

**cd**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> cd		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		March 29, 2025	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>cd</b>	<b>1</b>
1.1	cd.doc	1
1.2	cd.device/CD_ADDCHANGEINT	1
1.3	cd.device/CD_ADDFRAMEINT	2
1.4	cd.device/CD_ATTENUATE	3
1.5	cd.device/CD_CHANGENUM	4
1.6	cd.device/CD_CHANGESTATE	4
1.7	cd.device/CD_CONFIG	5
1.8	cd.device/CD_EJECT	7
1.9	cd.device/CD_GETGEOMETRY	7
1.10	cd.device/CD_INFO	8
1.11	cd.device/CD_MOTOR	9
1.12	cd.device/CD_PAUSE	9
1.13	cd.device/CD_PLAYLSN	10
1.14	cd.device/CD_PLAYMSF	11
1.15	cd.device/CD_PLAYTRACK	12
1.16	cd.device/CD_PROTSTATUS	12
1.17	cd.device/CD_QCODELSN	13
1.18	cd.device/CD_QCODEMSF	14
1.19	cd.device/CD_READ	15
1.20	cd.device/CD_READXL	16
1.21	cd.device/CD_REMCHANGEINT	17
1.22	cd.device/CD_REMFRAMEINT	18
1.23	cd.device/CD_SEARCH	18
1.24	cd.device/CD_SEEK	19
1.25	cd.device/CD_TOCLSN	20
1.26	cd.device/CD_TOCMSF	21
1.27	cd.device/CloseDevice	22
1.28	cd.device/OpenDevice	23

# Chapter 1

## cd

### 1.1 cd.doc

```
CD_ADDCHANGEINT
CD_ADDFRAMEINT
CD_ATTENUATE
CD_CHANGENUM
CD_CHANGESTATE
CD_CONFIG
CD_EJECT
CD_GETGEOMETRY
CD_INFO
CD_MOTOR
CD_PAUSE
CD_PLAYLSN
CD_PLAYMSF
CD_PLAYTRACK
CD_PROTSTATUS
CD_QCODELSN
CD_QCODEMSF
CD_READ
CD_READXL
CD_REMCHANGEINT
CD_REMFRAMEINT
CD_SEARCH
CD_SEEK
CD_TOCLSN
CD_TOCMSF
CloseDevice()
OpenDevice()
```

### 1.2 cd.device/CD\_ADDCHANGEINT

#### NAME

CD\_ADDCHANGEINT -- add a disk change software interrupt handler.

#### FUNCTION

This command lets you add a software interrupt handler to the

disk device that gets invoked whenever a disk insertion or removal occurs.

You must pass in a properly initialized Exec Interrupt structure and be prepared to deal with disk insertions/removals immediately. The interrupt is generated by the exec Cause function, so you must preserve A6.

To set up the handler, an Interrupt structure must be initialized. This structure is supplied as the io\_Data to the CD\_ADDCHANGEINT command. The handler then gets linked into the handler chain and gets invoked whenever a disk change happens. You must eventually remove the handler before you exit.

This command only returns when the handler is removed. That is, the device holds onto the IO request until the CD\_REMCHANGEINT command is executed with that same IO request. Hence, you must use SendIO() with this command.

#### IO REQUEST INPUT

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_ADDCHANGEINT
io_Length	sizeof(struct Interrupt)
io_Data	pointer to Interrupt structure

#### IO REQUEST RESULT

io_Error	- 0 for success, or an error code as defined in <devices/cd.h>
----------	--

#### SEE ALSO

CD\_REMCHANGEINT, <devices/cd.h>, <exec/interrupts.h>, exec.library/Cause()

## 1.3 cd.device/CD\_ADDFRAMEINT

#### NAME

CD\_ADDFRAMEINT -- add a CD-frame software interrupt handler.

#### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_ADDFRAMEINT
io_Length	sizeof(struct Interrupt)
io_Data	pointer to Interrupt structure

#### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
----------	--

#### FUNCTION

This command lets you add a software interrupt handler to the disk device that gets invoked whenever a new frame is encountered while CD audio is being played.

---

You must pass in a properly initialized Exec Interrupt structure and be prepared to deal with frame interrupts immediately. The interrupt is generated by the exec Cause function, so you must preserve A6.

To set up the handler, an Interrupt structure must be initialized. This structure is supplied in io\_Data of the CD\_ADDFRAMEINT command. The handler then gets linked into the handler chain and gets invoked whenever a frame event occurs. You must eventually remove the handler before you exit.

This command only returns when the handler is removed. That is, the device holds onto the IO request until the CD\_REMFRAMEINT command is executed with that same IO request. Hence, you must use SendIO() with this command.

#### NOTES

The interrupt handler can be added before or after a play command is sent. Interrupts will only be generated while CD audio is playing. Interrupts will not be generated when audio is paused.

#### SEE ALSO

CD\_REMFRAMEINT, <devices/cd.h>, <exec/interrupts.h>, exec.library/Cause()

## 1.4 cd.device/CD\_ATTENUATE

#### NAME

CD\_ATTENUATE -- Attenuate CD audio volume (immediately or gradually)

#### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_ATTENUATE
io_Data	NULL
io_Length	duration of volume fade in frames
io_Offset	target volume level (0 - 0x7FFF) (-1 = status only)

#### RESULTS

io_Error	Returns an error if drive does not support attenuation
io_Actual	current volume level (fade may be monitored)

#### FUNCTION

This command will ramp the CD audio volume up or down from its current value to the value contained in io\_Offset. The range is 0 (silence) to 0x7FFF (full volume). If -1 is specified as the target, the attenuation will not be modified; the current attenuation value will be returned in io\_Actual.

io\_Length contains the duration of the fade. In seconds, this is io\_Length divided by the current frame rate (usually 75).

Note that this command returns before the fade has completed. Thus, once started, a fade cannot be aborted. You can, however, send a new CD\_ATTENUATE command, which will immediately override any fade

currently in progress. An `io_Length` of zero means attenuate immediately.

If a gradual attenuation command is sent before the play command, the fade will begin as soon as the play command is sent.

#### EXAMPLE

#### NOTES

This command has no effect on Amiga audio volume, only CD audio.

If the drive does not support volume attenuation, but does support mute, a value of under \$0800 should be considered mute, and equal to or above should be full volume. If chunky attenuation is supported, the drive should do the best it can. If the drive does not support volume attenuation at all, an error should be returned. Even if only mute is supported, if gradual attenuation is requested, the device should still emulate the fade command and mute based on the \$0800 boundary.

#### BUGS

#### SEE ALSO

`CD_INFO`

## 1.5 `cd.device/CD_CHANGENUM`

#### NAME

`CD_CHANGENUM` -- return the current value of the disk-change counter.

#### FUNCTION

This command returns the current value of the disk-change counter. The disk change counter is incremented each time a disk is inserted or removed from the cd unit.

#### IO REQUEST INPUT

<code>io_Device</code>	preset by the call to <code>OpenDevice()</code>
<code>io_Unit</code>	preset by the call to <code>OpenDevice()</code>
<code>io_Command</code>	<code>CD_CHANGENUM</code>

#### IO REQUEST RESULT

<code>io_Error</code>	- 0 for success, or an error code as defined in <code>&lt;devices/cd.h&gt;</code>
<code>io_Actual</code>	- if <code>io_Error</code> is 0, this contains the current value of the disk-change counter.

## 1.6 `cd.device/CD_CHANGESTATE`

#### NAME

`CD_CHANGESTATE` -- check if a "valid" disk is currently in a drive.

#### FUNCTION

---

This command checks to see if there is a "valid" disk in a drive.

#### IO REQUEST INPUT

io\_Device            preset by the call to OpenDevice()  
 io\_Unit             preset by the call to OpenDevice()  
 io\_Command          CD\_CHANGESTATE

#### IO REQUEST RESULT

io\_Error - 0 for success, or an error code as defined in  
           <devices/cd.h>  
 io\_Actual - 0 means there is a disk while anything else indicates  
           there is no disk.

#### NOTES

A "valid" disk is a disk with a readable table of contents.

## 1.7 cd.device/CD\_CONFIG

#### NAME

CD\_CONFIG -- Set drive preferences

#### IO REQUEST

io\_Device            preset by the call to OpenDevice()  
 io\_Unit             preset by the call to OpenDevice()  
 io\_Command          CD\_CONFIG  
 io\_Data             pointer to first entry of TagList  
 io\_Length           0

#### RESULTS

io\_Error            0 for success, or an error code as defined in  
           <devices/cd.h>

#### FUNCTION

This command sets one or more of the configuration items.  
 The configuration items are:

TAGCD_PLAYSPEED	Default: 75
TAGCD_READSPEED	Default: 75 (do not count on this)
TAGCD_READXLSPEED	Default: 75
TAGCD_SECTORSIZE	Default: 2048
TAGCD_XLECC	Default: 1 (on)
TAGCD_EJECTRESET	Default: can be 0 (off) or 1 (on)

The speed settings are described in the number of frames (sectors) per second. All CD-ROM drives are capable of the 75 frames/second rate. Some drives are capable of 150 frames/second, and some even more. To determine the maximum frame rate of the drive, use the CD\_INFO command. Valid values for caddyless Commodore CD-ROM drives are 75 and 150 (normal speed and double speed). All other values are invalid. You should always make sure the drive is capable of the configuration you are requesting by either using the CD\_INFO command, and/or by checking for an error condition after submitting your request.

There are three different types of CD-ROM sectors. Mode 1 sectors



(2048 bytes), mode 2 form 1 sectors (2048 bytes), and mode 2 form 2 sectors (2328 bytes). Normally, disks are encoded in Mode 1 format. Mode 2 form 1 is basically the same as mode 1; however, the mode 2 form 2 sector format contains no CD-ROM error correction information. In order to read information encoded in this sector format, the drive's sector size must be configured to 2328 byte sectors.

Error correction (ECC) of the READXL command can be turned off or on with this command. Error correction can be implemented in either hardware or software (depending on the CD-ROM drive). When ECC is implemented in software, CPU usage can become bursty. Errors rarely occur on CDs unless they have numerous scratches, but when they do occur, they will cause a loss of CPU bandwidth. When ECC is implemented in hardware, no CPU bandwidth is lost -- in this case, ECC will always be on no matter how you configure the drive because it is free. The READXL command is used primarily for displaying movie-like data. In this case, speed is essential and data integrity is not; however, if the CPU is not being utilized during an XL animation there is no need to disable ECC (since the bandwidth is there to be used). The only time ECC should be disabled is when you are doing intense calculations in the background of a READXL command, AND your program is time-critical. Do not forget to change this back when you are done!

To make the computer reset when a CD is ejected (for an application that does not exit), use the TAGCD\_EJECTRESET tag. When possible, titles should be able to exit cleanly back to Workbench. Error conditions should be monitored when doing disk I/O.

#### EXAMPLE

```
/* Configure ReadXL for double-speed reading and turn off ECC when */
/* the ReadXL command is used.                                     */

struct TagItem ConfigList[] = {

    { TAGCD_READXLSPEED, 150 },
    { TAGCD_XLECC,       0   },
    { TAG_END,           0   }
};

ior->io_Command = CD_CONFIG;
ior->io_Data     = (APTR)&ConfigList;
ior->io_Length   = 0;
DoIO(ior);

if (ior->io_Error) printf("Could not be configured\n");
```

#### NOTES

Setting the configuration will not modify the behavior of a read or play command already in progress.

This can be a very dangerous command. If for instance you set TAGCD\_SECTORSIZE to 2328, you will no longer be able to read any data encoded at 2048 byte sectors (e.g. the file system will not be able to read the disk anymore). After you read any data stored with this sector format, you should immediately configure back to the original default value (even if the read failed -- the disk could

be removed in the middle of your read). You should NEVER use this command if you are not the exclusive owner of your disk.

#### BUGS

TAG\_IGNORE, TAG\_MORE, and TAG\_SKIP do not work. Do not use these.

When switching speeds from single to double (or double to single), If the drive is prefetching in single-speed the data you are going to use in double-speed, the drive will not switch to double-speed (and visa versa). To avoid this problem, switch to the desired speed, begin reading at least 4k into the data (just read two bytes), then begin reading at the beginning. This will force the prefetch buffer to clear and issue a new read command with the desired speed. (Fixed in 40.24).

#### SEE ALSO

CD\_INFO, <utility/tagitem.h>

## 1.8 cd.device/CD\_EJECT

#### NAME

CD\_EJECT -- Open or close the CD's drive door

#### IO REQUEST

io_Command	CD_EJECT
io_Data	NULL
io_Length	requested state of drive door (0 == close, 1 == open)
io_Offset	0

#### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
io_Actual	previous state of drive door

#### FUNCTION

This command causes the CD-ROM drive's door to open or close. The desired state of the drive door is placed in io\_Length. The previous state of the drive door is returned in io\_Actual.

#### EXAMPLE

#### NOTES

#### BUGS

#### SEE ALSO

## 1.9 cd.device/CD\_GETGEOMETRY

#### NAME

CD\_GETGEOMETRY -- return the geometry of the drive.

---



```
}
```

NOTES

BUGS

SEE ALSO

<devices/cd.h>

## 1.11 cd.device/CD\_MOTOR

NAME

CD\_MOTOR -- control the on/off state of a drive motor.

FUNCTION

This command gives control over the spindle motor. The motor may be turned on or off.

If the motor is just being turned on, the device will delay the proper amount of time to allow the drive to come up to speed. Turning the motor on or off manually is not necessary, the device does this automatically if it receives a request when the motor is off.

IO REQUEST INPUT

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_MOTOR
io_Length	the requested state of the motor, 0 to turn the motor off, and 1 to turn the motor on.

IO REQUEST RESULT

io_Error	- 0 for success, or an error code as defined in <devices/cd.h>
io_Actual	- if io_Error is 0 this contains the previous state of the drive motor.

## 1.12 cd.device/CD\_PAUSE

NAME

CD\_PAUSE -- Pause or unPause play command.

IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_PAUSE
io_Data	NULL
io_Length	pausemode : 1 = pause play; 0 = do not pause play;
io_Offset	0

RESULTS

io_Actual	- if io_Error is 0, this contains the previous pause state.
-----------	---

---

## FUNCTION

This command will place the CD in, or take the CD out of pause mode. The desired pause state is placed in `io_Length`. This command only effects play commands. When the audio is playing and the pausemode is set, this command will immediately pause the audio output suspending the play command until the play is unpaused. When audio is not playing and the pausemode is set, this command will set the pause mode (having no immediate effect). When a play command is submitted, the laser will seek to the appropriate position and pause at that spot. The play command will be suspended until the play is unpaused (or the play is aborted).

## EXAMPLE

## NOTES

## BUGS

## SEE ALSO

## 1.13 cd.device/CD\_PLAYLSN

## NAME

`CD_PLAYLSN` -- Play a selected portion of CD audio (LSN form).

## IO REQUEST

<code>io_Device</code>	preset by the call to <code>OpenDevice()</code>
<code>io_Unit</code>	preset by the call to <code>OpenDevice()</code>
<code>io_Command</code>	<code>CD_PLAYLSN</code>
<code>io_Data</code>	<code>NULL</code>
<code>io_Length</code>	length of play
<code>io_Offset</code>	starting position

## RESULTS

<code>io_Error</code>	0 for success, or an error code as defined in <code>&lt;devices/cd.h&gt;</code>
-----------------------	---

## FUNCTION

This command causes the drive to start playing CD audio from the specified position until the specified length has passed.

`io_Offset` specifies the starting position. `io_Length` contains the amount of time to play. All data is specified in LSN format.

A `DoIO()` will not return until the requested number of sectors have been played. A `SendIO()` will return as soon as the PLAY has been started. At this time other commands can be sent (like `CD_PAUSE`). To stop a play before the specified length has been reached, use `AbortIO()`.

## EXAMPLE

```
/* Play two minutes, ten seconds of audio starting at 20 minutes, */
/* 58 seconds, and 10 frames.                                     */

ior->io_Command = CD_PLAYLSN;    /* Play CD audio                */
```

```

ior->io_Offset = 94360;          /* 20*(60*75) + 58*75 + 10 */
ior->io_Length  = 9750;          /* 02*(60*75) + 10*75 + 00 */
DoIO (ior);

```

NOTES

BUGS

SEE ALSO

CD\_PLAYTRACK, CD\_PAUSE, CD\_SEARCH, CD\_ATTENUATE

## 1.14 cd.device/CD\_PLAYMSF

NAME

CD\_PLAYMSF -- Play a selected portion of CD audio (MSF form).

IO REQUEST

```

io_Device      preset by the call to OpenDevice()
io_Unit        preset by the call to OpenDevice()
io_Command     CD_PLAYMSF
io_Data        NULL
io_Length      length of play
io_Offset      starting position

```

RESULTS

```

io_Error       0 for success, or an error code as defined in
               <devices/cd.h>

```

FUNCTION

This command causes the drive to start playing CD audio from the specified position until the specified length has passed.

io\_Offset specifies the starting position. io\_Length contains the amount of time to play. All data is specified in MSF format.

A DoIO() will not return until the requested number of sectors have been played. A SendIO() will return as soon as the PLAY has been started. At this time other commands can be sent (like CD\_PAUSE). To stop a play before the specified length has been reached, use AbortIO().

EXAMPLE

```

/* Play two minutes, ten seconds of audio starting at 20 minutes, */
/* 58 seconds, and 10 frames.                                     */

ior->io_Command = CD_PLAYMSF; /* Play CD audio          */
ior->io_Offset  = 0x00143A0A; /* $14=20, $3A=58, $0A=10 */
ior->io_Length  = 0x00020A00; /* $02=02, $0A=10, $00=00 */
DoIO (ior);

```

NOTES

BUGS

SEE ALSO

CD\_PLAYTRACK, CD\_PAUSE, CD\_SEARCH, CD\_ATTENUATE

## 1.15 cd.device/CD\_PLAYTRACK

### NAME

CD\_PLAYTRACK -- Play one or more tracks of CD audio.

### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_PLAYTRACK
io_Data	NULL
io_Length	number of tracks to play
io_Offset	start playing at beginning of this track

### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
----------	---

### FUNCTION

This command causes the drive to play the specified audio track(s).  
The command will return when the audio has completed.

io\_Offset specifies the track number (starting from 1).

io\_Length specifies the number of tracks to play (0 is invalid).

### EXAMPLE

```
ior->io_Command = CD_PLAYTRACK;    /* Play audio tracks    */
ior->io_Offset  = STARTTRACK;      /* Start with this track */
ior->io_Length  = 3;               /* Play three tracks     */
DoIO(ior);
```

### NOTES

PLAY commands are asynchronous with many other CD commands.  
Using a separate I/O request, other commands can be sent to the device  
that can change the behavior of the PLAY command.

### BUGS

### SEE ALSO

CD\_PLAYMSF, CD\_PLAYLSN, CD\_PAUSE, CD\_SEARCH, CD\_ATTENUATE

## 1.16 cd.device/CD\_PROTSTATUS

### NAME

CD\_PROTSTATUS -- return whether the current disk is write-protected.

### FUNCTION

This command is used to determine whether the current disk is  
write-protected. Currently, this function always returns write-

protected status. If write-once CDs are made available at some point, this may change.

#### IO REQUEST INPUT

io\_Device            preset by the call to OpenDevice()  
 io\_Unit             preset by the call to OpenDevice()  
 io\_Command          CD\_PROTSTATUS

#### IO REQUEST RESULT

io\_Error - 0 for success, or an error code as defined in  
           <devices/cd.h>  
 io\_Actual - 0 means the disk is NOT write-protected, while any other  
           value indicates it is.

## 1.17 cd.device/CD\_QCODELSN

#### NAME

CD\_QCODELSN -- Report current disk position.

#### IO REQUEST

io\_Device            preset by the call to OpenDevice()  
 io\_Unit             preset by the call to OpenDevice()  
 io\_Command          CD\_QCODELSN  
 io\_Data             pointer to QCode structure  
 io\_Length           0 - MUST be zero (for future compatability)

#### RESULTS

io\_Error            0 for success, or an error code as defined in  
           <devices/cd.h>

#### FUNCTION

This command reports current subcode Q channel time information. This command only returns data when CD Audio is playing (or paused). At any other time, an error is returned. The Q-Code packet consists of:

```
struct QCode {
    UBYTE            CtlAdr;            /* Data type / QCode type            */
    UBYTE            Track;            /* Track number                       */
    UBYTE            Index;            /* Track subindex number               */
    UBYTE            Zero;            /* The "Zero" byte of Q-Code packet */
    union LSNMSF TrackPosition; /* Position from start of track       */
    union LSNMSF DiskPosition; /* Position from start of disk        */
};
```

#### EXAMPLE

```
struct QCode qcode;

ior->io_Command = CD_QCODELSN; /* Retrieve TOC information */
ior->io_Length = 0;            /* MUST be zero               */
ior->io_Data = (APTR)qcode; /* Here's where we want it */
DoIO (ior);

if (!ior->io_Error) {            /* Command succeeded            */
```



```
printf("Current position is: %ld\n", qcode.DiskPosition.LSN);
}
```

#### NOTES

This function may not return immediately. It may take several frames to pass by before a valid Q-Code packet can be returned. Use SendIO() and CheckIO() if response time is critical, and the information is not.

#### BUGS

#### SEE ALSO

CD\_PLAYMSF, CD\_PLAYLSN, CD\_PLAYTRACK, <devices/cd.h>

## 1.18 cd.device/CD\_QCODEMSF

#### NAME

CD\_QCODEMSF -- Report current disk position.

#### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_QCODEMSF
io_Data	pointer to QCode structure
io_Length	0 - MUST be zero (for future compatability)

#### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
----------	--

#### FUNCTION

This command reports current subcode Q channel time information. This command only returns data when CD Audio is playing (or paused). At any other time, an error is returned. The Q-Code packet consists of:

```
struct QCode {
    UBYTE      CtlAdr;          /* Data type / QCode type      */
    UBYTE      Track;           /* Track number                 */
    UBYTE      Index;           /* Track subindex number       */
    UBYTE      Zero;            /* The "Zero" byte of Q-Code packet */
    union LSNMSF TrackPosition; /* Position from start of track */
    union LSNMSF DiskPosition; /* Position from start of disk  */
};
```

#### EXAMPLE

```
struct QCode qcode;

ior->io_Command = CD_QCODEMSF; /* Retrieve TOC information */
ior->io_Length   = 0;           /* MUST be zero             */
ior->io_Data     = (APTR)qcode; /* Here's where we want it */
DoIO (ior);
```

```
if (!ior->io_Error) {                /* Command succeeded      */

    printf("Current position is: %02d:%02d:%02d\n",
           qcode.DiskPosition.MSF.Minute,
           qcode.DiskPosition.MSF.Second,
           qcode.DiskPosition.MSF.Frame);
}
```

#### NOTES

This function may not return immediately. It may take several frames to pass by before a valid Q-Code packet can be returned. Use SendIO() and CheckIO() if response time is critical, and the information is not.

#### BUGS

#### SEE ALSO

CD\_PLAYMSF, CD\_PLAYLSN, CD\_PLAYTRACK, <devices/cd.h>

## 1.19 cd.device/CD\_READ

#### NAME

CD\_READ -- read data from disk.

#### FUNCTION

Reads data from the CD into memory. Data may be accessed on WORD boundaries (you are not restricted to sector boundaries as with normal disk devices). Data lengths can also be described in WORD amounts.

#### IO REQUEST INPUT

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_READ
io_Data	pointer to the buffer where the data should be put
io_Length	number of bytes to read, must be a WORD multiple.
io_Offset	byte offset from the start of the disk describing where to read data from, must be a WORD multiple.

#### IO REQUEST RESULT

io_Error	- 0 for success, or an error code as defined in <devices/cd.h>
io_Actual	- if io_Error is 0, number of bytes actually transferred

#### NOTES

If an error occurs when attempting a CD\_READ, the software will retry up to 10 times before giving up on the request. If the drive is in double-speed and an error occurs, the software will retry once more in double-speed, and if this fails, will retry the next 9 times in single-speed.

#### SEE ALSO

CD\_READXL

---

## 1.20 cd.device/CD\_READXL

### NAME

CD\_READXL -- Read from CD-ROM into memory via transfer list.

### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_READXL
io_Data	pointer to transfer list (i.e. struct List *).
io_Length	maximum transfer length (WORD multiple) or 0.
io_Offset	byte offset from the start of the disk describing where to read data from, must be a WORD multiple.

### RESULTS

io_Error	0 for success, or an error code as described in <devices/cd.h>
io_Actual	if io_Error is 0, number of bytes actually transferred

### FUNCTION

This command starts reading data off the disk at the specified location and deposits it into memory according to the nodes in a transfer list. The pointer to the list of transfer nodes is placed in io\_Data. If you have a non-circular transfer list, simply set io\_Length to 0 (0 is special and means ignore io\_Length) -- your transfer will end when your transfer list has been exhausted. If you have a circular transfer list, the list will never end. In this case, the transfer will terminate when io\_Length bytes have been transferred.

The fields in the CDXL node structure are:

```
struct CDXL {
    struct MinNode Node;          /* double linkage          */
    char *Buffer;                 /* data destination        */
    LONG Length;                  /* must be even # bytes    */
    LONG Actual;                  /* bytes transferred       */
    APTR IntData;                 /* interrupt server data segment */
    VOID (*IntCode)();            /* interrupt server code entry */
};
```

The philosophy here is that you set up the buffers you want filled, create CDXL nodes describing the locations and sizes of these buffers, link all the nodes together in the order that you'd like (even make a circular list for animations), and execute the command. The data will be streamed into the appropriate buffers until the list has been exhausted, an entry with a Length of zero is encountered, io\_Length bytes have been transferred (if io\_Length is non-zero), or the command is aborted with AbortIO().

If you fill in the (\*IntCode)() field with a pointer to an interrupt routine, your routine will be called when the transfer for the node is complete. Your code will be called before the driver proceeds to the next node. The interrupt should follow the same rules as standard interrupts (see AddIntServer of Exec autodocs). Register A2 will

contain a pointer to the node just completed. You may manipulate the list from within the interrupt. Your code must be brief (this is an interrupt). When returning from this interrupt, D0 should be cleared and an RTS instruction should be used to return.

Servers are called with the following register conventions:

```
D0 - scratch
D1 - scratch

A0 - scratch
A1 - server is_Data pointer (scratch)
A2 - pointer to CDXL node just completed

A5 - jump vector register (scratch)

all other registers must be preserved
```

#### EXAMPLE

#### NOTES

Try to make sure that small buffers are not overused. Each time a node is completed, an interrupt is generated. If you find that your computer is acting sluggish, or the CD\_READXL command is aborting, you are probably generating too many interrupts. It is not efficient to have more than a few of these interrupts generated within a vertical blank.

Unlike the READ command, the READXL command will not retry a sector if there is an error. Since the READXL command's purpose is primarily for animations, data streaming is considered more important than the data itself. An error will be returned in io\_Error if a data error did occur. This command will never drop to a lower speed in the event of an error.

#### BUGS

#### SEE ALSO

CMD\_READ, CD\_SEEK, Autodocs - AddIntServer

## 1.21 cd.device/CD\_REMCHANGEINT

#### NAME

CD\_REMCHANGEINT -- remove a disk change software interrupt handler.

#### FUNCTION

This command removes a disk change software interrupt added by a previous use of CD\_ADDCHANGEINT.

#### IO REQUEST INPUT

The same IO request used for CD\_ADDCHANGEINT.

```
io_Device      preset by the call to OpenDevice()
io_Unit        preset by the call to OpenDevice()
io_Command     CD_REMCHANGEINT
```

io\_Length            sizeof(struct Interrupt)  
 io\_Data             pointer to Interrupt structure  
  
 IO REQUEST RESULT  
   io\_Error - 0 for success, or an error code as defined in  
             <devices/cd.h>  
  
 SEE ALSO  
   CD\_ADDCHANGEINT, <devices/cd.h>

## 1.22 cd.device/CD\_REMFRAMEINT

NAME  
   CD\_REMFRAMEINT -- remove a CD-frame interrupt handler.  
  
 IO REQUEST  
   The same IO request used for CD\_ADDFRAMEINT.  
  
   io\_Device          preset by the call to OpenDevice()  
   io\_Unit            preset by the call to OpenDevice()  
   io\_Command         CD\_REMFRAMEINT  
   io\_Length          sizeof(struct Interrupt)  
   io\_Data            pointer to Interrupt structure  
  
 RESULTS  
   io\_Error          0 for success, or an error code as defined in  
                      <devices/cd.h>  
  
 FUNCTION  
   This command removes a CD-frame software interrupt added  
   by a previous use of CD\_ADDFRAMEINT.  
  
 BUGS  
  
 SEE ALSO  
   CD\_ADDFRAMEINT, <devices/cd.h>

## 1.23 cd.device/CD\_SEARCH

NAME  
   CD\_SEARCH -- configure the mode in which PLAY commands play  
  
 IO REQUEST  
   io\_Command         CD\_SEARCH  
   io\_Data            NULL  
   io\_Length          searchmode  
   io\_Offset          0  
  
 RESULTS  
   io\_Actual - if io\_Error is 0, this contains the previous search mode.  
  
 FUNCTION

---

This command causes a play command to play in fast-forward, fast-reverse, or normal play mode. These modes are defined as:

CDMODE_NORMAL	0	Normal play (current speed setting)
CDMODE_FFWD	1	Play in fast forward mode
CDMODE_FREV	2	Play in fast reverse mode

The search mode can be set before the play command is sent, or during a play. If CD\_SEARCH is sent before a play command is sent, the mode is set and the command immediately returns. If the mode is set to REV mode, when the play command is sent the play will begin at the requested end position and work backwards to the start position.

If CD\_SEARCH is sent during a play, the play will automatically switch to the desired mode and continue playing until the original play command is completed. If REV mode is set and the beginning of the play is encountered before switching back to forward play, the play command will terminate with no error.

#### EXAMPLE

```
/* Search in fast forward mode. */
ior->io_Command = CD_SEARCH;
ior->io_Data     = NULL;
ior->io_Offset   = 0;
ior->io_Length   = CDMODE_FFWD;
DoIO(ior);
```

#### NOTES

#### BUGS

#### SEE ALSO

## 1.24 cd.device/CD\_SEEK

#### NAME

CD\_SEEK -- position laser at specified location.

#### FUNCTION

CD\_SEEK moves the laser to the approximate position specified. The io\_Offset field should be set to the offset to which the head is to be positioned.

#### IO REQUEST INPUT

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_SEEK
io_Offset	position where head is to be moved (always LSN format)

#### IO REQUEST RESULT

io_Error	- 0 for success, or an error code as defined in <devices/cd.h>
----------	--

## 1.25 cd.device/CD\_TOCLSN

### NAME

CD\_TOCLSN -- Return table of contents information from CD (LSN form).

### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_TOCLSN
io_Data	pointer to array where TOC is to be stored
io_Length	number of CDTOC entries to be fetched
io_Offset	entry to begin at (entry 0 is summary information)

### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
io_Actual	Actual number of entries copied

### FUNCTION

This command returns the table of contents of the disk currently in the drive. The table of contents consists of up to 100 entries. Entry zero is summary information describing the number of tracks and the total number of minutes on the disk. Entries 1 through N contain information about each individual track. All position information will be in LSN format.

The io\_Data field points to an array of CDTOC structures to receive the TOC data.

The io\_Length field specifies the total number of entries to be fetched. The array pointed to by io\_Data must be at least this many elements in size.

The io\_Offset field specifies the entry number at which to start copying TOC data into \*io\_Data.

Entry zero (the summary entry) contains the following:

```
struct TOCSummary {
    UBYTE      FirstTrack;    /* First track on disk (always 1) */
    UBYTE      LastTrack;     /* Last track on disk */
    union LSNMSF LeadOut;     /* Beginning of lead-out track */
};
```

Track entries (entries 1 through number of tracks) contain:

```
struct TOCEntry {
    UBYTE      CtlAdr;        /* Q-Code info */
    UBYTE      Track;         /* Track number */
    union LSNMSF Position;    /* Start position of this track */
};
```

CDTOC is described as a union between these two structures:

```
union CDTOC {
    struct TOCSummary Summary; /* First entry is summary info. */
    struct TOCEntry Entry; /* Entries 1-N are track entries */
};
```

#### EXAMPLE

```
union CDTOC toccarray[100];

ior->io_Command = CD_TOCLSN; /* Retrieve TOC information */
ior->io_Offset = 0; /* Start with summary info */
ior->io_Length = 100; /* Max 99 tracks + summary */
ior->io_Data = (APTR)toccarray; /* Here's where we want it */
DoIO (ior);

if (!ior->io_Error) { /* Command succeeded */

    firsttrack = toccarray[0].Summary.FirstTrack;
    lasttrack = toccarray[0].Summary.LastTrack;
    totalsectors = toccarray[0].Summary.LeadOut.LSN -
                    toccarray[1].Entry.Position.LSN;
}
```

#### NOTES

In the above example, the amount of data on the disk is calculated as being equal to the location of the lead-out track minus the start of the first track (which is never zero).

#### BUGS

#### SEE ALSO

## 1.26 cd.device/CD\_TOCMSF

#### NAME

CD\_TOCMSF -- Return table of contents information from CD (MSF form).

#### IO REQUEST

io_Device	preset by the call to OpenDevice()
io_Unit	preset by the call to OpenDevice()
io_Command	CD_TOCMSF
io_Data	pointer to array where TOC is to be stored
io_Length	number of CDTOC entries to be fetched
io_Offset	entry to begin at (entry 0 is summary information)

#### RESULTS

io_Error	0 for success, or an error code as defined in <devices/cd.h>
io_Actual	Actual number of entries copied

#### FUNCTION

This command returns the table of contents of the disk currently in



the drive. The table of contents consists of up to 100 entries. Entry zero is summary information describing the number of tracks and the total number of minutes on the disk. Entries 1 through N contain information about each individual track. All position information will be in MSF format.

The `io_Data` field points to an array of CDTOC structures to receive the TOC data.

The `io_Length` field specifies the total number of entries to be fetched. The array pointed to by `io_Data` must be at least this many elements in size.

The `io_Offset` field specifies the entry number at which to start copying TOC data into `*io_Data`.

Entry zero (the summary entry) contains the following:

```
struct TOCSummary {
    UBYTE      FirstTrack;    /* First track on disk (always 1) */
    UBYTE      LastTrack;     /* Last track on disk */
    union LSNMSF LeadOut;     /* Beginning of lead-out track */
};
```

Track entries (entries 1 through number of tracks) contain:

```
struct TOCEntry {
    UBYTE      CtlAdr;        /* Q-Code info */
    UBYTE      Track;         /* Track number */
    union LSNMSF Position;    /* Start position of this track */
};
```

CDTOC is described as a union between these two structures:

```
union CDTOC {
    struct TOCSummary Summary; /* First entry is summary info. */
    struct TOCEntry   Entry;   /* Entries 1-N are track entries */
};
```

EXAMPLE

NOTES

BUGS

SEE ALSO

## 1.27 cd.device/CloseDevice

NAME

CloseDevice - terminate access to the CD

## SYNOPSIS

```
CloseDevice(IORequest);  
           A1
```

## FUNCTION

This function will terminate access to the unit opened with `OpenDevice()`.

## INPUTS

`iORequest` - pointer to a struct(`IOStdReq`)

## RESULTS

## NOTES

## SEE ALSO

`OpenDevice()`

## 1.28 cd.device/OpenDevice

## NAME

`OpenDevice` - Open a CD unit for access

## SYNOPSIS

```
error = OpenDevice("cd.device", UnitNumber, IORequest, flags);  
D0           A0           D0           A1           D1
```

## FUNCTION

Opens the `cd.device` and creates an `IORequest` for use in accessing the CD.

## INPUTS

`UnitNumber` - Normally zero; however, this is described as:  
              Ones digit      = Unit (SCSI unit number)  
              Tens digit      = LUN (disk within disk changer)  
              Hundreds digit = Card number (SCSI card)  
              Thousands digit = Reserved (must be zero)  
`IORequest` - Pointer to a struct(`IOStdReq`)  
`flags`      - Should be zero.

## RESULTS

`error`      0 = success, otherwise this is an error.

## NOTES

## SEE ALSO

`CloseDevice()`

---