Microsoft MS-DOS CD-ROM Extensions 2.1

Product Overview Version 2.10 Beta

The Microsoft MS-DOS CD-ROM Extensions are an extension to the MS-DOS operating system which permit reading CD-ROM disks which conform to both the High Sierra May 28th format and the ISO-9660 version of the High Sierra format. The CD-ROM disc appears just like a magnetic disk to the user and to applications software, ensuring compatibility with current software.
Microsoft, as creator of the MS-DOS operating system, is able to ensure compatibility with MS-DOS.

Product Components

The complete product consists of a program supplied by Microsoft and of a hardware-dependent device driver supplied by an OEM customer. The program supplied by Microsoft is named MSCDEX.EXE. Technical documentation as well as a sample driver are also supplied by Microsoft.

Technical Overview

Characteristics

ţ Requires MS-DOS 3.1 or higher or 4.0 (or PC-DOS 3.1 or higher or 4.0)
ţ Implements the High Sierra May 28th format and ISO-9660
ţ Requires a hardware-dependent device driver

This product uses the Microsoft Networks interface to MS-DOS so it requires MS-DOS version 3.1 or higher or 4.0. MS-DOS 3.1 virtualizes the interface to drives. The entire CD-ROM (potentially all 660 megabytes) will appear to applications as a single MS-DOS drive letter. The extensions overcome the 32 megabyte disk size limitation in MS-DOS. The Microsoft MS-DOS CD-ROM Extensions provide a high degree of compatibility with applications that depend on MS-DOS standard interfaces. Applications can access files on the CD-ROM just as they would on any disk.

The program MSCDEX.EXE is an installable file system driver implemented as a terminate and stay resident module. The user will load this program upon booting the computer using AUTOEXEC.BAT. The hardware-dependent device driver implements basic functions to read the CD-ROM disc and is loaded with the MS-DOS CONFIG.SYS file.

The Microsoft MS-DOS CD-ROM Extensions implement both the May 28th High Sierra file format and the ISO-9660 version of that standard. All features defined in May 28th proposal for Level 1 are implemented. In addition the following are implemented:

Features Beyond High Sierra Level 1

ţ Support for Interleaved Files
ţ Support for 31 Character File Names when possible through truncation
ţ Support for Hidden Files
ţ Support for Access to VTOC
ţ Ignores Higher Level Files and Functions when present on the disk:
   - Associated Files
   - Protection Bits
   - Record Bits
   - File Version Numbers

ţ Support for shift-JIS Kanji (Japanese character) filenames

Hardware-Dependent Device Driver

This product requires a hardware-dependent device driver that interfaces to a specific OEM drive or drives. A detailed specification for the device driver as well as a sample driver are included. The driver implements the basic functions of reading the CD-ROM and is installed using the DOS CONFIG.SYS conventions. A minimum set of functions that allow reading the CD-ROM disc are required to be in the device driver; there are optional additional functions which provide increased performance when the CD-ROM drive and controller can provide additional functions and these are implemented in the driver. These functions are detailed in the Microsoft MS-DOS CD-ROM Extensions Hardware-Dependent Device Driver Specification.

The device driver is written by the OEM customer for the MS-DOS CD-ROM Extensions. Development of the device driver is estimated to take approximately 1-3 man-months. This estimate assumes an engineer experienced in 8086 assembler programming and familiar with MS-DOS and the CD-ROM drive hardware. If a previous device driver has already been written, less time will probably be needed to implement the driver for the Microsoft MS-DOS CD-ROM extensions. There are third party companies who will write the hardware-dependent device drivers on a consulting basis.

Licensing the Microsoft MS-DOS CD-ROM Extensions

Microsoft will license the MS-DOS CD-ROM Extensions to manufacturers and marketers of CD-ROM disc drives. The license agreement will allow the use of the product on a personal computer to which a licensed disc drive is attached. Developers of CD-ROM disks will not need to acquire any license or pay any royalty in order to develop or sell CD-ROM discs, and will not be entitled to distribute the MS-DOS CD-ROM Extensions. The end user will purchase the driver from drive manufacturer or marketer, not the CD-ROM disc developer.

The Microsoft MS-DOS CD-ROM Extensions will be delivered to licensees on a 5 1/4" MS-DOS diskette. Licensees are expected to distribute the Extensions to their customers on a floppy diskette containing both MSCDEX.EXE and the hardware-dependent device driver written by the licensee. The floppy would be included in the package containing the CD-ROM drive.

Creating CD-ROM Disks in the High Sierra Format

The Microsoft MS-DOS CD-ROM Extensions provides for reading CD-ROM discs in the High Sierra/ISO-9660 format on MS-DOS computers. They do not create CD-ROM disks in the High Sierra/ISO-9660 format. Microsoft does not manufacture CD-ROM discs, nor provide pre-mastering services. Third party companies can create CD-ROM discs in the High Sierra/ISO-9660 format and provide other pre-mastering services. Microsoft can supply a list of companies providing or planning to provide these services upon request.

Software developers do not need the MS-DOS CD-ROM Extensions in order to create either applications software which reads CD-ROM discs, or to create CD-ROM discs. Once the software is ready and a disc has been pressed, developers will want a copy of the Extensions for testing; however, they are not needed in order to start development.

Software developers need do nothing special for accessing CD-ROM discs; they
issue the same MS-DOS OPEN and READ calls as for opening any magnetic disks.
Programmers can develop CD-ROM applications using standard MS-DOS tools.
They need to be aware that they cannot create any temporary files or write
any files in either the directory or on the entire CD-ROM disc. Software
developers will want to minimize the number of seeks to the CD-ROM because
of the comparatively long seek times of CD-ROM drives.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

Hardware-Dependent Device Driver Specification

Intent

This document (Document Number: 000080010-100-O00-1186) describes the CD-ROM
hardware-dependent device driver and its interface with MSCDEX.EXE, the MS-
DOS CD-ROM Extensions resident program. Differences between CD-ROM drives
and hard- or floppy-disk drives account for the differences in this device
driver specification from the normal MS-DOS block and character device
driver specification. The chapters on device drivers in the MS-DOS
Programmer's Reference Manual (MS-PRM) provide more information.

The MS-DOS operating system reads CONFIG.SYS and installs the device.
MSCDEX.EXE performs an open system call on the device driver name in order
to communicate with it and uses an IOCTL call to ask the device driver for
the address of its device header. From the device header address, MSCDEX.EXE
locates the device driver's interrupt and strategy routines. After that, all
requests the device driver receives come directly from MSCDEX.EXE, not MS-
DOS. To avoid reentrancy problems and allow MSCDEX to monitor all media
changes, all other applications that wish to communicate directly with CD-
ROM device drivers should do so through the Send Device Driver Request INT
2Fh function 10h. MSCDEX.EXE interfaces with MS-DOS so that normal requests
for I/O with files on a CD-ROM drive down to the MS-DOS INT 21h service
layer will work just as they would for a normal MS-DOS device.

Installation

The device driver will be installed in the same way as any other device with
an entry in CONFIG.SYS. The syntax is:

```
  DEVICE=<filename> /D:<device_name> /N:<number of drives>
```

The following are examples:

```
  DEVICE=HITACHI.SYS /D:MSCD001 /D:MSCD002

  DEVICE=SONY.SYS    /D:MSCD003 /N:2
```

The arguments will be the character device names that will be used on the
command line when starting MSCDEX.EXE so that it can find and communicate
with the device driver.

A device driver may support one or more physical drives or logical disks.
This may be done by having multiple device headers in the device driver file
(in which case it will be necessary to have more than one device_name on the
command line - one for each device header; see the HITACHI.SYS example

above) or through the use of subunits. Each disk handled by a device driver that supports multiple disks using subunits is addressed by the subunit field of the request header when a request is made for that disk. A device driver that supports more than one disk can share code and data instead of requiring separate device drivers for each disk. A "jukebox" CD-ROM system would be an example of a CD-ROM device that might wish to support more than one drive or a disk pack using a single device driver.

Device drivers that use multiple subunits should use the optional switch /n:<number of drives> to say how many drives are present. If not present, the default number of drives is 1. If the driver can tell how many drives are installed without a command line switch, then this argument is not necessary. Unless there are special considerations, it is better practice to support multiple drives using subunits than to have multiple device headers in the same device driver file.

Device Header

The device header is an extension to what is described in the MS-PRM.

```
DevHdr    DD   -1        ; Ptr to next driver in file or -1 if last driver
          DW   ?         ; Device attributes
          DW   ?         ; Device strategy entry point
          DW   ?         ; Device interrupt entry point
          DB   8 dup (?) ; Character device name field
          DW   0         ; Reserved
          DB   0         ; Drive letter
          DB   ?         ; Number of units
```

The following are the device attributes for MSCDEX.EXE device drivers:

```
  Bit 15       1     - Character device
  Bit 14       1     - IOCTL supported
  Bit 13       0     - Output 'till  busy
  Bit 12       0     - Reserved
  Bit 11       1     - OPEN/CLOSE/RM supported
  Bit 10-4     0     - Reserved
  Bit  3       0     - Dev is CLOCK
  Bit  2       0     - Dev is NUL
  Bit  1       0     - Dev is STO
  Bit  0       0     - Dev is STI
```

MSCDEX.EXE device drivers will be character devices that understand IOCTL calls and handle OPEN/CLOSE/RM calls.

The drive letter field is a read-only field for the device driver and is initialized to 0. The field is for MSCDEX.EXE to use when it assigns the device driver to a drive letter (A = 1, B = 2...Z = 26). It should never be modified by the device driver. For drivers that support more than one unit, the drive letter will indicate the first unit, and each successive unit is assigned the next higher drive letter. For example, if the device driver has four units defined (0-3), it requires four drive letters. The position of the driver in the list of all drivers determines which units correspond to which drive letters. If driver ALPHA is the first driver in the device list, and it defines 4 units (0-3), they will be A, B, C, and D. If BETA is the second driver and defines three units (0-2), they will be E, F, and G, and so on. The theoretical limit to the number of drive letters is 63, but it should be noted that the device installation code will not allow the installation of a device if it would result in a drive letter > 'Z' (5Ah).

All block device drivers present in the standard resident BIOS will be
placed ahead of installable device drivers in the list.

ÄÄÄ
ÄÄÄÄÄÄÄÄ
NOTE:
  It is important that one set lastdrive=<letter> in CONFIG.SYS
  to accommodate the additional drive letters that CD-ROM device drivers
  will require.
ÄÄÄ
ÄÄÄÄÄÄÄÄ

The number-of-units field is set by the device driver to the number of disks
that are supported. Normal character devices do not support more than one
unit and MS-DOS does not expect a character device to handle more than one
unit or have a nonzero subunit value in the request header. Since these
device drivers are not called by MS-DOS directly, this is not a problem.
Nonetheless, the number of units returned by the device driver in the
number-of-units field during the INIT call must be 0, since MS-DOS makes the
INIT call and does not expect a nonzero value for a character device.
MSCDEX.EXE will never see what is returned anyway, and relies on the number-of-units field in the device header.

Sample device header:

```
HsgDrv    DD   -1            ; Pointer to next device
          DW   0c800h        ; Device attributes
          DW   STRATEGY      ; Pointer to device strategy routine
          DW   DEVINT        ; Pointer to device interrupt routine
          DB   'HSG-CD1 '    ; 8-byte character device name field
          DW   0             ; Reserved (must be zero)
          DB   0             ; Drive letter (must be zero)
          DB   1             ; Number of units supported (one or more)
```

As with other MS-DOS device drivers, the code originates at offset 0, not
100H. The first device header will be at offset 0 of the code segment. The
pointer to the next driver is a double word field (offset/segment) that is
the address of the next device driver in the list, or -1 if the device
header is the only one or the last in the list. The strategy and interrupt
entry points are word fields and must be offsets into the same segment as
the device header. The device driver is expected to overwrite the name(s) in
each of its one or more device headers with the <device_name> command line
arguments during its initialization.

MSCDEX.EXE will call the device driver in the following manner:

1. MSCDEX.EXE makes a far call to the strategy entry.

2. MSCDEX.EXE passes device driver information in a request header to the
   strategy routine.

3. MSCDEX.EXE makes a far call to the interrupt entry.

Request header

MSCDEX.EXE will call the device's strategy routine with the address of a
request header in ES:BX. The format of the request header is the same as
what is described in the MS-PRM.

```
ReqHdr    DB   ?        ; Length in bytes of request header
          DB   ?        ; Subunit code for minor devices
          DB   ?        ; Command code field
          DW   ?        ; Status
          DB   8 dup (?) ; Reserved
```

Status

The status word also has the same format as described in the MS-PRM. It is 0 on entry and is set by the device driver.

```
  Bit 15        - Error bit
  Bit 14-10     - Reserved
  Bit  9        - Busy
  Bit  8        - Done
  Bit  7-0      - Error code (bit 15 on)
```

Bit 15, the error bit, is set by the device driver if an error is detected or if an invalid request is made to the driver. The low 8 bits indicate the error code.

Bit 9, the busy bit, should be set by the device driver when the drive is in audio play mode. Device drivers should fail all requests to the physical device that require head movement when the device is playing and return the request with this bit and the error bit set and an error code. Requests that would not interrupt audio play may return without error but will also have this bit set when the drive is in audio play mode. Play mode can be terminated prematurely with a reset or STOP AUDIO request and a new request can be made at that point. Monitoring this bit in each successive request, an Audio Q-Channel Info IOCTL for example, will tell when play mode is complete.

Bit 8, the done bit, is set by the device driver when the operation is finished.

Error codes are the following:

0	Write-protect violation
1	Unknown unit
2	Drive not ready
3	Unknown command
4	CRC error
5	Bad drive request structure length
6	Seek error
7	Unknown media
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure
D	Reserved
E	Reserved
F	Invalid disk change

Command Code Field

The following values are valid command codes:

0	INIT
1	MEDIA CHECK (block devices)

2	BUILD BPB (block devices)	
3	IOCTL INPUT	
4	INPUT (read)	
5	NONDESTRUCTIVE INPUT NO WAIT	
6	INPUT STATUS	
7	INPUT FLUSH	
8	OUTPUT (write)	
9	OUTPUT WITH VERIFY	
10	OUTPUT STATUS	
11	OUTPUT FLUSH	
12	IOCTL OUTPUT	
13	DEVICE OPEN	
14	DEVICE CLOSE	
15	REMOVABLE MEDIA (block devices)	
16	OUTPUT UNTIL BUSY	
128	READ LONG	(NEW)
129	Reserved	
130	READ LONG PREFETCH	(NEW)
131	SEEK	(NEW)
132	PLAY AUDIO	(NEW)
133	STOP AUDIO	(NEW)
134	WRITE LONG	(NEW)
135	WRITE LONG VERIFY	(NEW)
136	RESUME AUDIO	(NEW)

Unsupported or illegal commands will set the error bit and return the error code for Unknown Command. This includes command codes 1, 2, 4, 5, 6, 8, 9, 10, 15, 16, and 129; and 11, 134 and 135 for systems that do not support writing.

If, in the time since the last request to that device driver unit, the media has changed, the device driver will return the error code for invalid disk change and set the error bit. MSCDEX.EXE will then decide whether to retry the request or abort it.

The minimal CD-ROM device driver will read cooked Mode 1 data sectors using HSG addressing mode and return appropriate values for the IOCTL calls. Most other features enhance performance or add useful capabilities.

INIT

Command code = 0
ES:BX = INIT

```
INIT     DB   13 dup (0); Request header
         DB   0         ; Number of units (must be 0)
         DD   ?         ; End address
         DD   ?         ; Ptr to BPB array
         DB   0         ; Block device number
```

This call is made only once, when the device is installed. INIT and a single IOCTL call for the device header address are the only device driver calls that come directly from MS-DOS. Because the INIT function is called from MS-DOS, the number of units returned is 0, as for normal MS-DOS character devices. MSCDEX.EXE will get the number of units supported from the device header.

The device must return the END ADDRESS, which is a DWORD pointer to the end of the portion of the device driver to remain resident. Code and data following the pointer is used for initialization and then discarded. If there are multiple device drivers in a single file, the ending address returned by the last INIT call will be the one that MS-DOS uses, but it is recommended that all the device drivers in the file return the same address. The code to remain resident for all the devices in a single file should be grouped together low in memory with the initialization code for all devices following it in memory.

The pointer to BPB array points to the character after the "=" on the line in CONFIG.SYS that caused this device driver to be loaded. This data is read-only and allows the device driver to scan the invocation line for parameters. This line is terminated by a carriage return or a line feed. During initialization, the device driver must set the device name field in the device header to the argument provided on the invocation line in CONFIG.SYS. The device driver must also check that the device_name command line argument is a legal 8-character filename and pad it out to 8 characters with spaces (20H) when copying it to the device name field.

The block device number and number of units are both 0 for character devices.

READ (IOCTL Input)

Command code = 3
ES:BX = IOCTLI

```
IOCTLI    DB   13 dup (0); Request header
          DB   0        ; Media descriptor byte from BPB
          DD   ?        ; Transfer address
          DW   ?        ; Number of bytes to transfer
          DW   0        ; Starting sector number
          DD   0        ; DWORD ptr to requested vol ID if error 0FH
```

The media descriptor byte, starting sector number, and volume ID fields are all 0.

The transfer address points to a control block that is used to communicate with the device driver. The first byte of the control block determines the request that is being made. If the command code is reserved or the function not supported, then the device driver will return the error code for Unknown Command. If, for some reason, the device driver is not able to process the request at that time, it will return the error code for Drive Not Ready.

Code	Number of Bytes to Transfer	Function
0	5	Return Address of Device Header
1	6	Location of Head
2	?	Reserved
3	?	Error Statistics
4	9	Audio Channel Info
5	130	Read Drive Bytes
6	5	Device Status
7	4	Return Sector Size
8	5	Return Volume Size
9	2	Media Changed
10	7	Audio Disk Info
11	7	Audio Track Info
12	11	Audio Q-Channel Info

```
 13          13          Audio Sub-Channel Info
 14          11          UPC Code
 15          11          Audio Status Info
 16-255       ?            Reserved
```

Return Address of Device Header

```
Raddr     DB   0         ; Control block code
          DD   ?         ; Address of device header
```

The device driver will fill the 4-byte field with the address of its device header. This is used by MSCDEX.EXE to locate the device driver's strategy and interrupt routines.

Location of Head

```
LocHead   DB   1         ; Control block code
          DB   ?         ; Addressing mode
          DD   ?         ; Location of drive head
```

The device driver will return a 4-byte address that indicates where the head is located. The value will be interpreted based on the addressing mode. (See function READ LONG for more information about addressing modes.)

ÄÄÄ
ÄÄÄÄÄÄÄÄ
NOTE:
  The drive could provide this information by monitoring the Q-channel on
  the disk.
ÄÄÄ
ÄÄÄÄÄÄÄÄ

Error Statistics

```
ErrStat   DB   3         ; Control block code
          DB   N dup (?) ; Error statistics
```

The format of the Error Statistics is not yet defined.

Audio Channel Info

```
AudInfo   DB   4       ; Control block code
          DB   ?       ; Input  channel (0, 1, 2, or 3) for output channel 0
          DB   ?       ; Volume control (0 - 0xff) for output channel 0
          DB   ?       ; Input  channel (0, 1, 2, or 3) for output channel 1
          DB   ?       ; Volume control (0 - 0xff) for output channel 1
          DB   ?       ; Input  channel (0, 1, 2, or 3) for output channel 2
          DB   ?       ; Volume control (0 - 0xff) for output channel 2
          DB   ?       ; Input  channel (0, 1, 2, or 3) for output channel 3
          DB   ?       ; Volume control (0 - 0xff) for output channel 3
```

This function returns the present settings of the audio channel control set with the Audio Channel Control Ioctl Write function. The default settings for the audio channel control are for each input channel to be assigned to its corresponding output channel (0 to 0, 1 to 1, etc.) and for the volume control on each channel is set at 0xff.

Read Drive Bytes

```
DrvBytes  DB   5         ; Control block code
```

```
        DB   ?          ; Number bytes read
        DB   128 dup (?); Read buffer
```

Data returned from the CD-ROM drive itself can be read using this function. The number-bytes-read field returns the length of the number of bytes read, which will not exceed 128 per call. If more than this needs to be returned, the call will be repeated until the number returned is 0.

The function and content of these bytes are entirely device and device driver dependent. This function is provided to allow access to device-specific features that are not addressed under any other portion of the device driver spec.

Device Status

```
DevStat   DB   6        ; Control block code
          DD   ?        ; Device parameters
```

The device driver will return a 32-bit value. Bit 0 is the least significant bit. The bits are interpreted as follows:

Bit 0	0	Door closed
	1	Door open
Bit 1	0	Door locked
	1	Door unlocked
Bit 2	0	Supports only cooked reading
	1	Supports cooked and raw reading
Bit 3	0	Read only
	1	Read/write
Bit 4	0	Data read only
	1	Data read and plays audio/video tracks
Bit 5	0	No interleaving
	1	Supports interleaving
Bit 6	0	Reserved
Bit 7	0	No prefetching
	1	Supports prefetching requests
Bit 8	0	No audio channel manipulation
	1	Supports audio channel manipulation
Bit 9	0	Supports HSG addressing mode
	1	Supports HSG and Red Book addressing modes
Bit 10-31	0	Reserved (all 0)

Return Sector Size

```
SectSize  DB   7        ; Control block code
          DB   ?        ; Read mode
          DW   ?        ; Sector size
```

The device driver will return the sector size of the device given the read mode provided. In the case of CD-ROM, the value returned for cooked is 2048, and the return value for raw is 2352.

Return Volume Size

```
VolSize   DB   8         ; Control block code
          DD   ?         ; Volume size
```

The device driver will return the number of sectors on the device. The size returned is the address of the lead-out track in the TOC converted to a binary value according to FRAME + (SEC * 75) + (MIN * 60 * 75). A disc with a lead out track starting at 31:14.63 would return a volume size of 140613. The address of the lead-out track is assumed to point to the first sector following the last addressable sector recorded on the disc.

Media Changed

```
MedChng   DB   9         ; Control block code
          DB   ?         ; Media byte
```

The normal media check function (command code 1) is not performed on character devices and contains additional semantics that are not needed for CD-ROM device drivers. This is why there is an IOCTL request for this function.

When the device driver receives a call to see if the media has changed on that subunit, it will return one of the following values:

```
   1         Media not changed
   0         Don't know if changed
  -1 (0FFh)  Media changed
```

If the driver can assure that the media has not been changed (through a door-lock or other interlock mechanism), performance is enhanced because MSCDEX.EXE does not need to reread the VTOC and invalidate in-memory buffers for each directory access. For drives that do not report if the media has changed, CD-ROM device drivers can utilize the same solution that has been applied to floppy disks. In some floppy-disk device drivers, if the MEDIA CHECK occurs within 2 seconds of a floppy-disk access, the driver reports "Media not changed." It is highly recommended though that drives be able to detect and report media changes.

If the drive can enforce a door lock mechanism so that the device driver is notified when the door lock has been unlocked or the device driver is requested to do so by MSCDEX.EXE, then to improve performance, the driver could return that the media has not changed without bothering to communicate with the physical device.

If the media has not been changed, MSCDEX.EXE will proceed with the disk access. If the value returned is "Don't know," or "Media changed," then MSCDEX.EXE will check to see if the disk has changed. It will continue if it has not, and reinitialize what it knows about the disk if it has.

It is not necessary for the device driver to do anything for the volume ID when the media has changed.

Audio Disk Info

```
DiskInfo  DB   10        ; Control block code
          DB   ?         ; Lowest track number
          DB   ?         ; Highest track number
```

```
        DD   ?          ; Starting point of the lead-out track
```

This function returns TOC (Table of Contents) information from the Q-Channel
in the lead-in track indicating what the first and last track numbers are
and the Red Book address for the lead-out track (PMIN/PSEC/PFRAME when POINT
= A2). The first and last track numbers are binary values and not BCD. It is
recommended that the information for Audio Disk Info and Audio Track Info
should be read by the drive when the disc is initialized and made accessible
to the driver so that when these functions are called, the drive or driver
do not have to interrupt audio play to read them from the TOC. If the TOC is
not made available to the driver and the driver must obtain the information
itself from the lead-in track, the driver should read and and attempt to
cache the disk and track information during the Audio Disk Info command and
invalidate this information only if the media changes.

Audio Track Info

```
TnoInfo   DB   11       ; Control block code
        DB   ?          ; Track number
        DD   ?          ; Starting point of the track
        DB   ?          ; Track control information
```

This function takes a binary track number, from within the range specified
by the lowest and highest track number given by the Audio Disk Info command,
and returns the Red Book address for the starting point of the track and the
track control information for that track. The track control information byte
corresponds to the byte in the TOC in the lead-in track containing the two
4-bit fields for CONTROL and ADR in the entry for that track. The CONTROL
information is in the most significant 4 bits and the ADR information is in
the lower 4 bits. The track control information is encoded as follows:

```
  00x00000  - 2 audio channels without pre-emphasis
  00x10000  - 2 audio channels with pre-emphasis
  10x00000  - 4 audio channels without pre-emphasis
  10x10000  - 4 audio channels with pre-emphasis
  01x00000  - data track
  01x10000  - reserved
  11xx0000  - reserved
  xx0x0000  - digital copy prohibited
  xx1x0000  - digital copy permitted
```

Audio Q-Channel Info

```
QInfo     DB   12       ; Control block code
        DB   ?          ; CONTROL and ADR byte
        DB   ?          ; Track number (TNO)
        DB   ?          ; (POINT) or Index (X)
                      ; Running time within a track
        DB   ?          ; (MIN)
        DB   ?          ; (SEC)
        DB   ?          ; (FRAME)
        DB   ?          ; (ZERO)
                      ; Running time on the disk
        DB   ?          ; (AMIN) or (PMIN)
        DB   ?          ; (ASEC) or (PSEC)
        DB   ?          ; (AFRAME) or (PFRAME)
```

This function reads and returns the most up to date Q-channel address
presently available. It should not interrupt the present status of the drive
as one of its intended purposes is to monitor the location of the read head

while playing audio tracks. This function should return valid information even when no audio tracks are being played and the head is stationary. The fields returned correspond to the data that is stored in the Q-channel as described in the Red Book. The values in MIN-SEC-FRAME, AMIN-ASEC-AFRAME and PMIN-PSEC-PFRAME are converted by the driver from BCD to binary so that minutes range from 0 to 59+, seconds from 0 to 59, and frames from 0 to 74. The Control and ADR byte, TNO, and POINT/Index bytes are always passed through as they appear on the disc and are not converted. If the drive returns Q-channel information when ADR is not equal to 1, then when ADR is not equal to 1 all ten bytes of information are passed through unmodified to the caller.

Audio Sub-Channel Info

```
SubChanInfo DB   13       ; Control block code
            DD   ?        ; Starting frame address
            DD   ?        ; Transfer address
            DD   ?        ; Number of sectors to read
```

This function takes a Red Book address for a particular frame (also known as a block or frame) and copies 96 bytes of sub-channel information per frame for all the sectors that are requested sequentially at the transfer address given. Each 96 bytes of information do not include the two sync patterns (S0 and S1) that head the subcoding block but only the the 96 bytes of subcoding symbols each with one bit of information for the eight different channels (P-W) that follow them. P is the MSB, W is the LSB of each byte.

The caller is responsible for making sure that 96 * Number_of_sectors_to_read bytes are available at the transfer address for the device driver to store the results.

Data definition and integrity restrictions for data received with this command are interpreted according to the CD-ROM standard (Red and Yellow Book).

UPC Code

```
UPCCode   DB   14       ; Control block code
          DB   ?        ; CONTROL and ADR byte
          DB   7 dup (?) ; UPC/EAN code
                        ; (last 4 bits are zero; the low-order nibble of
                        ; byte 7)
          DB   ?        ; Zero
          DB   ?        ; Aframe
```

This function returns the UPC/EAN (Universal Product Code - BAR coding) for the disc. This information is stored as a mode-2 (ADR=2) Q-channel entry. The UPC code is 13 successive BCD digits (4 bits each) followed by 12 bits of zero. The last byte is the continuation of FRAME in mode-1 though in the lead-in track (TNO=0) this byte is zero. If the CONTROL/ADR byte is zero or if the 13 digits of UPC code are all zero, then either no catalog number was encoded on the disc or it was missed by the device driver. If the command is not supported, then the driver will return an error code of Unknown Command. If the command is supported but the disc does not have a UPC Code recorded, then the driver will return an error code of Sector not Found.

Audio Status Info

```
AudStat DB   15       ; Control block code
        DW   ?        ; Audio status bits
```

```
                  ; Bit 0 is Audio Paused bit
                  ; Bits 1-15 are reserved
        DD   ?       ; Starting location of last Play or for next Resume
        DD   ?       ; Ending location for last Play or for next Resume
```

The Audio Paused bit and Starting and Ending locations are those referred to in the RESUME command.

WRITE (IOCTL OUTPUT)

Command code = 12
ES:BX = IOCTLO

```
IOCTLO    DB   13 dup (0); Request header
          DB   0       ; Media descriptor byte from BPB
          DD   ?       ; Transfer address
          DW   ?       ; Number of bytes to transfer
          DW   0       ; Starting sector number
          DD   0       ; DWORD ptr to requested vol ID if error 0FH
```

The media descriptor byte, starting sector number, and volume ID fields are all 0.

The transfer address points to a control block that is used to communicate with the device driver. The first byte of the control block determines the request that is being made. The Length of Block is the number of bytes to transfer.

Code	Length of Block	Function
0	1	Eject Disk
1	2	Lock/Unlock Door
2	1	Reset Drive
3	9	Audio Channel Control
4	?	Write Device Control String
5	1	Close Tray
6-255	?	Reserved

Eject Disk

```
Eject        DB   0       ; Control block code
```

The device driver will unlock the drive and eject the CD-ROM disk from the drive unit. The door will report as being open until the user has inserted a disk into the drive unit and closed the door. The status bit for door open can be monitored to determine when a disk has been reinserted.

Lock/Unlock Door

```
LockDoor  DB   1       ; Control block code
          DB   ?       ; Lock function
```

When this function is received, the device driver will ask the CD-ROM drive to unlock or lock the door. If lock function is 0, the device driver will unlock the door. If lock function is 1, it will lock the door.

Reset Drive

```
ResetDrv  DB   2         ; Control block code
```

This function directs the device driver to reset and reinitialize the drive.

Audio Channel Control

```
AudInfo   DB   3     ; Control block code
          DB   ?     ; Input  channel (0, 1, 2, or 3) for output channel 0
          DB   ?     ; Volume control (0 - 0xff) for output channel 0
          DB   ?     ; Input  channel (0, 1, 2, or 3) for output channel 1
          DB   ?     ; Volume control (0 - 0xff) for output channel 1
          DB   ?     ; Input  channel (0, 1, 2, or 3) for output channel 2
          DB   ?     ; Volume control (0 - 0xff) for output channel 2
          DB   ?     ; Input  channel (0, 1, 2, or 3) for output channel 3
          DB   ?     ; Volume control (0 - 0xff) for output channel 3
```

This function is intended to provide playback control of audio information on the disk. It allows input channels on the CD-ROM to be assigned to specific output speaker connections. The purpose of this function is to allow two independent channels to be recordedÄÄin different languages for exampleÄÄand to play back only one of them at a time or to be able to manipulate an audio signal so that the source appears to moveÄÄto make a sound seem to move from left to right for example.

Output channel 0 is the left channel, 1 is right, 2 is left prime, and 3 is right prime. The Red Book specification allows for 4 audio channels. The two "prime" channels (2 and 3) extend stereo to quadrophonic stereo.

An audio volume setting of 0 means off. Drives that don't support 4 output audio channels may ignore output to channels 2 and 3. Assignment of input channels 2 and 3 to output channels 0 and 1 may be treated as though the volume control for that channel is 0.

Drives that do not support variable audio control will treat a setting of 0 as off and 1-0xff as on. Drives that support less than 256 volume settings will do their best to break up the 256 settings among the settings they can support. E.g. if there are 16 settings supported, then the first setting will cover 0x01-0x10, the second 0x11-0x20...the sixteenth 0xf1-0xff. Drives that can't play a single channel in both must play only that one channel and try to suppress the other if possible. Drives that can't swap channels should play the channel that was moved in its normal channel.

Write Device Control String

```
DrvBytes  DB   4         ; Control block code
          DB   N dup (?) ; Write buffer
```

This function is provided to allow programs to talk directly to the CD-ROM drive. All remaining bytes are sent uninterpreted to the drive unit.

The function and content of these bytes are entirely device and device driver dependent. This function is provided to allow access to device-specific features that are not addressed under any other portion of the device driver spec.

Close Tray

```
CloseTray DB   5         ; Control block code
```

This command is the logical complement to the Eject Disk command. This command will instructs drives that can do so to close the door or tray.

READ LONG

Command code = 128
ES:BX = ReadL

```
ReadL     DB   13 dup (0); Request header
          DB   ?        ; Addressing mode
          DD   ?        ; Transfer address
          DW   ?         ; Number of sectors to read
          DD   ?        ; Starting sector number
          DB   ?        ; Data read mode
          DB   ?        ; Interleave size
          DB   ?        ; Interleave skip factor
```

The request block is different from a normal character device READ to accommodate the larger size and different characteristics of CD-ROM devices.

The media descriptor byte, which has no meaning for character devices, is now the addressing mode field. The following values are recognized addressing modes:

0	HSG addressing mode
1	Red Book addressing mode
2-255	Reserved

The default addressing mode is the HSG addressing mode. Long (DWORD) address values are treated as logical block numbers, as defined by the High Sierra proposal. When Red Book addressing mode is on, all disk addresses are interpreted as Minute/Second/Frame addresses, according to the Philips/Sony Red Book standard. Each of these fields is 1 byte. The frame byte is the least significant byte of the address field, the "second" byte the next most significant, the minute byte the next, and the most significant byte of the 4-byte field is unused. These values are represented in binary rather than in BCD format. For example, if we are referencing the sector addressed by minute 36, second 24, frame 12, the hex long value for this would be 0x0024180C. The relationship between High Sierra sectors and Red Book frames is described by the equation:

Sector = Minute * 60 * 75 + Second * 75 + Frame - 150

The byte/sector count field becomes the number of sectors to read and the starting sector number expands from one word to two, which means we can address up to 4 giga-sectors (over 8 terabytes). The DWORD ptr for requested volume ID is eliminated and MSCDEX.EXE will keep track of what volume is needed.

MSCDEX.EXE handles buffering requests, but performance may be improved if the device driver reads ahead or uses a sector caching scheme, given the slow seek times of CD-ROM drives. The operating system will use the prefetch function when it can to give hints to the driver.

The data read mode field will be one of the following:

0	Cooked mode
1	Raw mode

2-255 Reserved

Cooked mode is the default mode in which the hardware typically handles the EDC/ECC and the device driver returns 2048 bytes of data per sector read. When raw mode is set, the driver will return all 2352 bytes of user data, including any EDC/ECC present independent of the actual sector mode (Mode 2 Form 1 vs. Mode 2 Form 2). User programs will have to consider this and allow enough room for buffer space when reading in raw mode as each sector returned will take up 2352 bytes of space. Drives that cannot return all 2352 bytes will return what they can and leave blank what they cannot. For example, drives that can return all 2336 bytes except the 16 byte header will leave a space in the first 16 bytes where the header would go so that the sectors align on 2352 byte boundaries. Drivers should do what they can to return as much of the user data per sector as possible.

The two interleave parameters are for drivers that support interleaved reading. If the driver does not support interleaving, these fields are both ignored. If it does, interleave size is the number of consecutive logical blocks or sectors that are stored sequentially, and the interleave skip factor is the number of consecutive logical blocks or sectors that separate portions of the interleaved file.

READ LONG PREFETCH

Command code = 130
ES:BX = ReadLPre

```
ReadLPre  DB   13 dup (0); Request header
        DB   ?       ; Addressing mode
        DD   0       ; Transfer address
        DW   ?       ; Number of sectors to read
        DD   ?       ; Starting sector number
        DB   ?       ; Read mode
        DB   ?       ; Interleave size
        DB   ?       ; Interleave skip factor
```

This function is similar in form to READ LONG, but control returns immediately to the requesting process. The device driver is not obligated to read in the requested sectors but can instead consider the request for these sectors as hints from the operating system that they are likely to be needed. It is recommended that at a minimum, the driver seek to the location provided. The attribute in the device status for prefetching is used to distinguish drivers that do more than just seek to the given location. The requests are low priority and preemptible by other requests for service. A READ LONG PREFETCH with 0 number of sectors to read should be treated as an advisory seek, and the driver can, if it is not busy, move the head to the starting sector. Since prefetching requests are advisory, there will be no functional difference between a device driver that supports prefetching from one that does not, except in terms of performance. The transfer address is not applicable for this call as the driver is not meant to transfer any data into the user address space.

SEEK

Command code = 131
ES:BX = SeekReq

```
SeekReq   DB   13 dup (0); Request header
```

```
        DB   ?         ; Addressing mode
        DD   0         ; Transfer address
        DW   0         ; Number of sectors to read
        DD   ?         ; Starting sector number
```

Control returns immediately to the caller without blocking and waiting for the seek to be completed. The number of sectors to be read and the transfer address are ignored. SEEK is used to relocate the head in order to begin playing audio or video tracks, or in anticipation of reading in a particular region on the disk. Further requests for disk activity will wait until the given SEEK is completed. This seek is not advisory and the head must move to the desired location.

PLAY AUDIO

Command code = 132
ES:BX = PlayReq

```
PlayReq   DB   13 dup (0); Request header
        DB   ?         ; Addressing mode
        DD   ?         ; Starting sector number
        DD   ?         ; Number of sectors to read
```

This function will cause the driver to play the selected audio tracks until the requested sectors have been exhausted or until play is interrupted with a AUDIO STOP request. Control returns immediately to the caller. Monitoring the busy bit in the status word will determine if the drive is presently playing audio and also when the play request is completed.

STOP AUDIO

Command code = 133
ES:BX = StopPlayReq

```
StopPlayReq    DB   13 dup (0)     ; Request header
```

This function is included to interrupt the drive unit when it is currently in play mode. At the next stopping point it reaches, the drive will discontinue playing and process the next request. If the drive is not currently playing or does not support playing, this request is ignored.

RESUME AUDIO

Command code = 136
ES:BX = ResumeReq

```
ResumeReq DB   13 dup (0)     ; Request header
```

This function is used to resume playing audio tracks when play has been interrupted with the STOP AUDIO command. Its behavior should correspond to the following:

```
 RESET, NEW DISC, PLAY/RESUME COMPLETED
     playing = FALSE;
     paused  = FALSE;
     last_startloc = 0;
     last_endloc   = 0;
```

```
PLAY_AUDIO(startloc, endloc) {
    if (play(startloc, endloc) != SUCCESSFUL) {
        return error;

    playing = TRUE;
    paused = FALSE;
    last_startloc = startloc
    last_endloc = endloc
    return no error;
    }

STOP_AUDIO() {
    if (playing) {
        last_startloc = present q-channel location
        playing = FALSE;
        paused = TRUE;
        if (stop() == SUCCESSFUL)
            return no error;
        return error;
        }
    else {
        playing = FALSE;
        paused = FALSE;
        last_startloc = 0;
        last_endloc = 0;
        return no error;
        }
    }

RESUME_AUDIO() {
    if (paused) {
        if (play(last_startloc, last_endloc) != SUCCESSFUL)
            return error;
        playing = TRUE;
        paused = FALSE;
        return no error;
        }
    else
        return error;
```

Note that the playing flag corresponds to the state that should be reported
by the busy bit in the status word in the request header when the drive is
in audio play mode. The paused flag corresponds to the Audio Paused bit and
last_startloc and last_endloc correspond to the starting and ending location
in the Audio Status Info IOCTL.

WRITE LONG

Command code = 134
ES:BX = WriteL

```
WriteL    DB   (dup 13 0); Request header
        DB   ?         ; Addressing mode
        DD   ?         ; Transfer address
        DW   ?         ; Number of sectors to write
        DD   ?         ; Starting sector number
        DB   ?         ; Write mode
        DB   ?         ; Interleave size
        DB   ?        ; Interleave skip factor
```

The device will copy the data at the transfer address to the CD RAM device at the sector indicated. The media must be writable for this function to work. Data is written sector by sector, depending on the current write mode and the interleave parameters. The following values are recognized as valid write modes:

0	Mode 0
1	Mode 1
2	Mode 2 Form 1
3	Mode 2 Form 2
4-255	Reserved

Writing in Mode 1 is the default and must be supported. If the device driver supports the other modes, then they can be used. If Mode 0 is used, the transfer address is ignored and all sectors are written with zeroes. If the current write mode is Mode 1 or Mode 2 Form 1, each sector will consist of 2048 bytes of data located sequentially at the transfer address. If the write mode is Mode 2 Form 2, the device driver will expect 2336 bytes of data per sector at the transfer address.

WRITE LONG VERIFY

Command code = 136
ES:BX = WriteLV

```
WriteLV   DB   (dup 13 0); Request header
        DB   ?        ; Addressing mode
        DD   ?        ; Transfer address
        DW   ?        ; Number of sectors to write
        DD   ?        ; Starting sector number
        DB   ?        ; Write mode
        DB   ?        ; Interleave size
        DB   ?        ; Interleave skip factor
```

This function is identical to WRITE LONG, with the addition that the device driver is responsible for verifying the data written to the device.

INPUT FLUSH

Command code = 7
ES:BX = FlushI

```
FlushI      DB   13 dup (0)     ; Request header
```

Requests that the device driver free all input buffers and clear any pending requests.

OUTPUT FLUSH

Command code = 11
ES:BX = FlushO

```
FlushO      DB   (dup 13 0)     ; Request header
```

Requests that the device driver write all unwritten buffers to the disk.

DEVICE OPEN
DEVICE CLOSE

Command code = 13,14
ES:BX = DevOpen, DevClose

```
DevOpen   DB   13 dup (0)     ; Request header
```

Used by the device driver to monitor how many different callers are currently using the CD-ROM device driver. All new device drivers should support these calls even if nothing is done with the information.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

Function Requests Specification

There is a need for access to features from the MSCDEX redirector that transend DOS capabilities. This proposal documents a means that the application can use to talk directly to MSCDEX to request information or set parameters that only MSCDEX can provide. This document outlines some of the features I think MSCDEX should support. Comments and suggestions are welcome.

Access to these functions is provided through an INT 2Fh interface. AH contains 15h which is what MSCDEX will use to tell its requests from those of other INT 2Fh handlers. AL will contain the code of the function to be performed.

Function Request Command Codes:

Contents of AL	Function
00h	Get Number of CD-ROM Drive Letters
01h	Get CD-ROM Drive Device List
02h	Get Copyright File Name
03h	Get Abstract File Name
04h	Get Bibliographic Doc File Name
05h	Read VTOC
06h	Turn Debugging On
07h	Turn Debugging Off
08h	Absolute Disk Read
09h	Absolute Disk Write
0Ah	Reserved
0Bh	CD-ROM Drive Check
0Ch	MSCDEX Version
0Dh	Get CD-ROM Drive Letters
0Eh	Get/Set Volume Descriptor Preference
0Fh	Get Directory Entry
10h	Send Device Request
11h-0FFh	Reserved

Get Number of CD-ROM Drive Letters

AX	1500h
BX	Number of CD-ROM drive letters used
CX	Starting drive letter of CD-ROM drive letters (A=0, B=1, ...Z=25)

MSCDEX will return the number of CD-ROM drive letters in BX and the starting
drive letter in CX. The first CD-ROM device will be installed at the
starting drive letter and subsequent drives will be assigned the next
greater drive letter. A single device driver may be assigned to more than
one drive letter, such as the case of a device driver that supports multiple
units. MSCDEX keeps track of which sub-unit a particular drive letter is
assigned to.

ÄÄ
ÄÄÄÄÄÄÄÄ
NOTE:
  This function can be used to determine if MSCDEX is installed by setting
  BX to zero before executing INT 2Fh. MSCDEX is not installed if BX is
  still zero on return.
ÄÄÄ
ÄÄÄÄÄÄÄÄ

Also, in a networking environment, one cannot assume that drive letters will
always be assigned contiguously beginning with the starting drive letter.
Use function Get CD-ROM drive letters instead.

Get CD-ROM Drive Device List

```
    AX        1501h
    ES:BX     Transfer address; pointer to buffer to copy drive letter
            device list
```

The buffer must be large enough to hold the device list. By calling function
Get Number of CD-ROM Drive Letters, one can find out the number of CD-ROM
drive letters and the buffer size will be a multiple of that. This will be
an absolute maximum of 26. Each drive letter device entry will consist of
one byte for the sub-unit followed by 4 bytes for the address of the device
header assigned to that drive letter. This byte for the sub-unit takes care
of the problem of distinguishing which unit is assigned to which drive
letter for device drivers that handle sub-units.

For example: Suppose there are two installed CD-ROM device drivers, FOO,
which supports 1 sub-unit, and BAR, which supports two sub-units, on a
system with 2 floppy drives (A=0 and B=1) and a hard disk (C=2). Then asking
for the number of CD-ROM drive letters will report that there are 3 drive
letters used starting at drive letter D=3. ES:BX must point to a buffer that
is at least 3 * 5 = 15 bytes long. The buffer will be filled as follows:

```
ES:BX  = Buffer

Buffer   DB   0                    ; sub-unit of FOO on drive letter D:
        DD   <far addr of FOO device header>
        DB   0                     ; sub-unit of BAR on drive letter E:
        DD   <far addr of BAR device header>
        DB   1                     ; sub-unit of BAR on drive letter F:
        DD   <far addr of BAR device header>
```

Get Copyright File Name

```
    AX     1502h
    ES:BX  Transfer address; pointer to a 38 byte buffer
    CX     CD-ROM drive letter (A=0, B=1, ... Z=25)
```

MSCDEX will copy the name of the copyright file in the VTOC for that drive
letter into the buffer space provided. The copyright filename is presently

restricted in the High Sierra proposal to 8.3 but we require 38 bytes here
for the possibility at a later date of handling 31 character file names plus
6 bytes for a ';' and 5 digit version number and 1 byte for a NULL at the
end. Carry will be set if the drive letter is not a CD-ROM drive and
error_invalid_drive (15) will be returned in AX.

Get Abstract File Name

```
    AX     1503h
    ES:BX  Transfer address; pointer to a 38 byte buffer
    CX     CD-ROM drive letter (A=0, B=1, ... Z=25)
```

MSCDEX will copy the name of the abstract file in the VTOC for that drive
letter into the buffer space provided. The abstract filename is presently
restricted in the High Sierra proposal to 8.3 but we require 38 bytes here
for the possibility at a later date of handling 31 character file names plus
6 bytes for a ';' and 5 digit version number and 1 byte for a NULL at the
end. Carry will be set if the drive letter is not a CD-ROM drive and
error_invalid_drive (15) will be returned in AX.

Get Bibliographic Documentation File Name

```
    AX     1504h
    ES:BX  Transfer address; pointer to a 38 byte buffer
    CX     CD-ROM drive letter (A=0, B=1, ... Z=25)
```

ÄÄÄ
ÄÄÄÄÄÄÄÄ
NOTE:
  This function is provided in advance of the ISO standard. For discs
  complying with the May 28th draft from the High Sierra Group, this
  function will return a null string as though the field is blank on the
  disc.
ÄÄ
ÄÄÄÄÄÄÄÄ

MSCDEX will copy the name of the bibliographic documentation file in the
VTOC for that drive letter into the buffer space provided. The bibliographic
documentation filename is presently restricted in the High Sierra proposal
to 8.3 but we require 38 bytes here for the possibility at a later date of
handling 31 character file names plus 6 bytes for a ';' and 5 digit version
number and 1 byte for a NULL at the end. Carry will be set if the drive
letter is not a CD-ROM drive and error_invalid_drive (15) will be returned
in AX.

Read VTOC

```
    AX     1505h
    ES:BX  Transfer address; pointer to a 2048 byte buffer
    CX     CD-ROM Drive letter
    DX     Sector index
```

This function is provided to scan the Volume Descriptors on a disc. A sector
index of 0 will read the first volume descriptor, 1 reads the second, etc.
If there is no error, then AX will return 1 if the volume descriptor read
was the standard volume descriptor, 0FFh if it was the volume descriptor
terminator and there are no more volume descriptors to be read, and 0 for
all other types.

If there is an error in processing the request, the Carry Flag will be set

and AL will contain the MS-DOS error code. These will be either
error_invalid_drive (15) or error_not_ready (21).

Turn Debugging On

AX	1506h
BX	Debugging function to enable

This is used for development and is reserved. It will be non-functional in the production version of MSCDEX.

Turn Debugging Off

AX	1507h
BX	Debugging function to disable

This is used for development and is reserved. It will be non-functional in the production version of MSCDEX.

Absolute Disk Read

AX	1508h
ES:BX	Disk Transfer Address; pointer to a buffer to copy data to
CX	CD-ROM Drive letter (A=0, B=1, ... Z=25)
DX	Number of sectors to read
SI:DI	Starting sector

This function corresponds to INT 25h. It will be converted directly into a READ_LONG device driver request and sent to the correct device driver. There are no requirements for this call to pop flags as there are with INT 25h. SI holds the high word and DI the low word for the starting sector to begin reading from.

If there is an error in processing the request, the Carry Flag will be set and AL will contain the MS-DOS error code. These will be either error_invalid_drive (15) or error_not_ready (21).

Absolute Disk Write

AX	1509h
ES:BX	Disk Transfer Address; pointer to buffer to copy data from
CX	CD-ROM Drive letter
DX	Number of sectors to write
SI:DI	Starting sector

This function corresponds to INT 26h. It is not supported at this time and is reserved. It is intended to be used by authoring systems.

CD-ROM Drive Check

AX	150Bh
BX	Signature word
CX	CD-ROM Drive letter (A=0, B=1,...Z=25)

This function returns whether or not a drive letter is a CD-ROM drive supported by MSCDEX. If the extensions are installed, BX will be set to ADADh. If the drive letter is supported by MSCDEX, then AX is set to a non-zero value. AX is set to zero if the drive is not supported. One must be sure to check the signature word to know that MSCDEX is installed and that AX has not been modified by another INT 2Fh handler.

MSCDEX Version

```
    AX     150Ch
    BX     MSCDEX Version
```

This function returns the version number of the CD-ROM Extensions installed on the system. BH contains the major version number and BL contains the minor version. Values returned are binary. For example, BX would contain 0x020a for version 2.10. This function does not work on versions earlier than 2.00 so if BX is zero before and after this function is called, an earlier version of MSCDEX is installed.

Get CD-ROM Drive Letters

```
    AX     150Dh
    ES:BX  Transfer address; pointer to buffer to copy drive letter
           device list
```

The buffer must be large enough to hold a list of drive letters. The buffer size will be a multiple of the number of drives returned by the Get Number of CD-ROM Drive Letters function. There are a maximum of 26 drive letters. Each drive letter entry is a single byte (0=A:, 1=B: .. 25=Z:) that exactly corresponds each respective entry returned by the command Get CD-ROM Drive Device List. This command is included to allow applications to locate CD-ROM drives supported by MSCDEX. CD-ROM drive letters may sometimes be noncontiguous so this command is necessary.

For example: Suppose there is an installed CD-ROM device driver FOO supporting 3 sub-units on a system with 2 floppy drives (A=0 and B=1), a hard disk (C=2) and a network drive (E=4). Note the network drive occupies one of the drive letters normally taken by a CD-ROM drive. MSCDEX assigns that CD-ROM drive to the next available drive letter. Asking for the number of CD-ROM drive letters reports there are 3 drive letters used starting at drive letter D=3. ES:BX must point to a buffer that is at least 3 bytes long and will be filled as follows:

```
ES:BX   = Buffer

Buffer  DB   3           ; drive letter for CD-ROM (D=3)
        DB   5           ; drive letter for CD-ROM (F=5)
        DB   6           ; drive letter for CD-ROM (G=6)
```

Get/Set Volume Descriptor Preference

```
    AX     150Eh
    BX     0 - Get Preference. 1 - Set Preference
    CX     CD-ROM Drive letter (A=0, B=1,...Z=25)
    DX     if BX = Get Preference
                DX = 0
                MSCDEX will return preference settings in DX
           if BX = Set Preference
                DH = volume descriptor preference
                    1 - PVD - Primary Volume  Descriptor
                    2 - SVD - Supplementary Volume Descriptor
                DL = Supplementary Volume Descriptor Preference
                     if DH = PVD
                           DL = 0
                     if DH = SVD
                           1 - shift-Kanji (an unregistered ISO coded
```

character set)

Normally, MSCDEX will scan for the PVD (Primary Volume Descriptor) when initializing a CD-ROM. This behavior can be altered for each individual drive to scan for a SVD (Supplementary Volume Descriptor) instead. A CD-ROM drive set to scan for an SVD will use the PVD if there is no SVD present. There can be more than one SVD on a CD-ROM but at present, MSCDEX will only recognize SVDs for shift-Kanji CD-ROMs. Carry will be set, AX will be set to error_invalid_function (1) and DX will be set to 0 if the coded character set is not recognized.

If BX contains Get_Preference, MSCDEX will report the present setting for that drive. If DX is still zero on return, that version of MSCDEX does not support this function or reading SVDs. Otherwise DX will contain the setting.

If the drive letter is not a CD-ROM drive, carry will be set and error_invalid_drive (15) will be returned in AX. If BX is anything other than Get/Set_Preference, AX will be set to error_invalid_function (1) and carry will be set.

Get Directory Entry

```
    AX     150Fh
    CX     CD-ROM Drive letter (A=0, B=1,...Z=25)
    ES:BX  Pointer to buffer with null-terminated path name
    SI:DI  Pointer to buffer to copy directory record information
    AX     0 is returned if the disc is High Sierra, 1 is returned if the
           disc is ISO-9660
```

The pathname expected is a null-terminated string e.g. char far *path = "\\a\\b\\c.txt"; (note: the "\\" characters map to a single '\' character in C so this would be '\a\b\c.txt' if printed). The path must consist only of valid High Sierra or ISO-9660 filename characters and must not contain any wildcards nor may it include entries for '.' or '..'.

The buffer to copy the directory record to can be a maximum of 255 bytes long including all system use information. The directory record is a direct copy from the directory file and it is up to the application to choose what fields to use.

Carry will be set and an error code returned if there were problems with the request. The error codes will be error_invalid_drive (15) if the drive letter is incorrect, error_not_ready (21) if the disc didn't initialize correctly, error_file_not_found (2) if the file was not found and error_no_more_files (18) if the pattern fails to find a match or if mscdex failed to allocate buffers.

The format of the directory record for High Sierra discs is:

```
    /* High Sierra directory entry structure */
typedef struct hsg_dir_entry {
   uchar     len_dr;         /* length of this directory entry  */
   uchar     XAR_len;        /* length of XAR in LBN's          */
   ulong     loc_extentI;    /* LBN of data Intel format        */
   ulong     loc_extentM;    /* LBN of data Molorola format     */
   ulong     data_lenI;      /* length of file Intel format     */
   ulong     data_lenM;      /* length of file Motorola format  */
   uchar     record_time[6];/* date and time                    */
   uchar     file_flags_hsg;/* 8 flags                          */
```

```
    uchar      reserved;      /* reserved field                  */
    uchar      il_size;       /* interleave size                 */
    uchar      il_skip;       /* interleave skip factor          */
    ushort     VSSNI;         /* volume set sequence num Intel   */
    ushort     VSSNM;         /* volume set sequence num Motorola*/
    uchar      len_fi;        /* length of name                  */
    uchar      file_id[...];  /* variable length name upto 32 chars       */
    uchar      padding;       /* optional padding if file_id is odd length*/
    uchar      sys_data[...]  /* variable length system data              */
    } hsg_dir_entry;
```

The format of the directory record for ISO-9660 discs is:

```
     /* ISO-9660 directory entry structure */
typedef struct iso_dir_entry {
    uchar      len_dr;        /* length of this directory entry  */
    uchar      XAR_len;       /* length of XAR in LBN's          */
    ulong      loc_extentI;   /* LBN of data Intel format        */
    ulong      loc_extentM;   /* LBN of data Molorola format     */
    ulong      data_lenI;     /* length of file Intel format     */
    ulong      data_lenM;     /* length of file Motorola format  */
    uchar      record_time[7];/* date and time                   */
    uchar      file_flags_iso;/* 8 flags                         */
    uchar      il_size;       /* interleave size                 */
    uchar      il_skip;       /* interleave skip factor          */
    ushort     VSSNI;         /* volume set sequence num Intel   */
    ushort     VSSNM;         /* volume set sequence num Motorola*/
    uchar      len_fi;        /* length of name                  */
    uchar      file_id[...];  /* variable length name upto 32 chars       */
    uchar      padding;       /* optional padding if file_id is odd length*/
    uchar      sys_data[...]  /* variable length system data              */
    } iso_dir_entry;
```

The difference between the two forms is the file flag byte moved to account for an additional byte of date and time used for a Greenwich mean time offset. See the May 28th draft of the High Sierra proposal or ISO-9660 for a more complete explanation of the fields. Note that the C structs above are not syntactically correct; C does not allow variable length arrays as struct elements.

Send Device Driver Request

```
     AX     1510h
     CX     CD-ROM drive letter (A=0, B=1, ... Z=25)
     ES:BX  Address of CD-ROM device driver request header
```

This function has been added to simplify communication with CD-ROM drivers and help prevent contention between applications that wish to communicate with the device driver. It is highly recommended that all applications communicate with device drivers through this function request. Applications using this function will not have to locate the device driver. The format of the request header is specified by the Microsoft MS-DOS CD-ROM Extensions Hardware-Dependent Device Driver Specification.

ŰŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

Networking CD-ROMS

Although it is possible to share CD-ROM drives on a local area network or

LAN, it is not as easy as it should be. While MS-DOS provides a single, stable platform to develop a file system driver like the Microsoft CD-ROM Extensions, there are a wide variety of LANs and LAN server implementations that do not provide a stable platform for which a file system driver like MSCDEX could be provided. Because each LAN implementation takes a different approach for server support, the approach for CD-ROM support on a server depends on what LAN implementation is being used.

This document should help clarify the present situation and help get you started.

At present, there are several CD-ROM products that allow sharing of CD-ROM drives on a LAN. LAN support may range from very simple and inexpensive to not so simple and inexpensive. At present, there are three products presently available that offer some form of LAN support. These are:

Microsoft	MSCDEX - The Microsoft CD-ROM Extensions
Meridian Data	CD-NET
Online	Opti-Net

Choosing which product depends on your LAN and your needs.

There are some LANs, such as Lantastic by Artisoft, that can share CD-ROM drives using any version of MSCDEX on a Lantastic server. This is possible because their servers run as an MS-DOS application and do I/O with standard MS-DOS INT 21 services. LAN servers like this, that do not make assumptions about the underlying media or try to bypass MS-DOS and do use standard MS-DOS INT 21 services to access the drive letter, will likely work with all versions of MSCDEX.

There are several LAN products based on MS-NET or a similar LAN server model such as Ungermann-Bass or 3COM. Unfortunately, these products do not access files on the server using standard INT 21 calls and for several reasons due to assumptions inside MS-DOS about non-standard calls from the server, you cannot share CD-ROM drives on MS-NET based servers. Although the server seems to allow sharing of the CD-ROM drive letter, requests to the server from workstations do not work correctly.

Fortunately, MSCDEX Version 2.10 has a command line switch (/S) that instructs MSCDEX to patch the in-memory image of MS-DOS during its initialization to fix these problems. By including this parameter on the MSCDEX command line, MSCDEX can be loaded before the network server software is started and the CD-ROM drive letters can then be shared by MS-NET based server software and workstations will see the correct behavior. This solution requires only that the server use MSCDEX Version 2.10 and no software or hardware changes to the workstation. Only the server runs MSCDEX or loads any CD-ROM related device drivers. To the workstation, the CD-ROM server drives are indistinguishable from other server drives.

For LAN products that are not MS-NET based yet have NETBIOS support such as Novell or IBM PC-NET, both Optinet and Meridian Data have adapted the MSCDEX and CD-ROM Device Driver model to provide LAN CD-ROM support. Each workstation runs MSCDEX and a special CD-ROM device driver that accepts normal CD-ROM driver requests from MSCDEX and uses the NETBIOS to transmit the command to a network driver on a server that submits the request to a true CD-ROM device driver on the server and transmits the results back to the workstation pseudo CD-ROM driver which in turn responds to MSCDEX. So long as the workstation CD-ROM device driver responds appropriately, MSCDEX is unaware that the command has passed through the network to a server. Contact Meridian Data and Online for information for these networks as they can both describe their products and features best.

Online offers one potential configuration for computer systems that do not
wish to dedicate a machine to be a server. The workstation operates as above
but instead of communicating the workstations driver request to a dedicated
server process, another user's workstation running a special TSR version of
their system can field the driver request, submit it to the CD-ROM driver,
and respond to the requesting workstation. This allows a network of
workstations to share the CD-ROM drives that each computer has connected to
it at the same time all workstations are available to the users. This option
does slow performance of the workstation when outside requests come in and
does use up valuable memory for the TSR system code but for some this option
may work.

At present, there is no available version of the CD-ROM Extensions for OS/2
although there is a way to access CD-ROM data in OS/2 on a network. Since
from the outside, workstations cannot tell MS-DOS server drives that are
shared CD-ROM drives using version 2.10 of MSCDEX from traditional block
drives, even OS/2 machines can access the CD-ROM drive on the server.
Although this does mean including an MS-DOS server on an OS/2 LAN, it does
provide at least an interim way to access CD-ROM data under OS/2 at this
time.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

Kanji Support

The Kanji support in MSCDEX presently recognizes High Sierra CD-ROM discs
with a coded character set that has bit 0 set to 1 in the volume flags
indicating at least one escape sequence is not registered according to ISO
2375, and has an escape sequence of three bytes in the coded character set
for descriptor identifiers field of "$+:". This indicates that the character
set is a private multi-byte G3 coded character set and identifies the disc
as having shift-Kanji.

In order to make MSCDEX scan for the SVD (Supplementary Volume Descriptor)
instead of the PVD (Primary Volume Descriptor), there is a new command line
argument /K. If this is present, MSCDEX will use the shift-Kanji SVD if it
is present, otherwise it will use the PVD. All discs are required by ISO-
9660 to have a PVD even if there is an SVD.

In addition, there is an accompanying program SVD that can be used to change
the default preference each CD-ROM drive has for scanning for a SVD or PVD.
The syntax is:

```
  SVD [<drive letter>: <std  svd>]
```

Running SVD with no arguments will report the current settings. Including a
drive letter and either STD or SVD will change the preference for that drive
from one to the other.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

CD-ROMifying Your Software

CD-ROM is the first of what will probably be several alien file structures
that will start appearing in the MS-DOS world primarily with the
introduction of installable file systems under newer versions of DOS. The

following will attempt to outline some guidelines for writing software that will help in porting your software to these new file systems and for CD-ROM specifically.

Choice of Filename Characters

On the first Microsoft Test CD-ROM disc, the Codeview demo failed because certain filename characters that were legal on MS-DOS were not allowed according to the High Sierra file format. When the software looked for file 'S1.@@@', it wasn't found because the character '@' is illegal for High Sierra filenames and during High Sierra premastering, the file was renamed 'S1'.

Valid High Sierra filename characters are the letters 'A' through 'Z', the digits '0' through '9', and the underscore character '_'. All other characters are invalid. Note that the letters 'a' through 'z' are not included so that High Sierra file names are not case sensitive. Under DOS, filenames are mapped to upper case before they are looked up so this is typically not a problem. When choosing file name characters, keep in mind the restrictions of the file structure format and the operating systems your media may be targeted towards.

Depth of Path

The High Sierra format allows for pathnames to be up to 8 levels deep. It's possible to create a path on MS-DOS that is deeper than that but you won't be able to transfer it to a CD-ROM.

```
  \one\two\three\four\five\six\seven\eight\file.txt /* Ok */
  \one\two\three\four\five\six\seven\eight\nine\file.txt /* Illegal */
```

Length of Path

The High Sierra format allows for the entire pathname to be a maximum of 255 characters. Since MS-DOS imposes a limit far lower than this, this should not present a problem. The MS-DOS call to connect to a sub-directory is limited to a directory string of 64 characters. The length of path restriction is more a concern for Xenix/Unix than MS-DOS.

Amusingly enough, the MS-DOS call to create a sub-directory allows a directory string greater than 64 characters which allows you to create sub-directories that you cannot connect to.

Unfortunately, a CD-ROM may potentially contain a pathname that is much larger than 64 characters long. This is not a concern here but is discussed in a related memo - "MS-DOSifying your CD-ROM". As a rule, try to keep the length of your longest path less than 64 characters and you should be pretty safe.

Read-only

Even though most people understand that CD-ROM discs are read-only, there's still a lot of software written by these same people that assumes the current disk is always writable. For example, the Microsoft Multiplan Demo assumes that it can create and write temporary files to the presently connected drive.

In order to avoid this problem, try to provide another means of letting the user specify where temporary files can be created. Many applications check the environment for the variables TMP or TEMP which contain the pathname to use when creating temp files. Most people understand this convention now (or should anyway) and an added benefit will be the speed improvement that will be recognized if the temp directory is located on a ram-drive. If the environment variable is not set, then the application can fall back on the assumption that the media is writable or ask where temporary files should be kept.

As a rule, for both temporary and permanent files, if a file creation error occurs, allow the user to re-specify the pathname used so that he can work around the error. The last thing that should happen is for work to be lost because the user was not allowed to store his output in a valid place.

Non-DOS Formatted Disks

Don't depend on the format of data on the disk. CD-ROM's do not have a FAT so don't even bother looking for one. Do not talk to any media at a physical level (reading/writing sectors) unless you expect to be media dependent (such as CHKDSK or FORMAT). MS-DOS INT 21h calls should provide everything you need to get at the file contents and attributes.

Small Directories

For performance reasons, try to keep directory sizes smaller than about 40 or so. Much beyond this and directory files grow beyond one 2048 byte sector. Typically this is not a problem but if the number of sector buffers chosen when MSCDEX is started is small and the directory files are large, whatever software scanning the directory could potentially thrash badly if every time the directory is searched for the next entry it has to bring earlier directory sectors back into memory from the CD-ROM drive.

For certain pathological programs, such as certain implementations of the Xenix utility find, the penalty is about 1 second per directory sector that you have to scan to get to the next entry. If the directory is large, say 8 sectors, the time for FIND to scan that one directory could potentially take a half hour for something that would take less than a second if all the entries fit in the cache.

The solution for this problem is to make sure that MSCDEX never throws out of the cache what it will need next. This is accomplished by growing the cache (very easy - simply change the parameter to MSCDEX) and to make sure that the largest object that goes through the cache will not clear it out. There is a balance between having too many directories and too many files in a few directories but the balance is heavily weighted towards many small to medium sized directories. Keep this in mind when laying out your files.

Since the penalty for using a file in the lowest sub-directory instead of the root-directory is virtually nil and as more directories don't cost much, it's a good idea to break up large directories into several smaller ones. This will help avoid problems of flushing the disc sector cache. Try to keep related files close together both in location on the CD-ROM and in the same directories. Close proximity will reduce seek time when accessing related files at the same time and having them in the same directory will help prevent swapping out directory sectors.

Updating CD-ROM Databases and Software

Many people are interested in providing updates to files that are contained on a CD-ROM disc. They would like to create a directory on their hard disk with all updated files in them and have the CD-ROM Extensions look there first before searching the CD-ROM. Unfortunately, by the time the Extensions get the request, it is very difficult for it to look for updates on the hard disk so whatever alternative searching that is necessary will have to be done in the application software.

For this reason, it's a good idea to have your path set so that it looks through directories on the hard disk first. Another good strategy is to copy executables to a directory on your hard disk so that they can be updated and will also start up faster. Also, have the application software itself search alternative hard disk directories for updates before it searches the CD-ROM. This way both software updates and updated or commonly used database files can be stored on a hard disk which will both speed performance and allow incremental updating.

Search Strategies

Try to avoid relying on the operating system to be part of your search strategy. If your database is broken up into a hierarchy and your order is imposed through the file structure by breaking up the database into many files in a tree, then accessing data in the database is typically going to require a lot of directory reading and searching.

Usually the time involved in doing this on a hard disk is not large but on a CD-ROM, the search times can add up. Opening a file can be an expensive operation simply because the directory must be read before the file can be opened. At best, seeking to a location on the CD-ROM can take 10 msec or so; at worst, a seek can run to over a second on some older CD-ROM drives. Some newer drives have worst case seek of about half a second. Whenever this can be avoided you will save time. MSCDEX caches as many directory sectors as it can so that searching the most active directories is very quick but any operations that search multiple directories once through continually clears out the cache and renders it ineffective.

The strategy used by Microsoft Bookshelf was to lump the entire database into a single file and structure the indexing so that searching used a minimum of seeks. Bookshelf uses packed b-trees with each node holding as many entries as will fit into a single sector and also cache in memory as much of the root of the tree as it can.

Combining databases avoids the extra overhead of repeatedly opening and closing database files. Caching as much of the indexes in memory as possible allows searching of keywords to be completed typically with a single seek.

In general, identify your software bottlenecks and through judicious use of faster storage media (either memory or hard disk) you can both have large storage and respectable performance.

Portability

One of the advantages of the High Sierra format is data interchangeability with other operating systems. One must take care to chose a subset of the range of High Sierra features that are presently supported across different

operating systems to be sure you can read the disc in each of them. The
lowest common denominator then (this list is not complete - see also what
must be done to target MS-DOS) would need a logical block size of 512 bytes,
both type L and M path tables and for all fields, single volume sets, at
least one primary volume descriptor and terminator. Be aware that if one of
your goals is data portability, you will have to do some additional research
to see what restrictions on the High Sierra format other operating systems
may impose.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

MS-DOSifying Your CD-ROM

Most of the following caveats apply to the present version of the Microsoft
CD-ROM Extensions. Future versions of the extensions are expected to support
many of the features listed below that are at present best avoided. The
behavior of the extensions with fields and records that are presently
ignored may change at any time.

Correctness

Make sure that your disc is in valid High Sierra format. Nothing is
guaranteed if your disc is not in valid format. Surprisingly enough, we have
received several discs that have one or more illegally formatted data areas
ranging from directories being sorted incorrectly, incorrect path table
sizes, incorrect directory file sizes, directories missing from the path
table, invalid directory names, etc. In almost every case, the Extensions
will behave incorrectly and at worst, the system will crash.

In addition to running validation software to verify the High Sierra image,
one should also verify that the Extensions work with your CD-ROM disc and
application software before distributing it. Unfortunately, it may not
matter if your disc is correct and the Extensions are wrong if they don't
work together. Please report any and all problems you think are in the
Extensions to Microsoft so that they can be fixed.

Pathtable and Directory Sizes

This bears repeating because many people have gotten it wrong. Directory
file sizes are always a multiple of the logical sector size - 2 kilobytes.
Path table sizes are always the exact number of bytes that are contained in
the path table which is typically not a multiple of 2k. You must not have
blank directory sectors and the directory length must reflect the correct
length of the directory file. The directory sectors always begin on a
logical sector boundary.

8.3 File Names

MS-DOS cannot handle longer than 8.3 filenames. If the CD-ROM filename is
longer than 8.3, then the filename will be truncated. If this happens, two
files that are not unique within 8.3 characters will map to the same
filename. For example:

```
  filename1.txt will appear as filename.txt
  filename2.txt will also appear as filename.txt
```

Kanji filenames are also limited to 8.3 or 4.1 kanji characters. Only shift-
kanji filenames are recognized at present. To get kanji, you must specify a
supplementary volume descriptor indicating you have kanji filenames. Contact

Microsoft to find out how this is done.

Record Formats

The extensions do not support any record formats so if the RECORD bit is set in the file flags byte in the directory entry for a file, it will be ignored.

Interleaving

In the present version, the Extensions do not support interleaving so if the Interleave size and Interleave factor are non-zero, the file will ignore these fields and return erroneous data.

Multi-Extent Files

Multi-extent files are not supported in the present version. Each extent of a multi-extent file will appear as a separate file with the same name.

Multi-Volume

Multi-volume disc sets are not supported in the present version. Directories that are located on another volume could potentially cause the Extensions to crash if searched and erroneous data will be returned for files that are located on another volume.

Coded Character Sets

Only one coded character set or supplementary volume descriptor is recognized in the latest version. This is for shift-Kanji.

Version Numbers

Version numbers are not supported by the Extensions. The Extensions will strip the version string off the end of the filename so that two identical filenames with different versions will appear to have the same name. There is no way to specifically ask for any but the first instance of that filename. Two files with the same name and different version numbers have the same accessing problem as two files with longer than 8.3 filenames that have been truncated to the same filename.

Protection

Protection bits are not used on MS-DOS. If the protection bit is set in the file flags byte in the directory entry for a file though, the file will not show up on any search or open even if the protection bits in the XAR are set to allow all access.

No XAR Support

At present, the Extensions ignore the contents of any XAR record.

Motorola Format Tables

The additional copies of the path table and any values in "Motorola" format (most significant bytes using the lowest address values) are ignored at present. MSCDEX only pays attention to "Intel" formatted values. They should be included though for portability sake.

Multiple Copies of the Path Table

The Extensions presently only read and use the first copy of the path table. Later versions may check to see that copies of the path table agree.

Additional Volume Descriptor Records

Boot records and Unspecified volume descriptors are ignored. The first standard volume descriptor found is the one that is used. Additional copies are ignored at present.

File Flags

The existence bit is treated the same as the hidden bit on MS-DOS. Some other operating systems may not handle the existence bit so you may not want to use it if you are targeting these systems. The directory bit for High Sierra is treated the same as the directory bit in MS-DOS. Files with the protection bit set are not found when searched for or opened.

None of the remaining bits, (Associated/Record/Multi-extent/Reserved), are handled at present. Using files with these bits set will have undefined behavior.

Unique Volume Identifiers

It is highly recommended that the volume identifier be unique. The Extensions use the volume identifier to do volume tracking and to double-check to see if the disc has changed. The more chance that users will have two discs with the same volume identifier, the more chance that this will confuse the Extensions and lead it to believe that the disc has not changed when in fact it has.

It is also highly recommended that application programs use the volume label to tell if the CD-ROM disc has changed. The volume label for a CD-ROM on MS-DOS is obtained from the volume identifier field in the primary volume descriptor. The call to get the volume label is very inexpensive to make once the CD-ROM has been initialized and will cause no disc I/O to be done unless the media has changed. This is the best way for an application to tell if the disc it wants to work with is in the drive. The application software should not communicate with the driver or drive to determine if the media has changed or the Extensions may not learn that the disc has changed and will not reinitialize what it knows about the new disc.

Many Small Directories or A Few Large Directories

As a rule, it is better to have many small directories that contain fewer files than one very large directory. The answer depends on your application's behavior because if you try very hard, you can thrash almost as badly with many small subdirectories as you can with one large subdirectory. Reading further will help explain.

What makes the difference? For each file open, suppose you have 1000 subdirectories with 40 files, on average you'll read about one sector per file open and scan 1/2 of it. On the other hand, you could have 1 directory with 4000 files. On average, each file open in this large directory (about 100 sectors) will involve scanning about 50 sectors to open that one file. As long as it is very inexpensive to get to each directory through the pathtable, clearly it is much better to have many small directories.

Further improvements can be made by grouping files that are related and will be opened together in each of these subdirectories so that as you open each successive file, the directory sector is very likely in the disc cache and this will help minimize hitting the CD-ROM disc. Putting each file in a separate subdirectory is extreme and will cost you because you will never gain the benefits of locating the next file in a directory sector that has already been cached and you will needlessly enlarge the pathtable.

There is a limit though to how many subdirectories you may want because if there are too many you may end up thrashing on the pathtable sectors. Each pathtable sector holds pointers to approximately 100 to 200 directory files depending on the directory name lengths. If you have a pathtable that is 10 sectors long, you will want at least 10 sectors of memory buffers to hold the pathtable or you may risk re-reading sections of the pathtable on every file open which will be very costly.

The most important point you can learn is that you can vastly improve your file open speed by making sure you have enough memory buffers. If you are repeatedly trying to scan a 10 sector directory file (approximately 400 entries) and you only have 4 sectors in the sector cache, the cache is going to work against you because you will end up churning it to death. If you allocate 14 sectors for example (/M:14), then the whole directory file will find its way into the cache and you will stop hitting the disc. The difference in speed may be several orders of magnitude. A safe bet is to recommend reserving as many sectors are in the pathtable plus the number of sectors for the largest directory plus 2. The last two are reserved for data sectors and internal dynamic data. This formula is complicated with multiple drives because the buffers are not tied to specific drives and are shared and because not all drives are active at the same time.

Another rule, do not rely on the file system to do your searching for you. If you are performance conscious, finding a chunk of data by looking for it with a file name through the file system is expensive if speed is a great concern. 99% of the time, locating data through the file system is fine because the cost is a single one time operation but if this is repeated often enough, it may pay to do some of the work yourself. What can be better is lump everything into one big file and cache your own hierarchy, indexing, binary trees, or whatever searching scheme you choose to use to get you to the data you need rather than asking for the file system to tell you where it is.

ŰŰ
ŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰŰ

Questions and Answers

Q: Does the /V option to MSCDEX actually cause a message to be displayed? I can't make it do anything. I use:

```
  MSCDEX /D:CDDRV /V /M:8
```

A: Yes, a series of statistics are displayed. MSCDEX uses INT 10 function 0eH to write to the console so if for some reason you are trapping this or have software that captures this and doesn't display bios output to the screen the information will not appear. All screen output uses this function so if any output appears on the screen after loading MSCDEX, It is not clear why this additional information does not appear.

Newer versions (2.0 and above) use the DOS write handle call for output which will fix problems of I/O redirection of this output.

As it turned out, the machine that had this problem was not a strict IBM compatible machine and did not emulate a PC at the BIOS level. Consequently, the messages written with INT 10h were not displayed.

Q: Is it normal for MSCDEX to hang the system if an error is returned by a driver's IOCTL function? Wouldn't an error message be better?

A: The only ioctl calls sent to the driver in version 1.01 are to request the device header address and media check. Media check calls will never hang no matter what is returned so long as the driver returns. Illegal values may not do what you want but it won't hang. Request device header address may hang if the driver fails to set error conditions correctly as DOS expects them as DOS will return from my ioctl call without error. MSCDEX will then assume the bogus return values are correct and jump to a random location.

Q: Does MSCDEX do anything that should preclude its working in a non-IBM-clone machine?

A: Except for the INT 10 problem mentioned earlier, no. If you can identify any problems whatsoever, we would be happy to learn about it but as yet, we have heard of no other problems. If your machine runs MS-DOS version 3.X, it should be capable of running the extensions correctly. As for driver/drive-controller compatability problems, that may be another matter. We do not guarantee anything about these because we do not write the device drivers or design the hardware interface boards and cannot make any claims concerning them. It is up to the drive manufacturer or device driver writer to do this kind of compatability testing.

Q: Based on what I read in the spec, I decided to support only HSG type addressing which seems to be allowed by the IOCTLI function #6 (Device Status). I return 4 bytes of 00h if that function is called. I would have thought that would be one of the first calls MSCDEX would make (after "Find Header") but so far it hasn't called the status function. How will MSCDEX know enough not to use Red Book addressing if it doesn't check status first?

A: In version 1.01, ioctl function #6 is not called. This is not to say that in a future versions it will not (in fact it will). Since all device drivers must support some default functionality and as MSCDEX only uses the basic default now (only High Sierra addressing for example), it wasn't a problem that it didn't call this function to find out about non-default features that were supported.

Some software is being written that controls audio on a CD-ROM that expects Red Book Addressing and checks the device status to see that it is supported. The conversion algorithms are fairly simple and code fragments are provided.

Q: Can you provide more info on how the READ/WRITE device control string should work. Does the read device bytes command get information that was written by the write device control string?

A: As of yet, no one to our knowledge has used this. There are a couple other features of which this can be said. Again, this is not to say that they won't be used at some later time.

The purpose of these commands was to allow a standard way of delivering commands that were not specified in the CD-ROM device driver spec to the drive. For example, sending SCSI command strings and reading the responses from the drive. This function is deliberately open-ended and vague because it was intended to provide a catch-all mechanism for application programs to communicate requests or request data in ways that were not specified by the device driver spec. For application programs to use these functions they have to know the driver supports these functions and also how to communicate with that specific drive. The mechanism would let the driver do what it does best and worry about which ports and interrupts to use. This relieves the application program from these details and allow it to deal with controlling the device at a higher level.

Right now, if the driver does not support these functions, it should return an error for Unknown Command. One could test whether these two function were supported this way.

ÄÄ
Note:
  If there are commands which you feel should be supported by the device
  driver specification, please communicate them to us and we will consider
  adding them if they are of sufficient general interest.
ÄÄ

Q: The version of MSCDEX I am using is 1.01 dated 3/20/87@11:06am and is marked Evaluation Copy. Is this any less functional than a "licensed" copy?

A: No. There may be a few more bug fixes in the licensed copy that are not in the Evaluation copy but none that should prevent correct operation. If you do find any bugs, please let us know as we will fix any bugs in the software that are reported. Our policy is to try to reward users that report bugs with updates of the software that fix their problem.

Q: Would it be possible for me to get a sample source file for a driver?

A: Yes. Licensees are given source code to device drivers for:

SONY - complete

HITACHI - missing two modules. These are owned by Hitachi so we cannot supply them, though Hitachi will.

PHILIPS - this driver communicates with the Philips CD-ROM driver supplied by Philips.

Q: How can an application access CD-ROM drives that are subunits of one driver? The IOCTL calls do not take an argument for subunit. MSCDEX seems to handle this OK since when I do a directory of each CD-ROM in turn it accesses the correct drive. I do not see any clean way for an application to, for example lock the door on CD-ROM drive G: which is the third drive handled by the driver.

A: Requests all have a sub-unit field in the request header. Commands that one would expect to be directed to a specific drive, such as open door, are targeted at a particular drive through the use of the sub-unit field.

Q: What is the current release version number of MSCDEX? The version I have is version 1.01 that is marked EVALUATION COPY. When can I get a final release version of it? Also will that final version include the changes to do all I/O through MSDOS (i.e. no INT 10h).

A: The most recent released version is version 2.0. You can purchase this from a number of licensees including all drive manufacturers. An Extensions availability list is attached.

Q: Why doesn't MSCDEX allow IOCTL access via the drive letter (i.e. DOS func. 44h subfunc. #4,5), as if the CD-ROM were a drive. I understand that the driver is not a block device, but this is being handled already in some way since you allow a user to perform file I/O to a CD-ROM making it appear to be a block device. It would seem that all that would be necessary to accomplish this is to intercept IOCTL calls in the same way that file access calls are being intercepted.

A: MSCDEX doesn't presently hook int 21h, which is what this would involve. It's doubtful that this will change. It's not that much more difficult to open the file and send an IOCTL to the handle. File access calls are not caught at an INT 21h level but are caught from within DOS at another interface. CD-ROM drives are far more like network drives than traditional MS-DOS FAT file structure block drives and their drivers. For example, try to FORMAT a CD-ROM drive and you'll see. Part of all this prevents IOCTL's to the drive letter from being directed to the appropriate driver.

Q: Why not allow access to the PLAY, STOP and SEEK functions via the INT 2Fh entry point as is allowed for READ LONG. This would be much simpler than requiring the application to locate the driver header and then find the STRATEGY entry point and create request control blocks etc. This is a lot of code to start the music playing!

A: It's preferable to see MS-DOS provide extensions to the application program interface for audio/video control (new int 21h calls). The reason we haven't included play, etc. in the int 2Fh interface is to avoid loading down MSCDEX with additional functionality that most people don't use. Your suggestion would only move that code from the CD-playing program into MSCDEX. It makes your program smaller, but in the whole, doesn't buy much. As time goes on, this may change and some of the functionality may move into the int 2Fh interface.

Q: Shouldn't the specification eliminate the need for the application to OPEN the driver by name, This is especially important in systems where the driver creates a new driver header for each CD-ROM drive. MSDOS allows so few file handles to be simultaneously open as it is that requiring applications to open even more is very bad.

A: Simply close the driver handle after you have located the device header. You no longer need to communicate through DOS to control it, so free the handle and make it available for other programs to use. With version 2.10, it is no longer necessary to OPEN the device driver in order to communicate with it. Applications can communicate with the device driver using Send Device Driver Request.

Q: Have you considered adding an addressing mode for the PLAY AUDIO function that would allow the application to specify the PLAY address by TNO instead of block number or min/sec/frame?

A: This has been considered but has not been added to keep from complicating the device drivers unnecessarily. At the moment, most CD-ROM drives are used without audio so our intent was to put what was needed for audio support in the audio playing software. In addition, we chose to keep the interface simple to leave more latitude for changes to the OS/2 API that may include newer data types like audio and video. Nonetheless, this may be added in the future. In the meantime, audio playing software has the extra overhead of reading the audio table of contents and interpreting the tracks itself.

Q: Why is there no CLOSE TRAY function in the driver spec? The CD-ROM drive we are using (Toshiba SCSI) has this capability but I see no way to use it via the extensions. Why allow the user to OPEN it without allowing him to close it?

A: A close tray command has been added.

Q: It seems that there should be bits in the Device Status word to indicate whether a driver supports Reading/Writing device control strings.

A: Reading and writing device control strings was put it in as a catch-all for anything that was missed so that application programs could send specific commands through the device driver to the device if they understood the device and knew how to communicate to it. A manufacturers CD-ROM diagnostic program would be an example of a program that might choose to communicate with the drive in this way. If the driver does not support these functions, it should return an error. One can test whether these two function are supported by testing if the error returned is for Unknown Command.

Q: In the spec, treatment of the BUSY bit in the status word with regard to the PLAY AUDIO function seems to assume only one CD-ROM drive. What happens when the user has two or more drives each of which want to be playing music? How does the application tell whether the desired drive is busy? It would seem better to use some of the bits in the upper word of Device Status to indicate BUSY for each drive. Perhaps allowing 8 or 16 drives.

A: Requests all have sub-unit numbers associated with them. A request for service from one sub-unit may report that the drive is busy at the same time another sub-unit was not busy. The sub-unit field is used to distinguish requests between the drives supported by the driver. The busy bit serves as an indication of drive status for the sub-unit the request is for.

Q: If a CD-ROM file is opened in write-only mode, no error occurs.

A: True. The same happens on a floppy drive with a write-protect tab on it. If you do have a handle to a CD-ROM file in open_for_write mode, as soon as you attempt to write, you will get an error. The correct model is to duplicate the behavior of a file that has been set READ-ONLY. Read-only files return error_access_denied if you try to open them in open_for_write or open_for_both modes. MSCDEX has been changed to duplicate this behavior.

Q: What other non-MSDOS calls are issued by MSCDEX besides INT 10h?

A: INT 2Fh - dos callbacks
  INT 67h - expanded memory
  INT 10h - all INT 10h calls went away with version 2.00 which uses DOS
      write handle instead.

Q: Why does MSCDEX do the READ LONG PREFETCH call after it has done a DEVICE CLOSE call? Is this intentional?

A: MSCDEX version 1.X never issued device open or close. These were issued by DOS as part of driver initialization. MSCDEX version 2.00 now issues device open calls and will precede the prefetch call.

Q: In the device driver spec, it says that if more than one unit is supported by the driver that this field should be set to the number of units. I suspect that this is wrong since this is not a block device. As far as I can see, this field should only ever be set to one since each unit will actually have its own header with its own unique name.

A: CD-ROM device drivers are a hybrid of block and char device drivers and
are not technically legal as one or the other. Block drivers make some
assumptions about the media format that aren't meaningful for CD-ROM and
don't have a read call that can deal with CD-ROM's large data space. They
were made as char devices with some additional calls and rules. One of the
changes that was made for CD-ROM device drivers was to allow multiple sub-
units for the device so the treatment of this field is correct as specified
even though CD-ROM device drivers are character device drivers.

If one has more than one CD-ROM drive, one can approach supporting them
from several ways. One could have separate device drivers for each drive and
load one per drive, have a single driver with multiple device headers, or
have a single driver with one device header that supports sub-units. This
last method is borrowed from block drivers. For the case that the drives and
drive commands are all the same, using sub-units will allow you to
distinguish between which drive receives which command. The alternatives
clutter things up with drivers or device headers. Sub-units may not be legal
character device driver fields but conceptually, they're the right thing.
Since CD-ROM device drivers could not be block drivers and had to be char
device drivers, some liberties were taken with the specification to merge
the best of both specifications.

Q: Is there any support through MSCDEX for WRITE LONG? I have a need for
this to support a CD mastering system. I would like to be able to treat a
WORM drive as a CD-ROM and allow writing to the drive once to create a
master and then be able to test it out by using it as CD-ROM to verify that
our data has been correctly stored in "High Sierra" format.

A: Such a call exists. It only serves to define a standard interface for CD-
ROM device drivers that are running over re-writable mediaÄÄsuch as a CD
mastering system. It is in the latest copy of the driver spec released with
version 2.00 of the CD-ROM Extensions.

Q: How important is it that I should support RAW mode access in my driver?
What would this typically be used for?

A: Not important now. No drive presently support reading raw at the moment
that we know of. Since drives and their command capabilities are hardware
dependent, you would know based on the capabilities of your hardware if you
wanted to support it. This function was added for completeness. A standard
way was needed to define how to get at the 288 bytes of EDC/ECC if drives
allowed access and to have an avenue prepared if people found useful
applications that would not use EDC/ECC where they wanted the additional
space (such as gracefully degrading low-fidelity audio or graphics). It will
be useful for copying audio information or for audio systems that will want
to be able to manipulate audio tracks. Many people have expressed interest
in having this capability.

Q: Driver spec page 13 (in the structure for the UPC Code function): "The
UPC/EAN code (last 4 bits are zero)". Does this mean the low order or high
order 4 bits?

A: This is less ambiguous if you read Red Book under mode-2 of the Q-channel
info. This is now clarified in the UPC Code call. It should be the low
nibble of byte 7. Red Book specifies that MSB comes out first so the high
nibble will contain the 13th nibble of the UPC code and the 14th nibble will
be zero.

Unfortunately, scanning for the UPC code is time consuming especially if
it is done by the software. This is due to the design of the q-channel in
Red Book. It's a pity because this could be a useful number to verify the
correct disc has been inserted. Most CD-ROMs do not have a UPC code or have
it zeroed out. The same seems to be true for CD-audio as well. We believe
that CD-ROM drives should scan for the UPC code as they read the Table of
Contents when initializing from power up or a new disc. If the hardware does
not do this, the UPC code has to be scanned by polling the q-channel which
may occasionally miss the UPC code.

Q: It would be nice if the device driver spec included a list of what types
of disk access functions would and would not work so that users could get an
idea of what utilities and applications will and will not work with the
extensions.

A: The device driver specification describes just what is necessary for
writing a CD-ROM device driver. The information you would like concerning
things such as INT 25h/26h not supported as well as the behavior
CHKDSK/FORMAT/etc belongs and is mentioned in the MSCDEX overview.

Q: If I have a low priority request and if the system has time, can the
prefetch request read into and transfer the data into a transfer address?

A: We have looked at this for some time but the bottom line is that
asynchronous I/O under DOS is virtually impossible to support in all cases.
Because of this it is unlikely we will be implementing this or designing it
into the device driver interface. There is no way for MSCDEX or the CD-ROM
device driver to know that the transfer address is still valid because DOS
never notifies MSCDEX or the device driver if the requesting process was
been terminated. The request runs the risk of writing over another program.
The best approach now is if the driver wants to, it can reserve internal
buffer space for data from the disc and put prefetched data there. Then it
can copy the data to the read transfer address once the read request finally
arrives. Alternately, some of the caching or prefetching can reside in the
CD-ROM controller or in the drive itself. True asynchronous reads will have
to wait for OS/2.

Q: Is there any status indication that the transfer has occurred? or some
interaction with the read long command?

A: There is no way to tell if a prefetch request was successful or the state
of it. The prefetch simply provides a hint and the read request later is the
request that finally expects delivery of the data.

Q: Read (ioctl input) When audio is playing, can location of head move?

A: I'm not entirely sure what you mean by this question but I will attempt
to answer a couple different ways and hope I provide the information you
need.

While audio is playing, the head is moving on the CD. If the driver
receives a command asking where the head is, the driver should ask the drive
where the head is presently positioned and report that. So as audio is
playing, an application program that is controlling the audio can monitor
the progress of audio playback and can either synchronize actions with the
audio or report the present location to the user. For example, a program to
play audio discs would be able to report the present track number and time
elapsed on the computer monitor much like a CD-audio player reports things
on its LED display. If the driver is sent a command that requires the
movement of the head or a change in the state of the machine (SEEK, READ,

INITIALIZE, PLAY AUDIO etc.) then the audio will be interrupted so that the drive can perform the new request. If the drive receives a command for status or some other function that does not require the head to be moved or the state of the machine to change, then audio play should continue.

Q: Are the parameters of Audio Disk Info and Audio Track Info BCD or binary?

A: The parameters were vaguely specified in the device driver spec. The spec has been clarified. They are as follows:

ţ Audio Disk Info

binary	Lowest Track Number (1-99)
binary	Highest Track Number (1-99)
red book	Starting point of lead-out track

ţ Audio Track Info

binary	Track Number (1-99)
red book	Starting point of track
as is	Track control information

The track control information is not a BCD or Binary number like the track number. The byte contents as it appears on the disc should be carried through unmodified by the driver and is interpreted according to the Philips/Sony Red Book standard.

Q: Are the parameters for the Audio Q-channel info BCD or binary?

A: The parameters are as follows:

ţ Audio Q-Channel Info

as is	Control and ADDR byte
as is	Track Number
as is	Point or Index (X)
red book	MIN/SEC/FRAME
zero	ZERO
red book	AMIN/ASEC/AFRAME or PMIN/PSEC/PFRAME

The notes on when to convert from BCD to red book addresses for MIN/SEC/FRAME, AMIN/ASEC/AFRAME and PMIN/PSEC/PFRAME is already fairly clear in the spec. The other fields are passed through as is from the disc. For additional information, see the two code samples AUDIO.C and AUDIO.ASM.

Q: Must we support read sub-channel info and the read upc code?

A: No. It is not necessary that these be supported. At the present time and in the forseeable future, MSCDEX will not be issuing these commands though some applications may wish to.

Read Sub-Channel Information
At the present time, nobody is using channels P or R through W. The read sub-channel command was added to provide a standard way to specify access to these channels in the event that they are used and to specify in one way or another access to all information on a CD-ROM. The error detection or correction for information in these channels is not as good as it is for data returned from READ commands.

Read UPC Code
This command is conceivably much more useful. It is advised that it be

supported so that application software can exactly identify the CD-ROM in the drive. This may be a consideration for audio software that wishes to try to identify inserted audio discs (if the UPC code is present) or for software that is specifically tied to a version of a CD-ROM. Unfortunately, the standard does not specify a specific location for this information so that unless the hardware reads it on disc initialization as we recommend, the device driver must poll the q-channel and hope that it locates it. See also the sample code in AUDIO.ASM.

Q: My driver seems to work ok except that whenever I connect to a sub-directory and do a directory, I am suddenly back in the root directory again. What's going wrong?

A: What is most likely happening is the driver is returning an incorrect value for MEDIA CHECK and MSCDEX thinks that the disc is changing all the time. When this happens, MSCDEX rereads the volume descriptors and pathtable and reinitializes what it knows about the disc and changes the current working directory back to root as if the drawer had been opened, the disc removed, and then reinserted. This will be accompanied with a larger amount of disc activity than one would expect for a simple directory scan. Fixing the driver to return the correct value when asked for a media check will correct this behavior.

Q: What is the best way for my application to know if the disc has changed since it was last accessed?

A: Use the DOS function find first and look for the volume id. When the disc has been read and MSCDEX has already initialized the internal information it keeps for each disc, this is a relatively inexpensive operation. The information is in memory and the disc does not have to be touched, so checking the volume id is very quick. Only if the disc has been changed does the disc have to be touched. This operation takes considerably longer than if the disc was not changed but even so, this has to be done anyway because MSCDEX has to read and initialize what it knows about the new disc so it can report the volume id correctly so the application can know if the disc in the drive is the one that it is looking for.

Q: When I do a directory, the first couple filenames are either duplicated or they are random characters. What might cause this?

A: The problem comes from having the incorrect bytes in the file identifier field for the first two directory entries. The first directory entry in each directory file is supposed begin with a copy of the directory record for that directory file followed by a copy of the directory record for the parent directory (also known as '.' and '..' on Unix or MS-DOS). The filename or directory identifier is supposed to be 1 byte long

Q: When I do a directory, the first couple filenames are either duplicated or they are random characters. What might cause this?

A: The problem comes from having the incorrect bytes in the file identifier field for the first two directory entries. The first directory entry in each directory file is supposed begin with a copy of the directory record for that directory file followed by a copy of the directory record for the parent directory (also known as '.' and '..' on Unix or MS-DOS). The filename or directory identifier is supposed to be 1 byte long and the contents are supposed to be 0 for the first directory entry and 1 for the second directory entry. This is discussed in clause 6.8.2.2 of the ECMA standard or the ISO-9660 proposal. Several places on the disc in question, you have a 1 where there should be a 0 and in one directory, the file identifier consists of 0x8A which is why DIR in that directory begins with an "e". Incorrectly formatted discs will not be handled by the extensions correctly. This is why it is a good idea to test your disc image using MSCDEX before you press a disc to make sure your data is formatted correctly and as MSCDEX expects it.

Q: I have a directory file that is 4Kb long but when I do a DIR in that directory, it is slower than usual and random filenames are printed out. I can tell by watching the device driver commands that MSCDEX is asking for sectors far beyond the end of the directory. I can see how this might account for the random filenames but why is it scanning so far?

A: Problems such as this result from having with multi-sector directory files that include empty sectors in the directory file. The High Sierra specification does not allow you to have empty directory sectors at the end or to have gaps in the middle. The problem stems from the fact that your directory length is too long. For example, for the disc in question, the root directory begins at sector 28 and its length is 4096 bytes but the second sector is completely blank (all 0's). This confuses MSCDEX because it does not expect to see empty sectors.

Because LEN_DR of what would be the first directory entry in sector 29 is 0, MSCDEX thinks that there are no more entries in that sector. When it reaches the end of the entries in each sector, MSCDEX rounds its offset up to the start of the next sector:

```
  offset += (SECTOR_SIZE - 1);
  offset &= ~(SECTOR_SIZE - 1);
```

Since the offset did not changed when scanning sector 29 (there were no entries in this sector to increment the offset) the above rounding algorithm does not change the offset (2048 in, 2048 out). This is why MSCDEX continues reading beyond the end of the directory file at sector 29 because the offset did not grow past the directory length. MSCDEX continues reading blank sectors (sectors 29 through 49 are all blank) until it reaches the first non-blank sector.

It looks like what you are attempting to do is implement updatable High Sierra and you want to allocate the directory space ahead of time and fill it in later as needed. The format you are using though is not valid High Sierra and also incurs the cost of reading the blank directory sectors at the end of every directory. Instead, you should record the correct High Sierra directory length in the directory length field for that directory (in this case 2Kb). What remains is finding a place to store the value which tells how many blocks have been reserved for each directory file. There are a number of places this can be done, either in system/application use fields in the directory record, in an XAR, or in a separate file either inside or outside the High Sierra directory structure. The first is the easiest approach to take.

Be aware thought that CDI may have plans to use the system use field in the directory record so you may want to investigate Philips' plans to make sure whatever scheme you use meshes well with what CDI has in mind. MSCDEX will ignore the system use or application use information recorded so you can store what you'd like in the form you like (ascii or binary). You may also want a final pass over the directory hierarchy before mastering to remove extraneous information from the directory record.

Q: I noticed that Function Request 0 Get Number of CD-ROM Drive Letters may not always return unambiguous results. Suppose I start the network first and

use one of the drive letters for a network drive (F:). When I start the
Extensions, it will begin assigning drive letters after the last used drive
letter (C: on my machine). If I have 4 CD-ROM drives on my system, they
will be assigned drive letters D:, E:, G:, and H:. Function 0 returns 4 in
BX for the number of CD-ROM drives and 3 in CX for drive letter D:
correctly. But as you can see, the CD-ROM drives do not use contiguous drive
letters so I cannot deduce from what this function returns that drive F: is
not a CD-ROM drive.

A: That is correct. This is why function 0Dh Get CD-ROM drive letters was
added. To get an unambiguous list of CD-ROM drives, use this function or use
function 0Bh CD-ROM Drive Check to tell if a drive letter is for a CD-ROM
drive.

Q: Is it possible to do an absolute read using the Extensions. I am trying
to read mode 2 (uncooked) data using Function Request 8 Absolute Read. I use
a normal device I/O to turn off error correction and perform a read but all
I get back is 2048 bytes of data instead of the full 2356 bytes. Is there
another way in Int 2F to get the data uncooked?

A: Not at present. If you want to get at the data including error correction
code, you will have to communicate directly with the device driver. The
Extensions will provide the location of the device drivers if asked.

Q: Is it possible to access a non-High Sierra disc with the Extensions using
an absolute disc read?

A: One can use either the extensions to read a non-High Sierra disc using
INT 2Fh or one can communicate directly with the device driver to do this.
The device driver itself makes no distinction between High Sierra and non-High Sierra discs so it can be used to read them although the burden of file
system translation and reading then falls on the application talking to the
driver. The INT 2Fh Absolute Read function simply packages the request to
read and sends it directly to the driver and returns the result.

Q: What we have done is, in AUTOEXEC.BAT, first loaded the MS-DOS CD-ROM
extensions and then the MS-NET software. The error message is "Redirector
already installed". The network software is then not loaded. We are using
MS-NET 1.1 in an HP product called ThinLAN. Any hints as to what they should
try next?

A: MSCDEX is a CD-ROM "redirector". It hooks into DOS the same way the
network redirector does to get requests for file access to files that are
not on local hard/floppy disks. As far as DOS is concerned, CD-ROM drives
look just like network drives. DOS passes all file accesses through the
redirector interface to the network redirector which in turn sends file
access requests out over the net. MSCDEX splices itself in front of the
network redirector and takes requests belonging to CD-ROM drives and passes
the rest to the network redirector.

The problem is that the network redirector code assumes that there will only
be one redirector installed (itself) whereas MSCDEX does not make this
assumption. If the network redirector is installed after MSCDEX (before it
in the interrupt chain), it will process all requests from DOS and never
pass any CD-ROM requests through to MSCDEX. For this reason, MSCDEX has to
be installed after the network redirector (before it in the interrupt chain)
and so MSCDEX prevents the network redirector from installing afterwards to
ensure this. Since you installed MSCDEX first, the network believes a
redirector is already installed so it does not install itself which is what
you are seeing. In order to install both, simply install your network

software first and MSCDEX second and you're set.

Q: CHKDSK, ASSIGN, and SUBST report that the CD-ROM is a network disc. Why is this?

A: From the above explanation, you understand that to DOS, the CD-ROM drives look like network drives. The programs CHKDSK, ASSIGN, and SUBST check the same fields DOS does and think the same thing. There is no way to get around this.

Q: RENAME gives error message "Duplicate file name or File not found" instead of something that makes sense such as "Access denied" or "Can't rename CD-ROM files".

A: The error message is coming from the code for RENAME and not MSCDEX. The error condition is being returned correctly but the error code returned by version 1.01 is correct according to DOS documentation. The problem seems to be that there are two error codes for access denied - 5 and a special one 65 which is error_net_access_denied which is returned by the network redirector when it has a problem. MSCDEX version 2.00 returns error code error_net_access_denied and so RENAME now reports "Access denied".