Multimedia

Technical Notes

CD-ROM Design and Optimization

Matt Saettler
Microsoft Multimedia
One Microsoft Way
Redmond, WA 98052-6399 USA

10/92 Revision 2.03

Overview

This tech note details how to design and optimize an application for use and delivery on CD-ROM using the Microsoft® MS-DOS® CD-ROM Extensions (MSCDEX).

For the purposes of this document, the references to High Sierra pertain both to the High Sierra standard and the ISO 9660 standard except where specifically noted.

This technical note contains information targeted to a couple types of readers:

- Application developers (C or High-Level Tool)
- Low-level application developers (assembler)
- Multimedia producers

Where to Look for Information

The sample code described in this technical note are available in this kit.

Current versions of this document as well as other technical update and technical notes and sample code are available from the sources listed in the *Multimedia Document Overview*, included in this kit.

Versions of this Document

This document is continually being updated and expanded. I am very interested in getting your feedback on specific issues you would like to see discussed or shows in sample code.

Please refer to the document , which is included in this kit, for lists of other documents and sample code.

Part Numbers

This document is contained in the following kit #s:

098-39214 Kit Win 3.1 Multimedia Technotes

Intended Audience

This document should be read by "multimedia producers" (as defined in the Multimedia Authoring Guide) as well as programmers using all types of tools. You should read this document to ensure that you have the current versions of the technotes and updates.

Readers should also read the Authoring Tools Guide section of the Multimedia Development Kit. This section is also contained in the a from MS-Press (ISBN 1-55615-391-0).

Designing an Application for CD-ROM

This sections describes data and file design optimizations and rules for CD-ROM.

CD-ROM is Just Like a Big Floppy

MSCDEX makes a CD-ROM appear as just another drive letter under MS-DOS. This makes using a CD-ROM from an application is just like reading from a big (really huge!) floppy. The seek time usually averages about 300ms with transfer rates being a fast (for floppy) 150K/second. The CD-ROM is a removable media, meaning that the user can eject it just when an application wants to read from it. This means that error recovery on file operations is very important.

Reading Data While Playing Audio

Data cannot be read while CD-Audio is playing. Any attempts to read will result in a critical error (a type of MS-DOS error that results in the Abort, Retry, Ignore message). Keep this in mind when designing how the application will use data.

You can either design the application so that all data is read before the CD-Audio starts playing, or the application can have its own internal 'read' routines stop and re-start the CD-Audio manually. Both require specific design work-arounds to overcome the problem.

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 '_'. <u>All other characters are invalid</u>. Note that the letters 'a' through 'z' are not included so that High Sierra file names are not case sensitive. Under MS-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 path names 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. The following example shows valid and invalid path names:

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

Length of Path

The High Sierra format allows for the entire path name 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 than MS-DOS.

Unfortunately, a CD-ROM may potentially contain a path name that is much larger than 64 characters long. This is not a concern here but is discussed in the section "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.

Unique Volume Identifiers

It is highly recommended that the volume identifier be unique. MSCDEX uses the volume identifier to do volume tracking and to double-check if the disc has changed. Using unique volume identifiers for each CD-ROM product and version minimizes the chances of a user having two discs with the same volume identifier. This will increase the effectiveness of the checks MSCDEX performs to determine if the user has changed the CD-ROM.

Your application should also use the volume label to tell if the CD-ROM disc has changed. Your application can obtain the volume label for a CD-ROM on MS-DOS 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 your application to know if the disc it wants is in the drive. **Your application should not communicate with the driver or drive to determine if the media has changed**. Doing this can prevent MSCDEX from learning that the disc has changed and from performing the initialization it needs for a new disc.

Keep Directories Small

For performance reasons, try to keep directory sizes smaller than about 40 files or so. Much beyond this and directory files grow beyond one 2048 byte sector. Typically this is not a problem. However, if the number of sector buffers chosen when MSCDEX is started is small and the directory files are large, the 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 programs, such as some 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 - the user simply changes the parameter to MSCDEX and reboots) 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.

Organize data by usage, not by type

Since the penalty for using a file in the lowest sub-directory instead of the root-directory is virtually nil, and since 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. Keeping related files close together will reduce seek time when accessing them concurrently. Keeping them in the same directory will help prevent swapping out directory sectors.

This means that data should not be organized by type (bitmap, audio, etc.), but by usage. The bitmaps and audio used together should be placed in the same directory when possible rather than putting all the audio in one directory and all the bitmaps in another. If necessary, the file could be duplicated in multiple directories to keep it 'close' to the related data. The performance is usually much more important than the 'wasted' space when there are 600MB to use.

What makes the difference? For example, suppose you have 1000 directories with 40 files. On average, you'll read about one sector per file open and scan one-half 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 path table, it is much better to have many small directories.

The size of the path table also limits how many subdirectories you might want. If there are too many you might "thrash" reading the path table sectors. Each path table sector holds pointers to 100 to 200 directory files. (The number of pointers depends on the length of the directory names.) If you have a path table that is 10 sectors long, you will want at least 10 sectors of memory buffers to hold the path table. If you use less than 10 sectors you risk re-reading sections of the path table on every file open which is very costly.

One important point from this discussion is that you can vastly improve your file open speed by making sure you have enough memory buffers in MSCDEX. 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 excessively. If you allocate 14 sectors for this example (/M:14), then the whole directory file will find its way into the cache and you will stop hitting the disc. You should consider placing recommendations to the user for MSCDEX settings in the documentation and help sections ("why is it slow"). You should also consider automatically modifying the users autoexec.bat file on setup, but only if they agree to let you.

The difference in speed may be several orders of magnitude. A safe recommendation is to reserve as many sectors as there are in the path table plus the number of sectors for the largest directory plus two. The last two sectors are reserved for data sectors and internal dynamic data. Multiple drives complicate this formula because the buffers are not tied to specific drives and are shared, and because not all drives are active at the same time.

Finally, do not rely on the file system to do your searching for you. If you are performance conscious, finding a chunk of data by searching the file system with a file name is expensive. Most of the time, locating data through the file system is fine because it is a one-time operation. However, if this is repeated often enough, it may pay to do some of the work yourself. A better method is to place everything into one big file and cache your own hierarchy, indexing, binary trees, or whatever searching scheme you choose rather than asking for the file system to locate it.

Disc Portability

One of the advantages of the High Sierra format is data interchange 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 must determine what restrictions on the High Sierra format other operating systems may impose.

Optimizing Your Program for Using a CD-ROM

This section covers programming techniques particular to CD-ROM. Generally these are speed optimizations. The techniques discussed in other sections should also be used (ex, storing files by usage, not by type).

Maintaining 150K/Second

To read data at the specified transfer rate of 150K/second, the application must request data at least at an average rate of 150K/Second. The drive is assuming that you are going to be requesting the 'next' block of data from the CD-ROM and gets it ready for you. The amount it can 'get ready' depends on how big its data buffer is. If you wait long enough without requesting data from the driver, it will run out of room to place the 'next' data in its buffer. This can result in degradation of read rates because the driver must re-seek back to where it left off to resume reading the 'next' data block.

Copy Protection

Because the capability if CD-ROM is so high, copy protection can be effectively achieved by just requiring a large (hundreds of megabytes) file to be present. This file will prove very difficult for the average (or even above-average) user to copy, much less distribute to his friends. As with all copy-protection schemes, this is far from perfect, but it is not as cumbersome for the user as current code-wheel or documentation-based schemes.

Search Strategies

Try to avoid relying on the operating system in 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.

On a hard disk, the time involved to access this type database is not large. However, on a CD-ROM the search times can add up. Opening a file can be an expensive operation simply because the system must read the directory before it can open the file. With a fast drive, seeking to a location on a CD-ROM can take about 10 milliseconds. At worst, a seek can exceed a second on some older CD-ROM drives. Some newer drives have a worst case seek time of about half a second. If you can avoid this, your application will respond faster. MSCDEX caches as many directory sectors as it can so that searching the most active directories is very quick. However, any operations that continually searchs multiple directories can clear out the cache and therefore renders MSCDEX's caching ineffective.

The strategy used by *Microsoft Bookshelf* is 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 caches in memory as much of the root of the tree as it can.

Combining databases avoids the 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.

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 the user's hard disk with all updated files in them and have MSCDEX look there first before searching the CD-ROM. Unfortunately, by the time MSCDEX 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. This lets you update them, and it lets them start 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. Since the CD-ROM appears as just another drive, this means searching a sequence of 'paths' much like the MS-DOS PATH environment variable does.

Read-only Attributes

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 assumed that it could create and write temporary files to the presently connected drive.

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 path name to use when creating temp files. Most people understand this convention now. (If the TEMP directory is located on a RAM-drive, the user gains an added benefit of improving the speed of accessing the information in the temporary file.) 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. File I/O errors will still need to be checked in case the assumptions are wrong.

As a rule, for both temporary and permanent files, if a file creation error occurs, allow the user to re-specify the path name 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.

Data and Disc Format

Don't depend on the format of data on the disk. For example, CD-ROMs do not have a FAT so an application should not rely on finding 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). Standard MS-DOS INT 21h calls should provide everything you need to get at the file contents and attributes.

Determining Your CD-ROM is Installed

Do not rely on functions that work on the directory entries to determine if a file is accessible (and therefore the correct CD-ROM is in the drive). This is because MSCDEX caches the directory data and does not check for media change on these calls. This means that the user can change disks, and this method will not detect it unless the user (or the application) first does a directory listing (ex, with File Manager, or using the File/Open dialog).

You should explicitly open a file on the CD-ROM to determine if it exists. Note that opening a file for read/write access will succeed, but actually writing to the file will fail. This is for compatibility reasons where applications use read/write open attribute even though they just do reads.

Some C-library calls that use the directory structure are: access(), stat(), fstat(), GetCWD().

For example, GetCWD() calls INT 21h Function 47h, Get Current Directory. This is listed in the section on INT 21h, below.

MSCDEX High-Sierra/ISO 9660 Support

The following is technical information about the current version of MSCDEX. It assumes knowledge of the CD-ROM formats. This sections is for low-level developers. **Most developers will not need to bother with the details of this section**. In most cases the formatting software for your system (or at the service house) will take of these details.

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

Correctness

Make sure that your disc is in valid High Sierra or ISO 9660 format. *Nothing* is guaranteed if your disc is not in valid format. Surprisingly enough, we have received several discs that have one or more invalidly formatted data areas. Some of the formatting errors include 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, MSCDEX will behave incorrectly (and at worst will crash the system).

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

Path Table and Directory sizes

Using invalid sizes for the path table and directory are a common error when creating CD-ROMs for MS-DOS. The following guidelines apply to path tables and directories:

- MS-DOS directory file sizes are always a multiple of the logical sector size—2 kilobytes.
- Path table sizes are always the exact number of bytes contained in the path table (which is typically not a multiple of 2 kilobytes).
- You must not have any blank directory sectors.

- The directory length must reflect the correct length of the directory file.
- Directory sectors always begin on a logical sector boundary.

8.3 Filenames

MS-DOS cannot handle longer than 8.3 filenames (8 characters for the filename and 3 characters for the filename extension). 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 map to the same filename. For example, both FILENAME1.TXT and FILENAME2.TXT will appear as FILENAME.TXT.

Take the time to verify the filenames. Most formatting software will report errors when it encounters invalid file names. If you are running the software, you can catch these errors. Error reporting from a service house will vary, but at worst you will lose time correcting the data.

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

MSCDEX does not support any record formats so MSCDEX will ignore the RECORD bit in the file flags byte in the directory entry.

Interleaving

In the present version, MSCDEX does 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 MSCDEX 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 current version. This is for shift-Kanji.

Version Numbers and Associated Files

Version numbers and associated files are not supported by MSCDEX. MSCDEX 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. In the case of associated files, they are not accessible by any MS-DOS function.

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.

MSCDEX Extended Errors

MSCDEX issues the following extended errors that your application can trap if it chains into the INT 24h vector:

Value	**Message**
100	CDR100: Unknown error
101	CDR101: Not ready
102	CDR102: EMS memory no longer valid
103	CDR103: CD-ROM not High Sierra or ISO-9660 format
104	CDR104: Door open

Special Considerations When Using a Read-Only Device

When developing software that is used with both read/write and read-only devices, your application should be prepared to handle the MS-DOS function responses from a read-only device if it attempts to write to it. This section summarizes how MSCDEX responds to the common MS-DOS functions used to update files or directories.

These low-level functions are eventually called by your applications whether you use a high level authoring tool, or the C runtimes supplied with your compiler.

INT 21h Function 3Ch - Create File
This function always sets the carry flag to indicate failure.

INT 21h Function 3Dh - Open File
This function fails with write access and when attempting an open with a non-existent file. To prevent compatibility problems, it does not fail when opening for read or for read/write access. Note that even when the file is opened for read/write access, any attempts to write to the file (INT 21h Function 40h) will fail.

INT 21h Function 4301h - Setting File Attributes
This function always sets the carry flag to indicate failure.

INT 21h Function 56h - Rename File
This function always sets the carry flag to indicate failure.

INT 21h Function 40h - Write File or Device
This function always fails. This is true even if the file was opened for read/write access.

INT 21h Function 39h - Create Directory
This function always sets the carry flag to indicate failure.

INT 21h Function 3Ah - Delete Directory
This function always sets the carry flag to indicate failure.

INT 21h Function 6Ch - Extended Open File
This function fails with write access and when attempting an open with a non-existent file. To prevent compatibility problems, it does not fail when opening for read or for read/write access. Note that even when the file is has been opened for read/write access, any attempts to write to the file (INT 21h Function 40h) will fail.

INT 21h Function 13h - Delete File (FCB and Extended FCB)
This function always sets AL = FFh to indicate failure.

INT 21h Function 16h - Create File (FCB and Extended FCB)

This function always sets AL = FFh to indicate failure.

INT 21h Function 17h - Rename File (FCB and Extended FCB)
This function always sets AL = FFh to indicate failure.

Special Considerations For Other MS-DOS INT 21h Functions

In most instances, file-based MS-DOS INT 21h functions behave identically to those used with any other media. Drive- or device-based MS-DOS INT 21h functions are a different matter. Since MSCDEX relies upon the MS-NET interface to DOS, many of these MS-DOS functions will behave as if the CD-ROM were a remote drive. With other MS-DOS functions, the MSCDEX redirected drives will not respond as a network drive would in order to preserve some characteristics of removable local media.

These low-level functions are eventually called by your applications whether you use a high level authoring tool, or the C runtimes supplied with your compiler.

INT 21h Function 42h - Set File Pointer
As a note, repositioning the file pointer (seeking) does not translate directly to moving the actual CD-ROM device head location. The CD-ROM drive might not reposition the heads until data is read from the disc.

INT 21h Function 43h - Get/Set File Attributes
This function does not cause a media-change check, and so will return data based on the most recently read directory information. For a reliable indication, execute the Volume Label function (INT 21h Function 440Dh Minor Code 66h) prior to executing this function.

INT 21h Function 44h (IOCTL) Subfunction 08h - Check Block Device Removable
This function fails when checking for a drive supported by MSCDEX, but since it checks as being remote this is correct.

INT 21h Function 44h (IOCTL) Subfunction 09h - Check Block Device Remote
Checking for a CD-ROM drive returns as a remote drive and fails when checking for removability. This is similar to a network drive; however, note that the CD-ROM drive does not appear on the network redirection list.

INT 21h Function 47h - Get Current Directory
After a disc is ejected, this function will return the directory selected for the last current directory. Thus, **this function is not a reliable indicator to determine if the media is present in the drive or if the directory is still valid**. For a reliable indication, execute the Volume Label function (INT 21h Function 440Dh Minor Code 66h) prior to executing this function.

INT 21h Function 5Fh Subfunction 02h - Get Redirection List Entry
The logical drive supported by MSCDEX does not show up in this list under any of the indexes. To this respect the CD-ROM drive behaves like a local drive; however, the drive checks as remote making it behave somewhere between a network drive and a local drive.

Determining If a Drive Is a CD-ROM

This sample code fragment demonstrates how to determine if a given drive is a CD-ROM drive. This code is available on the MMSys BBS (206 936-4082) in the samples area as ISCDROM.ZIP

```
;     (C) Copyright Microsoft Corp. 1991.  All rights reserved.
;
;     You have a royalty-free right to use, modify, reproduce and
;     distribute the Sample Files (and/or any modified version) in
;     any way you find useful, provided that you agree that
;     Microsoft has no warranty obligations or liability for any
;     Sample Application Files which are modified.

;*----------------------------------------------------------------------------*
;*                                                                            *
;*  BOOL IsCDROMDrive(int iDrive)                                             *
;*                                                                            *
;*  Determine if passed drive is a CDROM drive controlled by MSCDEX.          *
;*  iDrive = drive index (0=A, 1=B, ...).                                     *
;*                                                                            *
;*  Return boolean.                                                           *
;*                                                                            *
;*----------------------------------------------------------------------------*

cProc IsCDROMDrive, <FAR, PUBLIC>

ParmW iDrive

cBegin
        mov     ax,150Bh        ; MSCDEX driver check API
        xor     bx,bx
        mov     cx,iDrive
        int     2Fh
        cmp     bx,0ADADh       ; was MSCDEX the int 2F guy?
        je      @f              ;    yes, AX != 0 if drive is a CDROM
        xor     ax,ax           ;    no, can't be an MSCDEX CDROM drive
@@:
cEnd
```

Playing CD-Audio

Use the MCI CD-Audio driver to play CD-Audio (MCICDA.DRV). Avoid using the direct MSCDEX interface. This interface is not translated by Windows automatically and therefore requires the caller to use the DPMI services. This greatly complicates interfacing directly to MSCDEX, and therefore this is not recommended.

The MCI command set for CD-Audio is defined in the MDK documentation, the MDK books from MS-PRESS, and the IBM/Microsoft RIFF/MCI definition document. The RIFF/MCI document is available on the MMSys BBS. The MDK is available through Microsoft or other dealers. The MDK Books are available from your local technical bookstore.

The MCICDA.DRV is also available with Windows 3.1. However, it is not automatically installed in the user's system. If you use this facility, you should include instructions to the user on how to install the driver from his Windows distribution disks (use the Driver applet in the Control Panel, etc.).

End.