MetaDOS™/CD-ROM Developer's Documentation

First Edition
January 3, 1989

Atari Corporation
1196 Borregas Avenue
Sunnyvale, CA 94086

This document was produced with an Atari Mega 4 computer using Microsoft Write and an Atari SLM804 laser printer.

Table of Contents

MetaDOS™

Introduction

MetaDOS™ is an Atari TOS system extension that uses a standard method for loading and executing multiple devices and drivers on the Atari ST or Mega computer systems. Through these extensions and the proper software driver, the ST computer systems can be configured to add any new devices and drivers in an orderly fashion. MetaDOS™ has the following advantages:

-> Fully GEMDOS compatible.
-> Improved throughput via one system request handler.
-> Allows addition of any foreign electrically-compatible device.
-> Controls auto-boot sequence.
-> Maps multiple drivers to multiple physical devices.
-> Extends logical devices (up to Z:).
-> Allows solution to existing GEMDOS 16 MEG partitions.

MetaDOS™ is broken up into three sections. MetaDOS™ is the controller. It receives requests from the operating system and routes the request to the appropriate logical DOS translator. The DOS handler is the second section of MetaDOS™. It is responsible for translation between the physical format of the device and the GEMDOS interface. Different DOS handlers can be created to interpret the different formats the physical device can read. For instance with a CD-ROM, all discs are identical. This means that all CD-ROM players can read all CD-ROM discs. Although the disc information can be read, this information must be translated from the format in which it was written (High Sierra May86, ISO9660...) into the format the computer system expects (i.e., GEMDOS format).

The third section of MetaDOS™ is the addition of physical device drivers. These drivers talk directly to the physical device they were assigned. There are 16 reserved calls for these devices. DOS translators communicate with the physical device via a standard array of 16 calls. Some of these calls are init, open, close, read, write, seek, and status. Some new devices have new features that are not present in todays devices. Six calls have been reserved to handle any new features these devices may present.

The following calls are supported by MetaDOS™ v1.0

Dgetdrv	Dsetdrv	Dgetpath	Dsetpath	Dfree
Dcreate	Ddelete	Fattrib	Fopen	Fclose
Fread	Fwrite	Fseek	Fsfirst	Fsnext
Fcreate	Fdelete	Fdatime	Frename	Fsetdta
Fgetdta	Pexec			

MetaDOS™ v1.0 Limitations

With MetaDOS™ version 1.0 the following limitations apply:

1) Files opened to a MetaDOS™ device MUST be closed by the application. Memory is still released by the operating system once the application terminates.

2) MetaDOS™ devices are not boot-able. Also MetaDOS™ device drivers MUST originate from a standard GEMDOS device.

3) Networks are not supported.

4) Standard I/O is not supported.

Desktop Limitations

Due to the growing amount of data a MetaDOS™ device can contain the following desktop limitations apply:

1) Fields in the "Show Information" dialog boxes may be too small. Some CD-ROM discs have over 100 megabytes of information. This will not fit in the desktop "Bytes Used" line. "Showing Info..." on a CD-ROM disc can result in this line being mysteriously removed from the information box. Reboot the computer to repair the problem.

2) A directory listing of a CD-ROM disc in "Show as Text Mode" may crash the computer system. Again the field used to display the size of a file is too small. Only view a CD-ROM disc in "Icon" mode.

Configuration File

The configure file is used to tell the Atari ST computer systems which devices to load into the Atari system. The configure file has the following features:

Load and execute a normal auto folder program with parameters.
Load new physical device drivers and assign physical id's.
Load in new DOS drivers and assign logical and physical device id's to use.

A config.sys file is located in the computers boot device auto folder. This file is read via MetaDOS™ and parsed for normal auto folder devices, DOS translators or physical device drivers to load. MetaDOS™ uses the information within this file to locate and associate DOS translators and physical device drivers. This file is an ascii file and can be edited by a simple text editor.

The configure file has the following characteristics:

1) Lines in the configure file that start with ';' are comment lines.
2) Blank lines are ignored.
3) Lines starting with '*' are MetaDOS™ device drivers.
4) Lines starting with a letter 'A-Z' are executed as auto-exec programs.

<u>Example Config.sys file</u>

```
; Config.sys   An ascii file that describes the system configuration for a
;              CD-ROM driver to boot from the floppy drive A:
;
*BOS, A:\AUTO\CDAR504.BOS,A:6
*DOS, A:\AUTO\HSMAY86.DRV,H:A
*DOS, A:\AUTO\ISO9660.DRV,I:A

c:\auto\boot\ram512.tos      ; OTHER AUTO FOLDER BOOT PROGRAMS
c:\auto\boot\gdos11.tos
c:\auto\boot\diab630.tos
c:\auto\boot\sdump.tos

; End Configure File
```

The order of the config.sys file is important. MetaDOS™ should be installed first. This way other "auto folder programs" will know of MetaDOS™ existence. In order to guarantee this, MetaDOS™ has the capability of executing normal auto folder programs. To do this, move all "prg" (programs) from the auto folder. Place the full path and the name of the file AFTER the '*' commands in the config.sys file. See config.sys file above. MetaDOS™ will execute the auto-boot programs with a command line input from the config.sys file.

NOTE: To speed up the boot process when using MetaDOS™, move the normal auto folder terminate and stay resident programs to another folder. This way the system will not look at these files to see if they are to be executed.

PHYSICAL (Basic Operation System) Format

The format for installing a physical Basic Operation System is as follows:

```
;*BOS,[Physical Bios driver][Arguments],[Physical id]:[DMA Channel],...
```

In the example above, DEVICE.BOS is loaded by MetaDOS™ from the specified GEMDOS device and path. The program is executed with the command line "MEM(8k)" as input. Once installed, the physical device init routine is called with the physical id "A" and the DMA channel number 6. (The DMA channel number is not necessary for all physical devices. It is only required for DMA devices.)

```
      *BOS, A:\AUTO\BOOT\DEVICE.BOS MEM(8K),A:6
```

NOTE: The physical id is not the same as GEMDOS logical id's (i.e. A: does not mean the floppy disk drive.) With this system TOS can have up to 26 physical devices.

LOGICAL DRIVER (DRV) FORMAT

The format for installing logical device drivers is as follows:

```
;*DOS [Logical DOS][Arguments],[Logical id][:[Physical id],...
```

Again, the example below shows HSMAY86.DRV is loaded by MetaDOS™ from the specified GEMDOS device and path. The program is executed with NO command line input. Once installed, the logical driver init routine is called with logical device "H" to talk to physical device "A".

```
      *DOS, A:\AUTO\HSMAY86.DRV,H:A
```

Another driver can be associated with the same physical device. In this fashion two logical icons can access one device. This way both HSMAY68.DRV and ISO9660.DRV can be installed at the same time, both talking to one CD-ROM unit. The user must install an "I" icon to have access to the ISO9660 driver from the desktop.

```
      *DOS, A:\AUTO\ISO9660.DRV,I:A
```

NOTE: The physical id is not the same as GEMDOS logical id's. In the example above, "I" is the logical system id. The user must install an ICON "I" to have access to the drive from the desktop or application.

MetaDOS™ Requirements and Installation

MetaDOS™ will install on any Atari ST or Mega system. To install MetaDOS™ for the CDAR504 CD-ROM the following files are required:

```
metados.prg -> MetaDOS™ system router.
config.sys  -> System configuration file.
cdar504.bos -> CD-ROM Basic Operation System.
iso9660.drv -> ISO9660 driver (translator).
hsmay86.drv -> High Sierra May86 driver (translator).
```

Copy these files into the AUTO folder of your boot device. On power-up, MetaDOS™ will read the config.sys file for logical and physical driver assignments. (See config.sys description.) It will then load in the physical drivers (.BOS). The DOS translator (.DRV) will issue commands to its assigned physical driver to access the physical device. An installation message is produced if initialization is successful.

Step by Step Installation

To install, do the following:

1) Move all .prg files in the auto folder into another directory on the boot device.

2) Copy the files from the MetaDOS™ distribution disk into the auto folder.

3) If your boot device is "A:" then rename config.a to config.sys, If your boot device is "C:" then rename config.c config.sys.

4) Edit the config.sys file and at the bottom add the other auto folder programs you wish to install. You MUST use the drive letter and full pathname in which they are located.

5) Reboot the computer.

6) After reboot, you must install icon "I" (iso96600) and "H" (High Sierra) to have access to the CD-ROM from the Atari Desktop.

There will be a message displayed indicating MetaDOS™ is installed. The message will look as follows:

```
**********************************
Atari MetaDOS™          RMS,RJZ
Version 1.0             09/14/88
Copyright Atari Corporation   1988
**********************************
CDAR504.BOS v1.0 installed as A: on DMA 6
HSMAY '86 v1.0 installed as H:
ISO9660 v1,0 installed as I:
**********************************
```

<u>Atari MetaDOS™</u>
<u>CD-ROM Bios Extensions</u>

Introduction

This is a description of the Atari CD-ROM Bios Extensions for use with MetaDOS™. MetaDOS™ loads the CD-ROM Bios Extensions via a configuration file and assigns a physical device id. MetaDOS™ intercepts trap #14 and vectors all opcodes between 0x30 (hex) and 0x3F to the assigned physical Bios. To install, see MetaDOS™ configuration file.

Through these bios calls, applications can access the physical device directly. Other calls allow access to the unique features of the particular device. The first seven calls are standard calls for MetaDOS™ Bios Extensions. These are open, close, read, write, and status. The last six calls are device explicit. These calls are used to access new features of the device.

There is no verification on the validity of the inputs to any of the Bios function calls. It is up to the application to keep within the specification of this document. Not doing so can cause strange results or system crash. The CD-ROM Bios is NOT reentrant. Therefore if used in a reentrant fashion (for example, at interrupt) will also cause the system to crash.

The following opcodes are now supported by the CD-ROM MetaDOS™ Bios Extensions:

Op-Code	**Function**	
0x30	Meta_init()	-Returns information on installed Extended Bios and the physical drives online.
0x31	open()	-Opens the physical device and returns information on the device driver.
0x32	close()	-Closes the physical device.
0x33	read()	-Reads a number of blocks.
0x34	write()	-Writes a number of blocks (NOT USED).
0x35	seek()	-Seeks to the physical block number.
0x36	status()	-Returns the current status of the CD-ROM.
0x37	Reserved.	
0x38	Reserved.	
0x39	Reserved.	
0x3A	Reserved.	
0c3B	start_aud()	-Starts audio play.
0c3C	stop_aud()	-Stops audio play.
0x3D	set_songtime()	-Sets begin and end time for song play.
0x3E	get_toc()	-Gets audio Table of Contents information.
0x3F	disc_info()	-Gets disk information.

These functions are available through a trap #14 call. A typical "C" binding for the trap is:

```
.text
_CDROM:
       move.l     (sp)+,savsp
       trap       #14
       move.l     savsp,-(sp)
       rts

.bss:
savsp: .ds.l      1
.end
```

The binds listed within this documentation use the trap call above EXPLICITLY. This shows the order in which parameters are placed on the stack. An example of this type of bind is:

CDROM((int) OPEN, (int) phydrv, (long) buffer);

For the above example, the same bind in assembly language is as follows:

```
pea        buffer             ; Address of buffer.
move.w     phydrv,-(sp)       ; Physical drive id.
move.w     #OPEN,-(sp)        ; Op-code.
jsr        _CDROM             ; Call trap routine.
add.l      #8,sp              ; Clean up stack.
```

* Note: For all of the functions listed, the following convention applies:
32-bits is considered a LONG value.
16-bits is considered an INT (integer) value.
8-bits is considered a BYTE (character) value.

Typical Application Calling Sequence

For a typical application the following sequence will allow applications to be compatible with all future versions of the CD Extended Bios. It will also allow applications that follow these guidelines to operate with any future Bios and MetaDOS™ released by Atari.

Steps

1) Make a **Meta_init()** call. This will return version information on MetaDOS™ as well as a drive map of the physical devices on the system. If the drive map is zero, no physical devices were found on the system or MetaDOS™ has not been installed. (See Meta_init()).

2) Use the physical drive map to find the physical device id. This id will be used in all Extended Bios calls. (See Meta_init()).

3) Make an **open()** call with the physical device id. The physical Bios will then initialize itself and also return information on the given physical device.

4) Use the device id to make any Extended Bios calls.

5) Make a **close()** call with the physical device id to tell the physical Bios that the application is finished with this physical device. This also de-initializes the Extended Bios for this physical device (provided no other application is currently using the physical device).

MetaDOS™ Extended Bios Commands

Meta_init()   (0x30)

This call returns information on the physical drives found on the system and on MetaDOS™ itself.   The buffer passed is 16 bytes long and must be ZERO before the call is made. Once the call has been made if the drive map is zero, no physical units are online or MetaDOS™ is not resident on the system. The physical drive map supports 26 devices (A-Z).

The drive map is a bit map of each physical device on the system. Bit zero (0) represents physical device A; bit one (1) represents physical device B; bit 16 represents physical drive Q. The physical device id is an ascii value representing a letter. For example, if bit 0 is set to one (1) then the physical device id is a capital "A" or the hex value 41.

NOTE: *The physical device id is not the same as GEMDOS logical device ids. Physical device ids communicate directly to the assigned physical device without interpretation of the data returned from the device.*

CDROM(METAINIT, buffer);

Given:

```
int   METAINIT    -Initialization op code (0x30).
long  buffer      -A pointer to a 16 byte buffer where information will
                   be returned.
```

Returns:

```
buffer:
      .ds.l 1     ; Physical Bios drive map (32 bits)
      .ds.l 1     ; Pointer to a null terminated ascii string
                    representing the version of MetaDOS™.
      .ds.l 1     ; Pointer to the physical device link list.
      .ds.l 1     ; Reserved.
```

open()   (0x31)

This call returns information on a given physical device. The physical devices on the system can be found by the Meta_init call. This call also serves to initialize the Bios for this physical device if necessary.

```
long  CDROM(OPEN, phydrv, buffer);
```

Given:

```
int   OPEN        -Open  opcode (0x31).
int   phydrv      -Physical device id (ascii value).
long  buffer      -Pointer to the users transfer buffer. (This MUST be
                   on an even boundary).  The buffer size must be 16
                   bytes long.
```

Returns:

```
      0           -No error condition. The physical device initialized
                   and the buffer filled with the following information.

buffer:
      .ds.l 1     ; Pointer to the device header.
      .ds.l 1     ; Pointer to physical bios id string.*
      .ds.l 1     ; Reserved.
      .ds.l 1     ; Reserved.

      else        -Error (See error codes)
```

*NOTE: For CD-ROM, the first five letters of the physical bios id string is "CDROM"

close()   (0x32)

This call is used to inform the Bios that the application is finished with the physical device. This call also serves to de-initialize the Bios for this physical device if necessary.

```
long  CDROM(CLOSE, phydrv);
```

Given:

```
int   CLOSE       -Close opcode (0x32).
int   phydrv      -Physical device id (ascii value).
```

Returns:

```
      0           -No error condition.
      else        -Error (See error conditions).
```

read()   (0x33)

This function transfers a number of blocks starting at the given block number (blockno) for the given number of blocks (numblks) . Block size is defined to be 2048 bytes (2K). The data is transfered to the user's data buffer directly. This is a block transfer call so the buffer address MUST start on an even boundary. The number of blocks to transfer MUST be 63 or less, therefore a maximum of 126K can be transfered at one time.

long CDROM(READ, phydrv, buffer, blockno, numblks);

Given:

```
int   READ        -Read opcode (0x33).
int   phydrv      -Physical device id (ascii value).
long  buffer      -Pointer to the user's transfer buffer. (This MUST be
                   on an even boundary)
long  blockno     -Starting block to begin transfer.
int   numblks     -Number of blocks to read. (This MUST be 63 or less)
```

Returns:

```
      0           -No error condition
      else        -Error (See error codes)
```

NOTE: *The first block on a CD-ROM disc is block number ZERO.*

write()   (0x34)

This function transfers a number of blocks from the beginning of the buffer (buffer) for the given number of blocks (numblks) . Block size is defined to be 2048 bytes (2K). The data is transfered from the user's data buffer directly. This is a block transfer call so the buffer address MUST start on an even boundary. The number of blocks to transfer MUST be 63 or less, therefore a maximum of 126K can be transfered at one time. For CD-ROM this function is not used.

long CDROM(WRITE, phydrv,buffer,blockno,numblks);

Given:

```
int   WRITE       -Write opcode (0x34).
int   phydrv      -Physical device id (ascii value).
long  buffer      -Pointer to the user's transfer buffer. (This MUST be
                   on an even boundary)
long  blockno     -Starting block to begin transfer.
int   numblks     -Number of blocks to read. (This MUST be 63 or less)
```

Returns:

```
      0           -No error condition
      else        -Error (See error codes)
```

seek() (0x35)

This function will seek the physical unit to a physical address. The function call will return once the seek has been completed or on an error condition.

```
long  CDROM(SEEK, phydrv, blockno);
```

Given:

```
int   SEEK         -Seek opcode (0x35).
int   phydrv       -Physical device id (ascii value).
long  blockno      -Block number to seek to.
```

Returns:

```
      0            -No error condition.
      else         -Error.
```

status()   (0x36)

This function returns the current status of the physical unit. The long returned is broken up into the most significant word and least significant word. If an error exists (error, timeout, or busy) the most significant word will be 0xFFFF indicating an error condition. Also, the most significant bit of the lower word will be set (example busy error: FFFF8004). The programmer should check the bits defined below explicitly to determine the error type. *This is the ONLY function that returns an error code in this form. All other functions return the proper bios error code.*

Bits defined are: | X R R R R R R R R | T R R R M B E 0 |

Status Codes:		
	0	-Reserved (zero).
	E	-Error exists.
	B	-Busy Device Not Ready (Playing audio).
	M	-Media Change Bit (Cleared on first read).
	R	-Reserved.
	T	-Timeout. (No Response).
	X	-Set (1) if error else (0). Set only on error, busy, or timeout condition.

long CDROM(STATUS, phydrv, buffer);

Given:

int	STATUS	-Get status opcode (0x36).
int	phydrv	-Physical device id (ascii value).
long	buffer	-Pointer to a 32 byte buffer where extended status information is returned. NOTE: If zero (0) then no extended status information is returned.

Returns:

X	-Current status condition as defined above.

```
buffer:
      .ds.l 1     ; Current error condition. (See error codes.)
      .ds.l 7     ; Reserved.
```

Example return errors:

FFFF8002	Device error condition.
FFFF8004	Busy error condition.
FFFF800C	Busy and media change condition.
FFFF8080	Timeout error.
00000008	Media changed.

NOTE: *This function should be used to check for media change. The media change bit is only cleared when the first read, extended read, or TOC read is performed.*

start_aud()   (0x3b)

This command allows the application to set the songs to be played in either a sequential or random manner. Up to 10 random selections can be played. Setting bit (1) within the flag indicates random selection mode. The byte_array contains a list of decimal song numbers to play in order. The byte_array should be null terminated. The Bios will only accept up to 10 selections.

If bit (1) is zero then sequential song play is selected. The byte_array contains the number of songs to play and the starting song number.

```
byte_array[ 0] = number of songs to play. (decimal)
byte_array[ 1] = starting song number. (decimal)
```

There is also a short song play flag. In this mode, each song will be played for approximately 10 seconds. This mode can be used by both sequential or random play modes. Setting bit (0) within the flag indicates 10 second play mode.

When an error condition occurs, the error returned is in a slightly different format than normal returned errors. Although the long returned will be negative indicating an error, it will be broken into two parts. The lower word will indicate the error condition (See error codes) and the upper word will contain the index into the byte_array with the most significant bit set to 1.

```
              MS      LS
              Word    Word
Example:    | 80 05 | FF ED |  Error  -Device not responding
                                       on song index 5.
```

long CDROM(STARTAUD, phydrv, flag, byte_array);

Given:

```
int   STARTAUD    -Audio play opcode (0x3b).
int   phydrv      -Physical device id (ascii value).
int   flag        -Mode flag.  This flag has the following format:
                        bit 0       1 -10 second play mode.
                                    0 -Normal play.
                        bit 1       1 -Random selection mode.
                                    0 -Sequential play mode.
                        bit x         -Reserved.
long  byte_array  -Pointer to a null terminated decimal byte array. The
                   Bios will only allow up to 10 songs or will terminate
                   upon finding a null (0) song number.
```

Returns:

```
      0           -No error condition.
      else        -Error.
```

stop_aud()   (0x3c)

The function stops all audio play. All information sent to the CD-ROM unit is lost. This includes random selection information as well as beginning song time information.

long CDROM(STOPAUD, phydrv);

Given:

```
int   STOPAUD     -Stop audio play opcode (0x3c).
int   phydrv      -Physical device id (ascii value).
```

Returns:

```
      0           -No error condition.
      else        -Error.
```

set_songtime()   (0x3d)

This function gives the application a method of setting the song to play via the minute, second, and frame information returned for the GET_TOC call. The start and end times for song play are in binary coded decimal. The upper byte is ignored.

long CDROM(SETSONG, phydrv, flag, start, endtime);

Given:

```
int   SETSONG     -Set song by time opcode.
int   phydrv      -Physical device id (ascii value).
int   flag        -When bit 0 is one (1) then repeat mode is initiated.
                   The given start and end times will be repeated until
                   another command is issued. With bit 0 is cleared (0),
                   no repeat will occur.
long  start       -A long (32 bits) that represents the minute, second,
                   and frame number in BCD.  The MSB (byte) of the long
                   is ignored.  Following the MSB is the minute then the
                   second and finally  LSB is the frame number.  By
                   treating the TOC information as records, the record
                   itself can be used without further modification.
long  end_time    -A long (32 bits) that represents the minute, second,
                   and frame number in BCD.  This is in the same format
                   as start_time.
```

Returns:

```
      0           -No error condition
      else        -Error (See error codes)
```

get_toc()   (0x3e)

This function transfers a maximum of 512 bytes of Table of Contents information for an audio disc. The function transfers the data directly to the user's buffer. This is a block transfer call so the buffer address MUST start on an even boundary. Each TOC record is composed of four bytes and is in a sequential order. The information returned is in BCD format where each byte is one BCD number. The format for the table of contents is as follows:

```
Track Number #1, minute, second, frame
Track Number #2, minute, second, frame
Track Number #3, minute, second, frame
```

The Track number (TNO) is the first byte of each record and is defined as:

TNO	**Meaning**
0x00	The following data has no meaning.
0xA0	The minute determines the number of the first song to play. Second and block are all zero.
0xA1	The minute determines the number of the next song to play. Second and block are all zero.
0xA2	The minute and second show the finishing time of the last song.
(0<TNO<0xA0)	The data is valid and represents the starting time of the song for that track (song) number.

long CDROM(GETTOC, phydrv, flag, buffer);

Given:

int	GETTOC	-Get Table of Contents opcode (0x3e).
int	phydrv	-Physical drive identifier.
int	flag	-When bit zero is zero (0) then the information is returned in a non-edited form. If one (1) then information is edited and returned. *
long	buffer	-Pointer to the users transfer buffer. (This MUST be on an even boundary and has a maximum size of 512 bytes.)

Returns:

0	-No error condition
else	-Error (See error codes)

* By default the TOC information is returned in non-edited form. Sometimes CD-discs will have redundant information on an audio disc. If the edit bit is set then redundant information is removed. If the disc contains more than 39 songs, all the TOC information will not be returned. Use Request Sense command to find out the number of songs.

disc_info()   (0x3f)

This function returns Sub-Q information on the disc in the drive. This information includes the type of disc and track information. The information is returned in a 512 byte buffer provided by the application.

```
long  CDROM(DISCINFO, phydrv, buffer);

Given:
int   DISCINFO    -Disc information opcode (0x3f).
int   phydrv      -Physical device id (ascii value).
int   buffer      -Pointer to a 512 byte buffer where Sub-Q information
                   is returned.

Returns:
      0           -Buffer filled with information.
      else        -Error (See error codes)

buffer:
      .ds.b 1     ; Disk type. 0-audio disc, 1-digital disc.
      .ds.b 1     ; Beginning track (song) number.
      .ds.b 1     ; Ending track (song) number.
      .ds.b 1     ; Current track (song) number.
      .ds.l 1     ; Optical head location relative to the
                  ; beginning of the current track number.*
      .ds.l 1     ; Absolute optical head location relative to
                  ; the beginning of the disc.
      .ds.l 1     ; Starting time of the disc lead-out area or the last
                    data block on the disc.
      .ds.l 124   ; Reserved.
```

* All optical head locations are in packed binary coded decimal (BCD). The long returned contains the minute, second, and frame number as described in the Red Book. The long is formatted as follows: **|00 MM SS FF|** where MM = minute, SS = second, and FF = frame number in BCD. Conversion from Red Book notation (minute, second, frame converted to binary) to a physical block number as defined by the High Sierra proposal is done by the following equation:

physical block = Minute*60*75+Second*75+Frame-150

Error Conditions

The error numbers returned are the same as the ST Bios. All error numbers are negative. For the Bios it ranges from -1 to -31. Some new error conditions have been added. The following is a list of the error conditions:

0	- **Success:** No error condition.
-1	- **Error:** General all-purpose error.
-2	- **Drive Not Ready:** The specified drive is not ready or busy.
-3	- **Unknown Command:** Unimplemented opcode.
-4	- **CRC Error:** Error occurred while reading a block from the device.
-5	- **Bad Request:** The device could not respond to the command or the command parameters are bad.
-6	- **Seek Error:** The device was unable to seek to the given block number.
-7	- **Unknown Media:** The wrong type of media was found for the given command. For CD-ROM this could mean that the application tried to play a Data CD-ROM disc.
-8	- **Sector Not Found:** The given sector number to read was not found on the disc.
-9	- **No Paper:** This error condition is not used with respect to CD-ROM's.
-10	- **Write Fault:** This error condition is not used with read-only drives.
-11	- **Read Fault:** Failed to read the desired block from the device.
-12	- **General Error.**
-13	- **Write Protection:** Attempting to write to write-protected or read-only media.
-14	- **Media Change:** The media has changed since last access.
-15	- **Unknown Device:** The physical device the command was meant for is not found on the system.
-16	- **Bad Sectors:** Not used by the CD-ROM Bios.
-17	- **Insert Other Disk:** Used by GEMDOS for two disk drive simulation mode.
-18	- **Insert Disc:** This indicates that there is no disc in the drive or the drive door is opened.
-19	- **Device Not Responding:** The device the command was sent to is not responding.
-20	- **Hardware Error:** The device indicates that there was some kind of unrecoverable hardware error.

CDAR504 CD-ROM
Command Specifications

This section describes the CD-ROM command set within the firmware of the CD-ROM unit. The commands allow computer control for both CD-Audio and CD-ROM programming of the unit.

Commands are sent to the CD-ROM controller via the Atari Computer System Interface (ACSI) command descriptor block. The Atari Computer System Interface (ACSI) is much like the Small Computer System Interface (SCSI). The command bytes are handshake over the DMA channel. Once the last byte of the command is sent and the DMA channel is enabled, the transfer will begin. When complete, the CD-ROM will acknowledge the transfer. At this point the status byte can be read to determine if the operation was successful.

The ACSI command descriptor block is defined as follows:

```
Byte    Bit Meaning

0      | C C C O O O O O |-> Controller Number(CCC) ,Opcode(OOOOO)
1      | D D D H H H H H |-> Sub-Device(DDD),Block Addr MSB(HHHHH)
2      | M M M M M M M M |-> Block Addr MID (MMMMMMMM)
3      | L L L L L L L L |-> Block Addr LSB (LLLLLLLL)
4      | C C C C C C C C |-> Block Count  (CCCCCCCC)
5      | B B B B B B B B |-> Control Byte (BBBBBBBB)
```

Controller Number
A 3-bit value which designates the physical controller to receive the command block. Up to 8 (0 thru 7) controllers can be selected. This number MUST match the DIP switch setting on the back of the CD-ROM unit.

Opcode
A 5-bit value specifying the operation to be performed by the unit given the information within the command block.

Sub Device Number
A 3-bit number which determines the sub-device to be used. Theoretically each controller can handle up to 8 sub-devices.

Block Address
A 21-bit value which designates the block address which data will be transfered to or from.

Block Count
A 8-bit value which determines the number of data blocks to be transfered.

Control Byte
A 8-bit value which is used for command specific modifiers.

Command List

Opcode	**Command**
0x00	- Test Unit Ready
0x03	- Request Sense
0x05	- Audio Stop
0x06	- Audio Start
0x08	- Read
0x0B	- Seek
0x11	- Audio Program
0x12	- Inquire
0x13	- Disc Spin-down Timer On/Off
0x15	- Mode Select
0x18	- Extended Read
0x19	- Read TOC (Table of Contents)
0x1A	- Mode Sense
0x1B	- Extended Seek
0x1E	- Prevent/Allow Media Removal

Test Unit Ready (0x00)

This command checks to see if the device is in a READY condition or not. It can be used to determine whether audio play is finished. The response from this command ONLY SENDS THE STATUS data. The status is defined below. If the status byte indicates an error, the **Request Sense** command can be used to determine the nature of the error (See Request Sense command).

```
Byte     Bit Meaning
0      | C C C 0 0 0 0 0 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 0 0 |
2      | 0 0 0 0 0 0 0 0 |
3      | 0 0 0 0 0 0 0 0 |
4      | 0 0 0 0 0 0 0 0 |
5      | 0 0 0 0 0 0 0 0 |
```

Status Byte

The status byte returned from the unit indicates whether the last CD-ROM operation was successful. This byte also returns information on the state of the unit. The Media Change bit is set to one (1) by default or when the open/ close switch on the unit has been used. The bit will remain one until AFTER the first READ, EXTENDED READ, or READ TOC has been performed (i.e., until the first read this bit will be set; then the bit will be cleared, until the disc table is opened indicating media changed).

Format: | 0 0 0 0 M B E 0 |

Status Codes:		
	M	Media Change Bit
	B	Busy Device Not Ready
	E	Error

Request Sense (0x03)

Returns the sense code from the target device. The sense data returns the error that has occurred. In order to confirm an error, this command must be used directly after the occurrence of the error condition.

Byte	**Bit Meaning**	
0	C C C 0 0 0 1 1	-> Controller Number (CCC)
1	0 0 0 0 0 0 0 0	
2	0 0 0 0 0 0 0 0	
3	0 0 0 0 0 0 0 0	
4	0 0 0 0 0 0 0 0	
5	0 0 0 0 0 0 0 0	

The Sense Data Block is sent to the host unit whenever a Request Sense command is executed. The block returned from this call is fixed to 32 bytes. The sense key nibble in byte number 2 is the error condition of the drive. The bytes 16 to 31 contain the current Sub-Q information from the disk.

Sense Data Block

Byte	**Bit Meaning**	
0	1 1 1 1 0 0 0 0	-> Fixed (0xF0)
1	0 0 0 0 0 0 1 1	-> Fixed (0x03)
2	0 0 0 0 K K K K	-> Sense Key (KKKK)
3	0 0 0 0 0 0 0 0	-> Fixed
4	0 0 0 0 0 0 0 0	-> Fixed
5	0 0 0 0 0 0 0 0	-> Fixed
6	0 0 0 0 0 0 0 0	-> Fixed
7	0 0 0 1 1 0 0 0	-> Fixed (0x18)
8	0 0 0 0 0 0 0 0	-> Future Expansion
	~ ~	
15	0 0 0 0 0 0 0 0	
16	0 0 F F 0 0 0 1	-> Sub-Q Information Flag (FF)
17	T T T T T T T T	-> Track (Song) Number (TTTTTTTT)
18	I I I I I I I I	-> Index (IIIIIIII)
18	M M M M M M M M	-> Relative Minute (MMMMMMMM)
20	S S S S S S SS	-> Relative Second (SSSSSSSS)
21	F F F F F F F F	-> Relative Frame (FFFFFFFF)
22	0 0 0 0 0 0 0 0	-> Fixed
23	M M M M M M M M	-> Absolute Minute (MMMMMMMM)
24	S S S S S S S S	-> Absolute Second (SSSSSSSS)
25	F F F F F F F F	-> Absolute Frame (FFFFFFFF)
26	B B B B B B B B	-> Beginning Song Number (BBBBBBBB)
27	E E E E E E E E	-> Ending Song Number (EEEEEEEE)
28	M M M M M M M M	-> Ending Song Minute (MMMMMMMM)
29	S S S S S S S S	-> Ending Song Second (SSSSSSSS)
30	F F F F F F F F	-> Ending Song Frame (FFFFFFFF)
31	0 0 0 0 0 0 0 0	

Sense Key

The following is a list of the sense key definitions. This is the current error condition returned in byte 2 of the Request Sense call.

0	Normal, no error condition
1	Recovered Error, any hardware recoverable error
2	Not Ready, the CD-ROM is not in a ready condition, i.e., playing audio song
3	Media Error, unable to read media, CRC error
4	Hardware, hardware-related error only, pickup switch, CLV servo
5	Illegal Request, unimplemented function or opcode
6	No Media, no media in the drive or the door is open
7	Wrong media, the wrong media type for the applied command

Sub-Q information is returned in bytes 16 thru 31 of the returned data. All Sub-Q information is in BCD. The meaning of this information is as follows:

Flag	The flag is set according to the type of disc within the unit. It can either be zero (0) for an audio disc or one (1) for an audio emphasis disc or two (2) for a digital data disc.
Track Number	The current track number that is playing. This is also referred to as the song number.
Index	This is used for further expansion of a CD-ROM disc.
Relative Time	The relative time is broken down into minutes, seconds, and frame. It is relative to the beginning of the current track number.
Absolute Time	The absolute time is the current block or sector number of the disc. It is relative to the beginning of the disc.
Beginning Song	The beginning song number on the disc.
Ending Song	The ending song or last song on the disc.
Ending Time	The ending time is the last audio or data block on the disk. It represents the starting time of the disc lead-out area.

Read and Extended Read

This command reads data from the disc starting at the logical address or block number given and sends it to the host. The number of blocks to read is the Block Count. This command automatically seeks the logical address.

Read (0x08)

```
Byte    Bit Meaning
0     | C C C 0 1 0 0 0 |-> Controller Number (CCC)
1     | 0 0 0 H H H H H |-> Block MSB (HHHHH bits 20-16)
2     | M M M M M M M M |-> Block (MMMMMMMM bits 15-8)
3     | L L L L L L L L |-> Block (LLLLLLLL bits 7-0)
4     | C C C C C C C C |-> Block Count (CCCCCCCC 1-255)
5     | 0 0 0 0 0 0 0 0 |
```

Extended Read (0x18)

```
Byte    Bit Meaning
0     | C C C 1 1 0 0 0 |-> Controller Number (CCC)
1     | 0 0 0 0 0 0 0 0 |
2     | B B B B B B B B |-> Must be zero / Min (BCD)
3     | B B B B B B B B |-> Block MSB / Sec (BCD)
4     | B B B B B B B B |-> Block/ Frame (BCD)
5     | B B B B B B B B |-> Block LSB / Must be zero
6     | 0 0 0 0 0 0 0 0 |
7     | C C C C C C C C |-> Block count MSB
8     | C C C C C C C C |-> Block count LSB
9     | 0 T 0 0 0 0 0 0 |-> Time Read bit
```

The Extended Read command allows addressing the CD-ROM in the Red Book (audio) format (minute,second,frame) as well as Yellow Book (data) format. It also allows 24 bits for the Yellow Book block address instead of 21 bits in the regular read command. For the Extended Read command, a bit in byte 9 is used to indicate which type of Extended Read is to be done. When the T bit is zero (0) then we are reading the CD disc via Yellow Book format. Setting this bit to one (1) indicates this read is using Red Book format.

Seek and Extended Seek

The Seek command seeks to a logical address. NOTE: After seeking, the CD is in a PAUSE condition. Red as well as Yellow Book format can be used to seek to the desired position using the Extended Seek command. (See Extended Read command for more information.)

Seek (0x0B)

```
Byte     Bit Meaning
0      | C C C 0 1 0 1 1 |-> Controller Number (CCC)
1      | 0 0 0 H H H H H |-> Block MSB (HHHHH bits 20-16)
2      | M M M M M M M M |-> Block (MMMMMMMM bits 15-8)
3      | L L L L L L L L |-> Block (LLLLLLLL bits 7-0)
4      | 0 0 0 0 0 0 0 0 |
5      | 0 0 0 0 0 0 0 0 |
```

Extended Seek (0x1B)

```
Byte     Bit Meaning
0      | C C C 1 1 0 1 1 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 0 0 |
2      | B B B B B B B B |-> Must be zero / Min (BCD)
3      | B B B B B B B B |-> Block MSB/ Sec (BCD)
4      | B B B B B B B B |-> Block / Frame (BCD)
5      | B B B B B B B B |-> Block LSB / All zero
6      | 0 0 0 0 0 0 0 0 |
7      | 0 0 0 0 0 0 0 0 |
8      | 0 0 0 0 0 0 0 0 |
9      | 0 T 0 0 0 0 0 0 |-> Time read bit
```

Audio Stop (0x05)

This command cancels the Audio Program or Audio Start command. All data sent to the unit via these commands is lost.

```
Byte     Bit Meaning
0      | C C C 0 0 1 0 1 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 0 0 |
2      | 0 0 0 0 0 0 0 0 |
3      | 0 0 0 0 0 0 0 0 |
4      | 0 0 0 0 0 0 0 0 |
5      | 0 0 0 0 0 0 0 0 |
```

Audio Program (0x11)

This command sets the starting and duration times for audio play. With the repeat flag set, the sequence is performed again. The Test Unit Ready command is used to determine whether the audio is finished or not.

```
Byte    Bit Meaning
0      | C C C 1 0 0 0 1 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 L R |-> Left Channel (L)
                          -> Right Channel (R)
2      | M M M M M M M M |-> Start Min (BCD)
3      | S S S S S S S S |-> Second (BCD)
4      | F F F F F F F F |-> Frame (BCD)
5      | M M M M M M M M |-> Finishing Minute (BCD)
6      | S S S S S S S S |-> Second (BCD)
7      | F F F F F F F F |-> Frame (BCD)
8      | 0 0 0 0 0 0 0 0 |
9      | R 0 0 0 0 0 0 0 |-> Repeat flag
```

The right and left channel bits determine whether or not the right or left channel has sound. When the bits are zero, that channel has sound. Of course, a one indicates that that channel has no sound.

Audio Start (0x06)

This command sets the first song and the number of songs to play by the song number. These song number are in decimal.

```
Byte    Bit Meaning
0      | C C C 0 0 1 1 0 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 L R |-> Left Channel (L)
                          -> Right Channel (R)
2      | 0 0 0 0 0 0 0 0 |
3      | S S S S S S S S |-> Song Number
4      | F F F F F F F F |-> Number of Songs
5      | C I 0 0 0 0 0 0 |-> Continue Flag (C)
                          -> Index Flag (I)
```

The left and right channel bits are the same as described under Audio Program. With the Index Flag set, the unit is in 10-second play mode for each song. If this flag is cleared (zero), the unit will be in Play All mode. The Continue Flag is used to tell the CD unit that this command is to be continued and that MORE song numbers will be sent to the unit. Clearing this bit ends the command.

Read TOC (0x19)

This command sends to the host computer 512 bytes of the Table Of Contents (TOC). The TOC data has a fixed size of 512 bytes. The TOC data record is composed of four bytes: the track number (TNO) , followed by the starting minute, second and frame. All records are sequential (i.e., every 4 bytes is the next record). All data returned from the TOC is in BCD form.

WARNING: If this command is used while audio is playing, the audio commands are terminated as if the Audio Stop command were used.

```
Byte    Bit Meaning
0     | C C C 1 1 0 0 1 |-> Controller Number (CCC)
1     | 0 0 0 0 0 0 0 0 |
2     | 0 0 0 0 0 0 0 0 |
3     | 0 0 0 0 0 0 0 0 |
4     | 0 0 0 0 0 0 0 0 |
5     | E 0 0 0 0 0 0 0 |-> Data Edit Flag
```

If the Data Edit flag is cleared (zero), the 512 bytes of TOC data are sent without editing. When this flag is set, the 512 bytes of TOC are sent after editing. (One complete set of data is sent, the remaining bytes are all 0x0.) If there are more than 39 songs on the disk some information will not be returned with the edit bit set. Use the Request Sense command to find the total number of songs on the disc. The format for the TOC is as follows:

```
Byte    Bit Meaning
0     | X X X X X X X X |-> TNO Time Number 1
1     | X X X X X X X X |-> Minute
2     | X X X X X X X X |-> Second
3     | X X X X X X X X |-> Frame
4     | X X X X X X X X |-> TNO Time Number 2
5     | X X X X X X X X |-> Minute
6     | X X X X X X X X |-> Second
7     | X X X X X X X X |-> Frame
            ~     ~
508   | X X X X X X X X |-> TNO Time Number 128
509   | X X X X X X X X |-> Minute
510   | X X X X X X X X |-> Second
511   | X X X X X X X X |-> Frame
```

The Track Number field (the first byte of each record) is defined as:

TNO	Meaning
0x00	The following data has no meaning.
0xA0	The minute determines the number of the first song to play. Second and block are all zero.
0xA1	The minute determines the number of the next song to play. Second and block are all zero.
0xA2	The minute and second show the finishing time of the last song. (Total song time on the disk.)
(0<TNO<0xA0)	The data is valid and represents the starting time of the song for that song number.

Mode Sense (0x1A)

The current setting of the CD-ROM is returned to the host computer. This includes the number of bytes/block and the current data mode type (See Mode Select).

Byte	Bit Meaning	
0	C C C 1 1 0 1 0	-> Controller Number (CCC)
1	0 0 0 0 0 0 0 0	
2	0 0 0 0 0 0 0 0	
3	0 0 0 0 0 0 0 0	
4	0 0 0 0 0 0 0 0	
5	0 0 0 0 0 0 0 0	

The Mode Sense data is returned in a 16-byte packet. This packet is defined as:

Byte	Bit Meaning	
0	0 0 0 0 1 1 1 1	-> Fixed to 0x0F
1	1 0 0 0 0 0 1 1	-> Fixed to 0x83
2	1 0 0 0 0 0 0 0	-> Fixed to 0x80
3	0 0 0 0 0 0 0 0	
4	0 0 0 0 0 0 0 0	
	~ ~	
14	0 0 0 0 0 0 0 0	
15	X X X X X X X X	-> Selected Mode Value

Mode Select (0x15)

This command is used to determine the number of bytes/block as well as the current data mode (Red or Yellow book). The default value is 0x41. It is not necessary to use this command when the default value is used. The information returned is broken into the MS and LS nibbles. The least significant nibble (0-4) determine the data type. The number of bytes per block is determined from the most significant nibble (5-11) in the table below.

Byte	Bit Meaning	
0	C C C 1 0 1 0 1	-> Controller Number (CCC)
1	0 0 0 0 0 0 0 0	
2	0 0 0 0 0 0 0 0	
3	0 0 0 0 0 0 0 0	
4	X X X X X X X X	-> Mode Select Data*
5	0 0 0 0 0 0 0 0	

*

LS	Nibble	Meaning	MS	Nibble	Meaning
0	X0	Normal (2052/2340) bytes	5	0X	512 bytes/block
1	X1	Character 2048 bytes	6	2X	256 bytes/block
2	X2	Bit 2336 bytes	7	3X	1024 bytes/block
3	X3	Data 1 2048 bytes	8	4X	2048 bytes/block
4	X4	Data 2 2336 bytes	9	5X	2052 bytes/block
			10	6X	2336 bytes/block
			11	7X	2340 bytes/block

Inquire (0x12)

This command allows an application to interrogate the DMA channel of the ST in order to find out what devices are on the DMA channel. This command can also be used to find out the current status of the disk table. Byte 1 of the returned information will be a 0x80 if the disk table can be opened using the Open/Close button or 0x00 if the disk table is locked.

Byte	**Bit Meaning**	
0	C C C 1 0 0 1 0	-> Controller Number (CCC)
1	0 0 0 0 0 0 0 0	
2	0 0 0 0 0 0 0 0	
3	0 0 0 0 0 0 0 0	
4	X X X X X X X X	-> Send data flag*
5	0 0 0 0 0 0 0 0	

NOTE: When comparing the information returned by this call with the information below, it is important to use only the string "CD-ROM". The number ":1:" may change with a future release.

* When data length is zero then no data is sent, else 16 bytes of information is sent.

The response from the Inquire command is as follows:

Byte	**Bit Meaning**	
0	0 0 0 0 0 1 0 1	-> Fixed (Read Only)
1	X X X X X X X X	-> If (0x80) Allow media removal
		-> If (0x00) Prevent media removal
2	0 0 0 0 0 0 0 1	-> ACSI Version
3	0 0 0 0 0 0 0 0	
4	0 0 0 0 1 0 1 1	-> Fixed (0x0B)
5	0 1 0 0 0 0 1 1	-> "C"
6	0 1 0 0 0 1 0 0	-> "D"
7	0 0 1 0 1 1 0 1	-> "-"
8	0 1 0 1 0 0 1 0	-> "R"
9	0 1 0 0 1 1 1 1	-> "O"
A	0 1 0 0 1 1 0 1	-> "M"
B	0 0 1 0 0 0 0 0	-> " "
C	0 0 1 1 1 0 1 0	-> ":"
D	0 0 0 0 0 0 0 0	-> "1"
E	0 0 1 1 1 0 1 0	-> ":"
F	0 0 1 0 0 0 0 0	-> " "

Media Removal (0x1E)

This command prevents the removal of the CD disc until such time as the computer signals the CD that the media can now be removed. If the Operation Flag is set to one, then the door cannot be opened. Clearing this bit allows the door to be opened. The default condition will be to "Prevent Removal". You can use the Inquire function to interrogate the CD-ROM as to the current removal status. (See Inquire function).

```
Byte    Bit Meaning
0      | C C C 1 1 1 1 0 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 0 0 |
2      | 0 0 0 0 0 0 0 0 |
3      | 0 0 0 0 0 0 0 0 |
4      | 0 0 0 0 0 0 0 F |-> Operation Flag
5      | 0 0 0 0 0 0 0 0 |
```

Disk Spin-down Timer (0x13)

After about 2 minutes the disc of the CDAR504 will spin down if the unit has not been accessed within this time period. This will prevent wear on the spindle motor. The timer can be set to up to 255 seconds or turned off altogether using this command. When the operation flag is zero (0), the timer is turned off and the disc will spin forever. When the operation flag is one (1), byte 3 of the command block indicates the time in seconds before the spindle motor will spin down.

```
Byte    Bit Meaning
0      | C C C 1 0 0 1 1 |-> Controller Number (CCC)
1      | 0 0 0 0 0 0 0 0 |
2      | 0 0 0 0 0 0 0 0 |
3      | T T T T T T T T |-> Time (TTTTTTTT)  in seconds
4      | 0 0 0 0 0 0 0 F |-> Operation Flag (F)
5      | 0 0 0 0 0 0 0 0 |
```

Appendix

APPENDIX A

Example cdbind.h file for programming in the "C" programming language for version (1.0) MetaDOS™ CD-ROM Bios Extensions.

```
/*******************************************************************
*
* cdbind.h    Standard "C" include file for use with cdbind.o to access
*             the CD-ROM Extended Bios.
*
*             Started:      08/01/88
*             Last Update:  08/01/88
*
*******************************************************************/

/* CD-ROM trap calling routine. */
extern long   CDROM();

/* Currently defined Extended Bios calls: */
#define Meta_init(a)         CDROM(48,a);
#define cd_open(a,b)         CDROM(49,a,b);
#define cd_close(a)          CDROM(50,a);
#define cd_read(a,b,c,d)     CDROM(51,a,b,c,d);
#define cd_seek(a,b)         CDROM(53,a,b);
#define get_status(a,b)      CDROM(54,a,b);

#define start_aud(a,b,c)     CDROM(59,a,b,c);
#define stop_aud(a)          CDROM(60,a);
#define set_songtime(a,b,c)  CDROM(61,a,b,c);
#define get_toc(a,b,c)       CDROM(62,a,b,c);
#define disc_info(a,b)       CDROM(63,a,b);
```

Source to cdbind.o trap call.

```
; *************************************************************************
; cdbind.o    Trap #14 bios call.
;
;             Started:      08/01/88
;             Last Update:  08/01/88
; *************************************************************************
;
; CD-ROM trap calling routine.
;
       .globl _CDROM;

       .text
       _CDROM:
              move.l (sp)+,savsp
              trap   #14
              move.l savsp,-(sp)
              rts

       .bss:
       savsp: ds.l   1
       .end
```

APPENDIX B

Quick Reference to the Extended Bios commands. All functions return a long value except the bios_init call. See function description.

```
Meta_init((long) buffer);

open( (int) phydrv, (long) buffer);

close( (int) phydrv);

read( (int) phydrv, (long) buffer, (long) blockno, (int) numblks);

write( (int) phydrv, (long) buffer, (long) blockno, (int) numblks);

seek( (int) phydrv, (long) blockno);

status((int) phydrv, (long) buffer);

start_aud( (int) phydrv, (int) flag, (long)byte_array);

stop_aud( (int) phydrv);

set_songtime( (int) phydrv, (int) flag, (long) start_time, (long)
      end_time);

get_toc((int) phydrv, (int) flag, (long) buffer);

disc_info((int) phydrv, (long) buffer);
```