needs to be converted.

Permanent Name Example

As an example, suppose the user double clicks on a geoWrite 1.0 document. The deskTop will look for a file with the name stored in the Parent Application String. If this program is not found on the current disk the deskTop will ask the user to insert a disk containing it. The deskTop only looks at the first 12 characters and will ignore the version number. After loading geoWrite, control is passed to the application. The deskTop passes a few appropriate flags and a character string containing the name of the data file. The application, in this case geoWrite, will look at the data file's Permanent Name String and especially its Version Number and determines if it can read the file, or if it needs to convert it to the more up-to-date version.. Similarly, if an older version of an application, e.g. ,geoWrite 1.0, cannot read a data file created with a newer version of the application, it needs to cancel itself and return to the deskTop or request another disk.

Constants for Accessing Table Values

Constants that are used with the file system and tables described above are included in the GEOS Constants file in the Appendix. These constants make code easier to read and support, and therefore are included here. Most of the constants are for indexing to specific elements of the file tables presented above. The constants are broken down into the following sections, GEOS File Types, Standard Commodore file types, Directory Header, Directory Entry, File Header, and Disk constants.

Disk Variables

When an application first gets called there is already some information waiting for it. Several variables maintained by the deskTop for its own use are still available to the application when it is run. Other variables are set up by the deskTop in the process of loading the application. This subsection covers all the variables an application may expect to be waiting for it when it is first run. This information set up for desk accessories is slightly different. For more details on running desk accessories see the routines GetFile and LdDeskAcc later in this chapter.

Several variables necessary to talk to the drive are available to the application. There is a variable curDrive, and curDevice. curDrive contains the number of the drive containing the application's disk, either 8 or 9. curDevice is location $00BA where the c64 keeps the device number of the current device. When first run, curDevice and curDrive will be the same. The ID bytes for the disk containing the application are in the drive as one might expect.

Numerous variables are set up during the process of loading an application. The first group of these have to do with how the application was selected by the user. If the user double clicked the mouse pointer on a data file, GEOS will load the application and pass it the name of the data file. The application may then knows which data file to use. A bit is set in r0L to indicate if a data has been specified. If this is the case, r3 will point to the filename of the data file, and r2 will point to a string containing the name of the disk which contains the data file. An application may have also been run merely in order to print a data file. Another bit is used in r0L to indicate this.

r0L -loadOpt flag

Bit 1 (application files only)

0	-	no data file specified
1	-	(contant for this bit is ST_LD_DATA) data file was double-clicked on and this application is its parent.

Bit 6 (application files only)

0	-	no printing
1	-	(constant for this bit is ST_PR_DATA) The deskTop sets this bit is set when the user clicked on a data file and then selected print from the file menu. The application prints the file and exits.

r2 and r3 are valid only if bits 1 and/or 6 in r0L are set.

r2 - Pointer to name of disk containing data file. Points to dataDisk-Name, a buffer containing the name of the disk which in turn contains a data file for use with the application we are loading. The application can then process the data file as indicted by bit 6 of r0L.

r3 - Pointer to data filename string. r3 contains a pointer to a filename buffer, dataFileName that holds the filename of the data file to be used with the application.

The Directory Entry and Directory Header are also available in memory as is the File Header Block.

dirEntryBuf -	Directory Entry for file
curDirHead -	The Directory Header of the disk containing the file.
fileHeader -	Contains the GEOS File Header Block.

There is also a table created as the file is read that contains the track and sector of each block of the file. This table is called fileTrScTab. It is one block long.

fileTrScTab - List of track/sector for file. Max file size is 127 blocks (32,258 bytes).

The first word of fileTrScTab is the track/sector of File Header block. The following bytes contain the track and sector list for the remaining blocks.

r5L - Offset from the beginning of fileTrScTab to the last track/sector entry in fileTrScTab

We now turn to discussing the actual routines used to access the disk. The next section presents an overview of how to use the disk routines, and how to use the serial bus with GEOS.

Using GEOS Disk Access Routines

The GEOS Kernal contains a multitude of disk routines. These routines span a range of uses, from general powerful routines, to specific primative routines. Most applications use only a handful out of the collection, mostly the general high-level routines. Other applications need more exacting level of disk interaction and so an intermediate level of disk access routine is provided. These are routines used by the high level routines to do what they do, and can be used to create other functions.

Finally the most primitive routines are interesting only to those who want to access a serial device other than a printer or disk drive, use the c64 DOS disk routines, or create a highly custom disk routines, a nonverified write for example.

Basic Disk Access

When running GEOS, only one device at a time may be selected on the serial bus. Usually this is one of the disk drives, A or B, but it may also be a printer or other device. The routine SetDevice is used to to change the currently selected device. You pass SetDevice the number of the device, 8 or 9 for you want to have access to the serial bus.

After selecting the device with SetDevice, call OpenDisk to initiate access to the disk. OpenDisk initializes both the drive's memory and various GEOS Kernal variables for accessing files on the disk.

Once the disk has been opened, the programmer may call any of the following high-level routines.

High-Level Disk Routines

GetPtrCurDkNm	-	returns a pointer to a buffer containing the name of the disk.
SetGEOS Disk	-	Converts a normal c64 DOS disk to a GEOS disk by allocating an Off-Page Directory Block.
CheckDkGEOS	-	Checks to see if the current disk is a GEOS disk.
FindFTypes	-	Generates a list of all files on the disk of a specific type.
GetFile	-	Given the name of file, it will load a GEOS data file application or desk accessory files will be loaded and executed.
FindFile	-	Searches the disk for the file. Returns its Directory Entry.
DeleteFile	-	Deletes a file from the disk.
SaveFile	-	Saves a GEOS file to disk.
RenameFile	-	Gives a file a new name.

CalcBlocksFree	-	See how many free block there are left on disk.
EnterDeskTop	-	Quit the application and return to the GEOS deskTop
VLIR Routines	-	See the VLIR chapter.

Intermediate Routines

The routines above handle many of the functions required of an operating system, but by themselves are by no means complete. These high-level routines are implemented on top of a functionally complete set of intermediate-level routines that may be used to implement any other function needed. For example, there are no routines for formatting disks, copying disks, or copying files in the GEOS Kernal. Most applications have little need for copying disks or files and so these function were not included in the Kernal. Instead, these functions are provided by the deskTop. The deskTop is an application like any other such as geoWrite or geoPaint, except that the deskTop is a file manipulation application, and not an editor. The copy and validate functions available in the deskTop are implemented by using more the intermediate GEOS Kernal routines.

Care must be taken when using these routines to make sure that all entry requirements are met before calling them. Calling one of these routines without the proper variables and/or tables set up may trash the disk, crash the system, or both. In particular, a block is set aside in the GEOS Kernal to contain a copy of the disk's Directory Header. Some of the routines expect this block, curDirHead, to be valid, and if any values were changed by the routine it will be necessary to write the header back to disk afterwards. Below is a list in decreasing order of usefulness of these more primitive routines.

Findfile	-	Returns a file's Directory Entry.
GetBlock	-	Reads a block from disk
PutBlock	-	Write a block to disk, and verifies it.
GetFHdrInfo	-	Given a Directory Entry, fetches the file's File Header block.
ReadFile	-	Reads a track/sector linked chain of blocks from disk.
WriteFile	-	writes memory data out to a linked chain of blocks on disk.
ReadByte	-	Simulates reading a byte at a time from a chain of blocks.
GetDirHead	-	Read the Directory Header and BAM from disk.
PutDirHead	-	Writes the Directory Header and BAM back to disk.

NewDisk	-	Initialize the drive for reading off a new disk without changing GEOS variables.
LdApplic	-	Loads and runs a GEOS application.
LdDeskAcc	-	Loads and runs a GEOS desk accessory.
LdFile	-	Loads a GEOS file.
GetFreeDirBlk	-	Get a free Directory Entry. Allocate a new Directory Block if necessary.
BlkAlloc, NextBlkAlloc	-	Allocate a chain of blocks on the disk.
SetNextFree	-	Allocates a free block on disk.
FreeBlock	-	Frees up one block on disk.
SetGDirEntry, BldGDirEntry	-	Create a Directory Entry from a Header Block.
FollowChain	-	Create the track/sector list in fileTrScTab for a chain of blocks.
FastDelFile	-	Frees blocks indicated by the track/sector list in fileTrScTab.
FindBAMBit	-	Returns information about a block.
FreeFile	-	Free all blocks in a file. Leave the Directory Entry intact.
ChangeDiskDevice	-	Changes the device number (8 or 9) the drive responds to.

The Most Primitive Level

An even more primitive level of routines is also available. There are only three reasons one might have for using these routines.

1. To access the standard c64 DOS routines. As mentioned before, the deskTop does this to access the formatting routines.
2. To talk to a device other than the disk drive or printer.
3. To write highly optimized disk routines for moving large numbers of blocks around that are ordered on the disk in some unusual way. The routines in the previous sections for reading and writing a linked chain of blocks on disk are almost always sufficient.

These are all ways you might want to use the serial bus that are outside the realm of what GEOS supports directly. The low level routines below are provided to allow safe access to the serial bus, and a safe return to GEOS disk usage.

InitForIO	-	Turn off all interrupts, disable sprites, bank switch the c64 Kernal and I/O space in.
DoneWithIO	-	Restore interrupts, enable sprites, and switch in the previous RAM configuration.
PurgeTurbo	-	Normally the turbo code is always running. PurgeTurbo removes the turbo code resident in the disk drive and returns control of the serial bus to the c64 DOS.
EnterTurbo	-	Uploads the turbo code to the drive and starts it running.
ReadBlock	-	Read a block from disk. Turbo code must already be running, and InitForIO must have been called.
WriteBlock	-	Write a block to disk. No verify is done, the Turbo code must be running, and InitForIO must have been called.
VerWriteBlock	-	Same as WriteBlock except that the block is verified after writing.

Accessing the Serial Bus

Follow the procedure below to use the c64 serial bus.

1. Call SetDevice to set up the device you want to use. SetDevice will give the serial bus to whatever device you request.
2. If you want to use c64 DOS disk routines, then you will have to turn off the disk turbo code running in the drive. To do this, call PurgeTurbo. If not using the c64 DOS routines skip this step.
3. Call InitForIO to turn off interrupts, sprites and set the I/O space and c64 Kernal in.
4. Call any of the standard c64 DOS serial bus routines to access the serial device on the bus.
5. When finished with the bus, call DoneWithIO. This sets the system configuration back to what it was before you called InitForIO. The next GEOS disk routine that you call (except for ReadBlock, WriteBlock, or VerWrBlock) will automatically restart the diskTurbo.