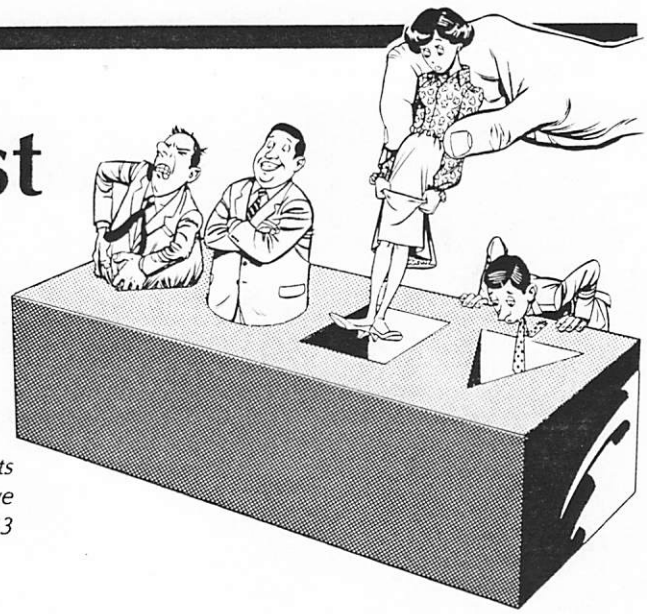
Squeezing The Most Out of Your HDOS Diskettes



Glenn F. Roberts
9035 F Countrywood Drive
Knoxville, TN 37923

The intent of this article is to explain ways in which one can reduce the amount of floppy disk space which must be reserved for system files in the Heath Disk Operating System (HDOS). This is a particularly important issue to H-8 and H/Z-89 users with only a single hard-sectored type floppy disk drive. To a large extent this article pulls together ideas from past articles in REMark, Sextant, and Microcomputing, however, some of the ideas presented here have not been described elsewhere. Even readers who don't feel they need any more storage capacity (if such people exist) should be interested in the disk concepts presented here.

Note: Some of the techniques presented here can lead to essentially irreversible damage to the disk indexing structure if not performed properly. It is recommended that you test them out on unused disks first or else back up everything before starting.

Introduction

When I finished constructing my H-8 several years ago, one of the first programs I wanted to try out was "Adventure" (HUG part no. 885-1010). I was recalling playing the game until the wee hours of the morning during my undergraduate years and was anxious to pick up my adventure where I had left off, somewhere in the twisty little maze (or was it the maze of little twisty ...?). It was then that I got my first real lesson in the realities of HDOS: it is big. After INITing and SYSGENing a new disk, I deleted all of the files that didn't have the L (lock) flag set and found myself with 240 free sectors on the new disk. A quick CATalog of the Adventure disk showed that I apparently needed at least 246 (35 for the game itself, 188 for the main data file, and 23 for the game parameters file). It took me a day or two of fooling around until I discovered I could simply delete the supposedly "locked" file DK.DVD to gain an extra 16 sectors, more than enough to let me get my Adventure program up and running!

Since then I have learned a great deal about how HDOS works and what tricks and techniques can be used to increase the user's portion of the disk space allocation. I thought it would be fun to go back now and see just how many of those system sectors I could have recovered for my own use had I known then what I know now. I ended up recovering 56 more sectors, enough to let me SAVE three Adventure games and still have room to spare. In the sections that follow, I will explain the steps I took to recover these sectors. Much of this information is taken from past articles in REMark, Sextant, and Microcomputing. The articles are referenced in the text by listing the author and year of publication; refer to the listing at the end of the article for the complete article citation. I urge the reader to look up the original articles whenever possible. I will limit my discussion to the "hard sectored" (H-17) type floppy disk formats which have been standard on the H-8s and H/Z-89s, since these are the ones most likely to exhibit space limitation problems. Much of what I say however applies to HDOS in general and thus is also applicable to the newer "soft sectored" disks.

Minimal SYSGENing

Much of my original frustration with the Adventure game could have been bypassed if I had simply taken the time to carefully read through the HDOS manuals. Had I done so, I would have found that the SYSGEN program has an option switch called "/MIN". SYSGEN is the program which copies the various system programs which are needed to make a disk bootable. By typing SYSGEN /MIN one can request that a minimal set of these system programs be copied to the new disk. The files copied in this minimal configuration are described below:

HDOS.SYS — Contains the main HDOS code including the TT: device driver and the resident SCALLs (system routines) such as .SCIN, .SCOUT, .EXIT, etc. This code must be resident in high memory at all times except during initial boot-up.

HDOSOVL0.SYS — This file contains the principal overlaid SCALLS such as .OPENR, .CLOSE, .POSIT. These are "overlaid" since HDOS has the ability to swap them out to disk when a user's program requires a large amount of memory.

HDOSOVL1.SYS — This file contains less frequently used SCALLS such as .MOUNT, .RESET, and .DAD. Like HDOSOVL0, this file is also overlaid.

SYSCMD.SYS — This is the command processor program which is loaded into low memory whenever the user is at the command level. This is the program which generates the familiar ">" command prompt. Its job is to process user commands either by calling system routines (SCALLs), by linking to PIP, or by linking to a user's program. SYSCMD is loaded and takes control whenever an .EXIT SCALL is issued by a running program.

PIP.ABS — This is a general file and device utility (Peripheral Interchange Program) which copies, lists, renames, and deletes files and provides the user access to the disk directories. Some of the system commands (e.g. COPY, TYPE, CAT, etc.) explicitly call PIP but it is loaded and executed only as needed.

SY.DVD — This is the device driver for the H-17 disk drives. Its job is to interpret specific device driver calls (e.g. read, write, mount, abort, open, close, etc.) and control the disk hardware accordingly. The important thing to remember about device drivers is that the driver calls are the same regardless of the type of hardware being accessed. This "device independence" allows the application programmer to use software to interface almost any type of I/O device to the system. For more information on device drivers and driver calls, see "The HDOS Device Driver Programmer's Guide", (Dallas, et al., 1981), or refer to the source files on the "Device Drivers" disk distributed with HDOS.

RGT.SYS — This is the Reserved Group Table. Its primary function is to allow HDOS to lock out disk sectors which are to be flagged as unreadable.

GRT.SYS — This is the Group Reservation Table. HDOS stores files in pieces which can be strung out on random tracks and sectors throughout the disk. The GRT is a crucially important file since it contains the pointer information which allows HDOS to piece files back together into their original form. If the GRT file is damaged in any way, HDOS may be unable to read any of the files on the disk.

DIRECT.SYS — This file contains the disk directory. The information in this file, combined with that in the RGT and GRT tables, allows HDOS to locate and modify any file catalogued on the disk, either through a utility program like PIP or via a user program's SCALLs.

Note: For more information on the RGT, GRT, and DIRECT files, the interested user is strongly urged to read Tom Jorgensen's article "Dissecting the HDOS Diskette" (Jorgenson, 1981) and Herb Friedman's article "Understanding HDOS" (Friedman, 1983).

The above files will require a total of 129 sectors and the CAT command will report that there are 256 free sectors, for a total of 385 sectors. We know however that the disk is capable of storing 400 sectors (40 tracks at 10 sectors per track), so what happened to the other 15 sectors? Ten of these "missing" sectors are accounted for by track 0, sometimes called the boot track. When a new disk is INITialized, a "bootstrap" program is written on the first nine sectors of this track. When the computer is booted, the routines in ROM seek out track zero, load these nine sectors, and execute this bootstrap program. In order to prevent HDOS from storing files here, the entire first track is flagged as unusable by locking it out in the Reserved Group Table (RGT) file. Thus, as far as HDOS is concerned, these 10 sectors are unusable. The sectors on track 0 are normally read only during the process of mounting, dismounting, or booting the disk.

The remaining five "missing" sectors are hidden in five of the nine system files listed above. This occurs because HDOS allocates file space in groups (also called clusters or extents) of two sectors at a time, thus in reality all files require an even number of sectors on the disk. Even though the CAT command may show that a file has an odd length (e.g. HDOS.SYS is 31 sectors long), the user must remember that an extra sector is reserved for use by such files and that this expansion space will not be counted as being available for other uses. If you issue the CAT command with the /ALL flag, you will see all allocated sectors shown and a total of 390 sectors (i.e. all but the locked out track 0).

Resetting SY0:

Before going any further, I should take some time to discuss how and when you may replace the main "boot" disk in drive 0. This is important on single disk systems where you may not always need the system files to be present. When a diskette is MOUNTed in HDOS either directly, via the MOUNT or RESET command, or indirectly by "booting" up with the disk, HDOS stores certain critical information regarding the diskette in system RAM. This information includes the volume number of the disk and the absolute track and sector locations of various system and directory related files on the disk. The primary benefit of this scheme is that it speeds up disk access time. The price we pay is that we cannot arbitrarily swap disks without first informing HDOS via the RESET command (or equivalently the /RES option of PIP).

Before RESETting the boot drive (SY0:) one should normally set the "STAND-ALONE" flag using the SET program (Cohn, 1983; Pinkston, 1983). The stand-alone mode of HDOS is a frequently used yet undocumented feature of HDOS which forces all "swapped" files to remain resident in system RAM (these files include the system overlays and device drivers). To set the stand-alone mode merely type:

SET HDOS STAND-ALONE

Normal swapping mode can later be restored by SETting NO-STAND-ALONE. The advantage of this feature is that SY0: can be RESET to a disk containing only SYSCMD.SYS and PIP.SYS. The disadvantage is that the swapped out files now use up a chunk of RAM (usually about 6K).

You can also change the disk in SY0: if you are running an application program which is written to explicitly anticipate this situation. The HUG DUMP program (HUG part #885-1062) does this by revectoring one of the system ROM routines to a new routine which does not check the disk volume number when it reads a sector from the disk. Thus when you are running DUMP you may swap diskettes arbitrarily in any of the drives so long as you put everything back where it was before exiting from the program! Other ways to get around these safety features of HDOS are mentioned in the article "Disk Programming Without HDOS" (Smith, 1982).

I mention these points now since they may be of use in making the various patches and changes discussed in the remainder of this article, especially for readers with single drive systems.

Patch History Tables

If the MINimal SYSGEN does not give you the file space you need, the first thing you might want to consider is eliminating the Patch History Table (PHT) sector from all of your system files. The PHT was designed as a special feature of HDOS to allow the PATCH program to maintain a log of patches made to system programs (see Swayne, 1982a). In practice the PHT feature is rarely, if ever, used and can easily be eliminated by simply removing the PHT sectors appended to each system file. Note however, in light of my previous comments, that removing the PHT sector will only be useful on files which originally had an odd number of sectors. Such files will be reduced to an even number of sectors and you will have recovered two sectors.

The technique for removing the PHT is documented in the article "Losing Weight with HDOS 2.0" (Swayne, 1981a). In this article, Pat Swayne presents an assembler program which automatically strips the PHT sector off system files. *(Editor' note: Be sure to correct the program as shown in REMark #21, page 4.)* The following files can be reduced by two seconds each: EDIT, PATCH, INIT, SYSGEN, TEST47, ASM, XREF, DBUG, PIP, HDOS, and HDOSOVL1. Reducing these last three would raise our free sector total to 262 in the minimal configuration described previously.

A Compact SY: Driver

If 262 sectors is not enough user space for you, you might next consider creating a compact SY: device driver. This is another trick first pointed out by Pat Swayne in his brief article "A Tiny SY.DVD" (Swayne, 1981b). The SY: device driver is actually two programs

joined together; the driver itself, and the initialization code. The designers of HDOS decided to include the device specific portion of the initialization code in the driver itself. This makes it possible to use the same INIT program to initialize various disk types (e.g. the 5-1/4" H-17 or the 8" H- 47), but it means we must carry along this initialization code in all our drivers, even those not used for INITialization.

The "Device Drivers" disk distributed with HDOS contains the file SYDVD.ASM which is the assembler source for the device driver portion of the SY: driver. This was designed to be combined with SYINIT (the initialization portion of the SY: driver) using the utility program MAKMSD, however, the device driver portion also functions fine by itself as long as you don't want to initialize any disks using it. If you assemble SYDVD.ASM, delete the original SY.DVD file (it can be deleted even though it is locked), and rename SYDVD.ABS as SY.DVD you will have an SY: device driver which occupies only 4 sectors. This represents a savings of 6 sectors. Total free space on our compact system disk is now 268 sectors! (**Note:** if you are working on a single drive system, be sure to reboot after doing anything with any device driver on your boot disk. This is necessary because HDOS looks for device driver addresses only once, at boot-up time, and moving the track and sector location of any device driver without informing HDOS can be disastrous.)

Trimming Fat DIRECTories

The HDOS directory (file DIRECT.SYS) is normally 9 groups (18 sectors) long. Each group can store 22 file entries for a total of 198 file entries. As Tom Jorgensen points out (Jorgenson, 1981), this is actually 22 more entries than the number of files it is currently possible to write on the diskette. In surveying my own disks, I have found that most of my diskettes contain no more than 30 or 40 files, and some have fewer than 20. This means that I usually need only 2 or 4 sectors allocated to the DIRECTory and thus can recover 14 to 16 sectors. There are a number of "public domain" programs which allow one to shorten the length of the directory. These may be available through a local HUG or in the HUG area on Compuserve. *(Editor's note: There is also REDUCDIR on HUG disk 885-1120.)* If you don't have access to one of these programs, you can recover these sectors using the HUG's DUMP program (disk #885-1062) and a little knowledge of the diskette's file structure.

The technique I will describe shortens the file DIRECT.SYS to only 2 sectors which will allow a maximum of 22 files to be stored on the diskette. If you perform this modification and you subsequently try to store more than 22 files, HDOS does not panic but merely informs you that it has run out of directory space. You should perform the following steps immediately after INITializing a new diskette (i.e. before SYSGENing or COPYing any files to the new disk). Since the technique is rather tricky, you might want to save a copy of the final disk so that you can make more "mini-directory" disks at a later time using an absolute sector by sector copy program such as DUP (also on HUG utility disk #885-1062).

The directory file DIRECT.SYS is created by INIT.ABS and is normally located on the 18 sectors starting at track 13, sector 0. The only time it will not be located here is if you indicate that one or more of the sectors in this area should be locked out because of some sort of damage to the media. In order to speed up access time, the groups in the directory are normally stored in an interleaved fashion starting at track 13, sector 2. Table 1 shows the normal locations of each of the 18 sectors in the standard DIRECT.SYS file.

A freshly INITialized disk will have only three files on it: RGT.SYS, GRT.SYS, and DIRECT.SYS. The directory entries for these files will normally be located at the end of the 4th directory sector, that is on track 13, sector 7. In order not to disturb these entries, the best way

Directory Sector	Track	Absolute Sector
1-2	13	2-3
3-4	13	6-7
5-6	13	0-1
7-8	13	4-5
9-10	13	8-9
11-12	14	2-3
13-14	14	6-7
15-16	14	0-1
17-18	14	4-5

Table 1. Absolute track and sector locations of the 18 sectors which comprise the file DIRECT.SYS on a diskette which is initialized via INIT.

to make a 2-sector directory is to relocate the start of the file DIRECT.SYS to be track 13, sector 6, and the end to be track 13, sector 7. There are three places on the disk where patches must be made to accomplish this: GRT.SYS, DIRECT.SYS, and the boot track.

The easiest patch to make is the boot track patch. As I mentioned before, sectors 0-8 on track 0 are reserved for a 9 sector bootstrap program. Sector 9 on track 0 is reserved for storing various pieces of system information about the disk including the disk name and volume (see Swayne, 1982c). The byte definitions for this sector are contained in the file LABDEF.ACM on the "Software Tools" disk distributed with HDOS 2. The fourth byte of this sector (LAB.DIS) is the pointer to the directory sector. This byte is needed by the bootstrap program to find the directory and subsequently find the file HDOS.SYS. This byte is normally hexadecimal 84 (decimal 132), thus it normally points to track 13, sector 2. Since we want the new directory entry to be track 13, sector 6, we must change this byte to hexadecimal 88 (decimal 136). This is easily done using the HUG utility DUMP. Using DUMP, look at track 0, sector 9 of the disk to be modified. The fourth byte should be changed from 84 to 88.

The next file to be patched is DIRECT.SYS. Since the bootstrap must be able to scan through the directory before the normal HDOS file handling routines are available, a simple chained structure is incorporated in the file itself to logically connect the directory sectors during boot-up. Each 512 byte group of the directory can store 22 file entries at 23 bytes each for a total of 506 bytes. The 507th byte is always 0; the 508th byte is the number of bytes in each file entry (normally hex 17); the next two bytes contain the address of the current group; the last two bytes contain the address of the next group (or zero to indicate the end of the directory).

The first patch we must make to DIRECT.SYS is to change these last two bytes to 0 in what is to be the last group of our new directory, in this case track 13, sector 7. Using DUMP to look at this sector, you will see that the next to last byte will be hexadecimal 82 (indicating that the next group in the directory is at track 13, sector 0). By changing this byte to 0, you will inform the bootstrap that this is now the end of the directory.

If any of the above patches are done incorrectly, the disk will probably not be bootable. The remaining patches are to tell the normal HDOS file handling routines the information it needs about the file DIRECT.SYS. If any of these are performed incorrectly, you will probably get the "Disk Structure Corrupt" message when you try to boot or mount the disk.

The next patch is also made to DIRECT.SYS. In this patch you must change the entries for the file "DIRECT.SYS" itself, in particular you must change the values for the starting and ending groups. Refer to Table 2 which is taken from the file DIRDEF.ACM (on the "Software Tools" disk that comes with the HDOS distribution package) and shows the layout of each 23 byte directory entry.

```
          ORG     0

DIR.NAM DS      8       File Name
DIR.EXT DS      3       Extension
DIR.PRO DS      1       Project
DIR.VER DS      1       Version

DIR.CLU DS      1       Cluster Factor
DIR.FLG DS      1       Flags
        DS      1       (reserved)
DIR.FGN DS      1       First Group Number
DIR.LGN DS      1       Last Group Number
DIR.LSI DS      1       Last Sector Index
DIR.CRD DS      2       Creation Date
DIR.ALD DS      2       Last Alteration Date
```

Table 2. The layout of each directory file entry.

The 16th and 17th byte of each directory entry is the first group number and last group number for that file, respectively. Since these are group numbers, we must multiply by the number of sectors per group (two) to get the track and sector. For the file DIRECT.SYS, the first group number is normally hexadecimal 42. Multiplying by 2 we get hexadecimal 84 which is decimal 132, or track 13, sector 2, just as we showed in Table 1. In the new "mini" DIRECT.SYS, the first group and last group will both be track 13, sector 6 (hexadecimal 88), thus the new values for entries DIR.FGN and DIR.LGN will be 88/2 = 44.

Using DUMP you can now modify the first and last group numbers for the file DIRECT.SYS to both be 44. These should normally be found in bytes DC and DD (hexadecimal) of track 13, sector 7.

The last patch which has to be made before the mini-directory is in place is a patch to the file GRT.SYS. This file contains the group reservation table which is nothing more than a series of pointers which tell HDOS how to reconstruct the files on the diskette. To see how a file can be strung out over many portions of a disk, you may examine a file's structure as follows. Using DUMP and the information in Table 2, locate the file's first and last group numbers in the file DIRECT.SYS, then DUMP the single sector file GRT.SYS (normally on track 14, sector 8) onto the screen. Beginning with the first group number, look into the GRT at the corresponding entry, for example if the first group number is hex 42 (as is normally the case for the file DIRECT.SYS), then look at the byte at location 42 in the GRT. The entry at this location will be the number of the next group in the file. If you look at the GRT entry corresponding to this number you will find the next group number, and so on. When you find an entry in the GRT which is zero, you know you are at the end of the chain. The cell containing this terminating zero should correspond to the last group number as found in the directory. If you reconstruct the sequence of groups for the file DIRECT.SYS in this manner, you will normally obtain the following: 42, 44, 41, 43, 45, 47, 49, 46, 48. If you convert these 9 numbers to decimal track and sector locations, you will get the values in Table 1.

Now with this information you can easily make the patch to the GRT. The new file DIRECT.SYS will start and end at group number 44 (hexadecimal). This means that we don't have to chain any groups together, we simply need to indicate that group 44 is the last group. We do this by using DUMP to place a zero in byte 44 in the GRT. Note that when you make changes to the GRT table, the disk being changed should NOT be MOUNTed since when it is later DISMOUNTed or RESET, HDOS will copy back the version of GRT which it maintains in RAM. You can do the patch using DUMP since DUMP allows you to change the disks at random, just be sure and put all disks back where they were before Control-C'ing out of DUMP.

You should now be able to mount this diskette and find that the file DIRECT.SYS is only 2 sectors long. If you get the "corrupt" message you did something wrong. Try and find your mistake using DUMP. If you have made no mistakes, then you now have a diskette with 284 free sectors.

Application and Turnkey Systems

Making the changes outlined so far is about as far as you can go and still retain all of the features of HDOS. Very often however, one needs to set up an application disk which will only be used to execute some pre-specified set of application programs. On most such disks you can delete the file PIP.ABS. (For help deleting "locked" files, see "Recovering a Deleted File", Harton, 1981, "Recovering Deleted Files In HDOS and CP/M", Swayne, 1982b, or "PATCH Mysteries Revealed", Swayne, 1982a.) After you have deleted PIP, you will no longer be able to use the commands HELP, TYPE, LIST, DELETE, RENAME, CAT, DIR, IND, or INDEX since all of these require PIP to perform their tasks. If you try to use one of these commands, HDOS will simply inform you that it needs PIP to perform the command. If you delete PIP you will have recovered 20 sectors, bringing the total to 304 free.

Going one step further, you may want to set up a "turnkey" system such that your application program is automatically executed on boot-up and the disks are automatically dismounted upon program termination. My ADVENTURE disk is a good example of such a case. HDOS has a built in feature which automatically scans the directory for a file called PROLOGUE.SYS upon booting up. If such a file is found, execution is passed to it as soon as the normal HDOS system files are loaded. If no such file is present, control is passed to SYSCMD.SYS and the user is prompted for a command in the usual manner. This feature allows you to set up such a turnkey system by renaming your program to "PROLOGUE.SYS". In a turnkey system, you won't normally need to have the file SYSCMD.SYS since all interaction with the user should be handled by your program. You could simply delete SYSCMD.SYS, but nasty things will happen if your program tries to return to HDOS via the .EXIT SCALL. A better way is to create a new version of SYSCMD.SYS which simply dismounts the system disks and returns to boot level. Listing 1 shows such a program. When you assemble this program and rename it to SYSCMD.SYS, then rename your application program to PROLOGUE.SYS, you will have a turnkey system which will run only your specified program and will dismount the disks upon program termination.

Going Further

One rather sophisticated trick which the ambitious programmer might want to try is to recover some of the sectors on the boot track (track 0). I mentioned previously that the first 9 sectors on track 0 are reserved for the bootstrap program. These are "locked out" to HDOS by flagging them as unusable in the Group Reservation Table (GRT.SYS). Much of this space is not needed, in fact I have written a bootstrap program which resides in only 2 sectors. I did this by disassembling the original 9 sector version, eliminating unneeded code, reassembling, and storing the code on track 0. A surprisingly large amount of the original code can be eliminated, for instance: 1) most of the first two sectors are reserved for use on DK: type devices where the basic disk access software is not in the H-17 ROM; 2) much of the code is for accessing the older type cassette serial interface (H-8-5); 3) part of the code is used to determine the baud setting (which I always leave at 9600); 4) a big piece of the code is used for computing sector checksums.

If you do create a custom bootstrap as I did, you will have up to 7 more sectors free on track 0. You might think that you could simply "unlock" these sectors in GRT.SYS to make them available for general use by HDOS. Unfortunately the boot track is initialized with

a volume number of zero which will be different from the volume number on the other tracks (it must be between 1 and 255). This is done so that the bootstrap program itself can initially be read in, however, it means that if you try to read it through the HDOS SCALLs, you will get an error. One way around this is to read and write to this sector using the ROM routines directly as described in "Disk Programming Without HDOS" (Smith, 1982). You can also read track 0 using the undocumented "read regardless" call to the SY driver (Swayne, 1982c). There are some nice things that can be done with these extra sectors. Because they are not "readable" as normal HDOS files, they are the perfect place to store passwords, deciphering keys for encoding files, and any other sensitive information.

This is about as far as you can normally go toward reducing the HDOS overhead on your system disks. If you need even more space, about the only thing you can do is run without HDOS altogether. This involves calling the H-17 ROM disk driver directly to do reading and writing, but it entirely eliminates all HDOS system files from the disk. It also requires the user to provide the means to maintain a file directory as well as any other services previously provided by HDOS. This technique is discussed in Smith (1982).

Conclusion

In this article, I have attempted to summarize some of the "tricks" which can be used to reduce the amount of disk space which must be set aside for HDOS system use. As a minimum, most users can easily retrieve 30 disk sectors for their own use, however, one can go further, even to the point of eliminating HDOS altogether! The articles I have cited represent some of the best technical articles I've seen on HDOS and I urge interested readers to refer to them as well as the excellent Heath manuals for more in-depth treatment of some of these topics.

Articles Cited

Cohn, C. E. Summer 1983. "Squeeze More Disk Space Out of HDOS", Sextant, Issue #6.

Dallas, A., Lamm, D., and Jorgenson, T. September 1981. "The HDOS Device Driver Programmer's Guide", REMark, Issue #20.

Friedman, H. Spring 1983. "Understanding HDOS, Parts 1, 2, and 3", Sextant, Issue #5.

Harton, D. August 1981. "Recovering a Deleted File", REMark, Issue #19.

Jorgenson, T. July 1981. "Dissecting the HDOS Diskette", Microcomputing, Vol. 5, No. 7.

Pinkston, W. June 1983. "Out In The Boonies With a Single Drive H/Z-89", REMark, Issue #41.

Smith, R. E. Spring 1982. "Disk Programming Without HDOS", Sextant, Issue #1.

Swayne, P. August 1981a. "Losing Weight with HDOS 2.0", REMark, Issue #19.

Swayne, P. December 1981b. "A Tiny SY.DVD", REMark, Issue #23.

Swayne, P. May 1982a. "PATCH Mysteries Revealed", REMark, Issue #28.

Swayne, P. October 1982b. "Recovering Deleted Files In HDOS and CP/M", REMark, Issue #33.

Swayne, P. December 1982c. "What's In A Name?", REMark, Issue #35.


```
***     TINYCMD - Tiny replacement for SYSCMD.SYS
*
*       Purpose: To deny the user access to HDOS
*       system commands in "turnkey" applications.
*
*       G. F. Roberts           9/25/83
*
*       This piece of code may be used to replace
*       SYSCMD.SYS on version 2 of HDOS.  It auto-
*       matically dismounts all disks when any user
*       program tries to return to HDOS command level
*       via an EXIT SCALL.  To install it first delete
*       the old SYSCMD.SYS, then assemble this code and
*       rename it to SYSCMD.SYS.  The XTEXT files
*       can be found on the HDOS distribution disks.
*
        XTEXT   HOSEQU          HDOS equates
        XTEXT   ASCII           ASCII equivalences
        XTEXT   HOSDEF          HDOS definitions
        XTEXT   TYPTX           Text typing routine
        XTEXT   OVLDEF          Overlay definitions

        ORG     USERFWA         ORG at start of user RAM

        LON     C               List all XTEXT code
*
*       Entry point
*
START   CALL    $DOS            Dismount all disks
        JMP     ROMBOOT         and jump to bootup

        XTEXT   DOS             Dismount disks routine
        XTEXT   RCHAR           Read character routine

        END     START           End of program.
```

Listing 1.