HDOS System Programmer's Guide

Software Reference Manual

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

595-2553-02
Printed in the United
States of America

TABLE OF CONTENTS

Part 1

INTRODUCTION

Purpose

This manual describes the advanced features of HDOS that are necessary for a user program to interface with HDOS at the assembly language level. This information is provided for use by the more advanced programmer and is not presented in a tutorial manner.

Background

The "HDOS Software Reference Manual" documents the various system commands and BASIC statements used to generate and maintain files at the higher language level. At this level, the novice or average programmer need not be concerned about the involved details of interfacing his programs with HDOS or the disk drives. Since the release of HDOS, Heath has received from some advanced programmers requests for information on how to interface with HDOS at the assembly language level. For their particular tasks, programs must be written in assembly language. It is in an effort to be of service to these users that this manual has been written.

Any comments or questions regarding the contents of this Manual should be directed to and only to the Heath Technical Consultation Department, Benton Harbor, MI 49022.

Preface

HDOS provides a full run-time support environment for assembly language programs. Communications with file-oriented devices, console communications, memory allocation, and other such services are provided by the HDOS system. Since the H8 and H89 do not afford any hardware protection, assembly language routines must be "polite", in that they should not damage the H8 or H89 running environment. This subject will be discussed in more detail further on in this document.

HDOS also contains many useful general-purpose subroutines, which may be called by user programs. These, together with the system services provided, make assembly language programming under HDOS very convenient.

Part 2

RUN-TIME ENVIRONMENT

When you type "RUN fname", HDOS will load your program into memory and run it. This section will discuss the initial run-time environment of the program. Refer to the memory map in Chapter 1 of the Heath HDOS System Manual.

Memory Layout

The first 64 bytes of RAM, from 040000 to 040100, are used by PAM-8. The PAM-8 source listing documents their use.*

The next 295 bytes are used by HDOS and the disk device driver for work cells. These cells are in low memory so that HDOS and its overlays can reference them without having to compute relocation factors (HDOS and its overlays are both relocatable in high memory). Some of the contents of these cells are of interest to assembly language programmers, and are available (indirectly) through HDOS system calls. You should refrain from accesssing them directly, since their position may change with future releases. Use of the proper HDOS symbols and system calls in assembly language programs will make it possible to transport your program to future Heath CPUs executing HDOS. There are a few cells that may be of interest to the programmer; they are documented in Part 7. They may be read, but must never be written.

Following the work cell area is a 279-byte stack area. When a user program is executed, the stack pointer is set to the symbol STACK, which is 042200A. Note that you may not set your stack pointer below that address, and then use the area below 42200A for code or data (other than data stored by a normal PUSH). You may make the stack larger, setting SP to a value larger than 042200A. Calls to the HDOS system will preserve this larger stack.

The user program area starts at 042200A, immediately after the top of the stack. The user program extends until the last byte loaded by the RUN command. Note that the assembler generates a dummy 00 byte as the last statement in a program, so that trailing DS declarations will be contained in the size of the running program. There is a system call which requests access to more memory. You must issue the call first, since HDOS may be using that area for its own code.

* Although the PAM-8 ROM will be referenced throughout this guide, the general-purpose routines of the MTR-88, MTR-89, PAM-8-GO, and XCON-8 ROMs all have common entry points. For specific information, refer to the particular ROM manuals and listings.

After the user program LWA, HDOS may (or may not) have HDOSOVLØ or HDOSOVL1 loaded. HDOSOVLØ and 1 are the HDOS system overlays. The HDOS functions which reside in the overlays are discussed in Part 7 and listed on Page 68. In general, HDOS will attempt to reside HDOSOVLØ. If there is sufficient free room for it, it will remain in memory. This is discussed further in Part 7.

Any active device drivers are loaded immediately before the resident HDOS code. A device driver is loaded when a file is opened on a device whose driver is not yet in memory. The TT: device driver is built into the resident HDOS code and the H17 ROM, and never needs to be loaded. Since the SY: driver is permanently loaded into memory when the system is first booted up, it also never needs to be loaded.

Finally, the HDOS system resides in high memory, up against the upper limit of available RAM. When the system is booted up, HDOS initially loads at a fixed lower address. After sizing memory, HDOS moves its permanently resident parts into high memory. This section contains the TT: and SY: device drivers, the SCALL dispatcher, the overlay loader, and the handlers for many SCALL functions. These are discussed in Part 5.

I/O Environment

HDOS has a vested interest in the I/O ports being used by the device drivers currently in memory. These ports should not be disturbed when HDOS (or a device driver) may be trying to use them. The ports are:

H89	Port	H8
H47 Floppy Disk	170-173Q (078-07BH)	H47 Floppy Disk
H17 Floppy Disk	174-177Q (07C-07FH)	H17 Floppy Disk
Reserved	300-307Q (0C0-0C7H)	Reserved
H88-3 Alternate Terminal	320-327Q (0D0-0D7H)	H8-4 Alternate Terminal
Reserved	330-337Q (0D8-0DFH)	Reserved
H14 Line Printer	340-347Q (0E0-0E7H)	H8-4 Line Printer
Console Terminal	350-357Q (0E8-0EFH)	H8-4 Console Terminal
Reserved	360-361Q (0F0-0F1H)	H8 Front Panel
H88-5 Cassette	370-371Q (0F8-0F9H)	H8-5 Cassette
Reserved	372-373Q (0F2-0FBH)	Console Terminal
Reserved	374-375Q (0FC-0FDH)	H8-5 Alternate Terminal
Reserved	376-377Q (0FE-0FFH)	Reserved

Since the TT: and SY: drivers are permanently resident, it is vital that you do not disturb the TT: and SY: ports. Disturbing the SY: port will destroy your disks. Disturbing the TT: ports will damage the console driver package. The console driver package communicates with the console device at interrupt time, so you will not be able to detect character entry by examining the console status bits. HDOS provides you with a facility to test the presence of a console character.

Interrupt Environment

HDOS is an interrupt-driven system, so be careful how you handle interrupts. Your program must not turn off interrupts via the DI instruction for other than very short periods of time. The H17 device driver makes use of the front panel clock interrupts, so you must not disable them, either directly via port 360Q or by the PAM-8 control word. Likewise, console interrupts are used by the system console handler, and should not be disturbed. HDOS does not currently support any interrupt-driven device drivers, but programs may still make use of interrupts. There are two major trouble areas in this: choosing a vector, and discontinuing the interrupts.

INTERRUPT VECTORS

Of the eight interrupt vectors available in an 8080A, HDOS makes use of six or seven of them. In brief:

0 — Master Clear. Returns control to PAM-8
1 — Clock Interrupts
2 — Single-Step. Used by DBUG. May be used by user program when not running DBUG.
3 — Console Interrupts.
4 — Reserved for Real-Time Clock (if implemented)
5 — Reserved for H47 (if implemented)
6 — Available for user programs.
7 — HDOS SCALL vector.

Set up the vectors by storing a JMP to your interrupt service routine in the PAM-8 ".UIVEC" area, as discussed in the PAM-8 Manual.

DISCONTINUING INTERRUPTS

When a user program causes a device to start issuing interrupts, it must somehow turn off that device before control returns to the system. HDOS will not alter the interrupt vector (JMP) in PAM-8's ".UIVEC", and an interrupt occurring after your program has been removed will be tragic. Also note that as a user, you must be careful of typing CTL-Z, as this can kill your program before it can shut down any interrupting devices.

NOTE: You must turn off the device interrupts before surrendering control to HDOS. Simply replacing your interrupt vector with EI and RET instructions will cause disaster, since the interrupting device will continue to request interrupts until it is serviced, and HDOS does not know how to service it. Your machine will then hang in an interrupt service loop.

CPU Environment

After loading your program, HDOS transfers control to the program's entry point. This is the address specified in the END (assembler) pseudo.

Channel Environment

HDOS allows user programs to communicate with file-oriented devices via "channels". These channels are discussed in Part 3. In all cases, channel -1 (377Q) is open for read access on the device and file that the program was loaded from. This is done so you can conveniently load overlays without having to know under what name and disk drive your program was run from. If your program was run in response to a RUN command, all other channels will be closed. If your program was run in response to a ".LINK" SCALL, then the other channels will remain as they were set up by the program which issued the ".LINK".

Part 3

I/O CHANNELS

All file I/O in the HDOS system is done via I/O channels. "File I/O" refers to normal I/O done to HDOS devices via HDOS device drivers. Naturally, a program may control its "private devices" (ones not suitable for device drivers) in any way it pleases.

In general, the sequence for doing file I/O is to issue an "open" SCALL (.OPENR, .OPENW, or .OPENU) to HDOS, supplying HDOS with the file descriptor as an ASCII string. HDOS will parse the string, load the device driver (if necessary), and open the file. When you issue the "open" SCALL, you supply a channel number from −1 (i.e., 377Q) to 5. This channel number must not already be in use. This means that you may open a maximum of seven files simultaneously.

Once the file has been opened, you can perform I/O by using the .READ, .WRITE, and .POSIT SCALLs. Make these requests by suppling HDOS with the channel number of the file you want read or written. After the initial open, you no longer need the file descriptor string. Should you suddenly need that file name, say to issue an error message, HDOS provides the .NAME SCALL to recall the file name used when that channel was opened.

All disk file I/O is done in multiples of 256 bytes, the system sector size. As many bytes as desired may be transferred at one time, so long as the count is an integer multiple of 256. HDOS normally performs I/O in a sequential fashion. For example, if your program is reading from a disk file one sector (256 bytes) at a time, the first read will return sector 0, the next read sector 1, etc. For each open channel, HDOS maintains a "sector cursor", which indicates which sector in the file is next to be read or written. HDOS does provide the facility, via .POSIT, to randomly read and write sectors to/from a disk file by changing the value of this "sector cursor".

When you are done with the file, use the .CLOSE SCALL, once more supplying the channel number. HDOS will close the file and thus make that channel available for another open.

NOTE: Although channel −1 can be used as a general purpose I/O channel, its use should normally be avoided. It is already open when your program is started; you must close it before you can open a file on it. Also, channel −1 will be cleared (see the .CLEAR SCALL) if you use the .LINK SCALL. Thus any file open for write on channel −1 at that time will be lost.

Part 4

PRECAUTIONS

We have discussed earlier in this document that the HDOS system does not provide any hardware protection, and thus is vulnerable to errors in assembly language programs. This segment discusses the "Do's and Dont's" of assembly language programming in more detail.

Memory Precautions

The two most important areas of memory precautions are: respect for the user program area, and maintenance of the stack.

USER MEMORY AREA

A user program must never write into memory outside of its domain. This "domain" consists of the memory area from 042200A (USERFWA) to the LWA of the user program area. When your program is first loaded, this LWA is set to the end of your program and its declared data areas (via DS, DW, or DB; not EQU). The ".SETTOP" SCALL is available to adjust this limit. User programs may adjust this limit as often as they like (see the .SETTOP SCALL documentation). Note that HDOS may use all memory after this limit for a storage area, which is going to cause trouble if your routine also tries to access it.

STACK MAINTENANCE

Since the HDOS system uses interrupts, and requires interrupts to handle the console, the H17 disk, and the H47 disk, your program may be interrupted at any time. You must always maintain a valid stack pointer, with at least 64 free bytes on the top of that stack. If you plan to fill the system stack area, then you should ORG your program above 042200A and set the stack pointer higher, giving yourself and HDOS a bigger stack. HDOS does not use a separate stack; it uses the top of the user program stack.

I/O Precautions

As we discussed earlier, I/O precautions consist of keeping your INs and OUTs to yourself. Don't disturb the H17, and don't disturb the console ports! Also, be careful what you do with the front panel ports, either directly or indirectly via PAM-8. These ports control the clock interrupts, which are necessary for the H17 device driver.

Interrupt Precautions

When you are using interrupts, you must use only the available vectors, which are 4 (if you are not using a real-time clock), 5 (if you are not using an H47), and 6. You may also use 2, if you will not be using DBUG. Before you enable your interrupting device, install the service vector in the appropriate ".UIVEC" location.

Most importantly, turn off the interrupting device so it cannot issue any more interrupts before you either return control to HDOS, or CTL-Z out of the program. If an interrupt occurs when your program is no longer there to service it, the operating system, and possibly the information on your diskettes, will be destroyed!

Since console and clock interrupts may occur at any time, your program should not turn off interrupts (via DI) except for very short periods of time.

Finally, HDOS uses the clock interrupts, so you should not overlay its interrupt vector. Programs desiring clock service should use all means possible to make do with the interrupt counter (PAM-8's .TICCNT). If you absolutely must have clock interrupts, save the address in the clock vector, install your own vector, and have your service routine exit the interrupt by jumping to the HDOS vector address. HDOS uses the clock interrupts for H17 timings; disturbing it might cause your motors to keep spinning, prematurely wearing the motors. Or worse, you might defeat the H17 driver's head settle delays, and cause a bad sector to be written.

CPU Precautions

This precaution should be familiar to all assembly language programmers: Don't let the CPU execute undefined memory locations. Should such a thing occur, it is unlikely that your disks will be damaged, due to some safeguards built into the system. However, you should immediately re-boot, and not try to warm-start the system, since the CPU may have damaged tables in memory. Remember, the HDOS system uses a sophisticated linked-allocation scheme to handle disk files. Damaging that table, or damaging the directory or allocation areas on the disk, can cause all files on that disk to become lost, not just one or two!

If you are debugging a program which consistently vectors into undefined memory locations, then it is best to use write-protect labels on the disks. Then, when you crash, you can quickly restart by using PAM-8 to start at the HDOS cold-start address, 040100A. Entering at this address should return you to HDOS command mode. Do this only if you have your disks write-protected. Otherwise it is too risky. Usually, when your program runs wild, the CPU ends up at some high memory location where you don't have any memory. The computer hardware generates Ø for nonexistent memory, so you will quickly run through a long string of NOP's, until you wrap from 377377A to 000000A, which is the master clear restart address for PAM-8. If you display the PC and find it set to your high memory address, then you probably took this "circumpolar" route into PAM-8.

Debugging Hints

The best way to debug programs is to ORG them above DBUG, and test them using DBUG. After entering DBUG, use the LOAD command to load in the program under test. You can then break-point and single step through your program. Do not single step through an HDOS SCALL, or you may damage the disk.

After the program seems to be working, ORG it back down to 042200A, (or wherever) and reassemble.

Part 5

RESIDENT SCALLs

This segment covers those HDOS service requests (called SCALLs) which are permanently resident in memory. The use of these SCALLs will not cause an overlay to be loaded.

In general, a SCALL (Sys CALL) consists of a

```
RST    7
```

instruction followed by a byte containing the request number. Most SCALLs require that some registers be set up before the call. Likewise, most may alter the registers, so a program should save any registers which it wants to preserve.

The ASM assembler has a special opcode for SCALLS:

```
SCALL    code
```

where "code" is the number of the request. This statement generates the equivalent of

```
DB    377Q,code
```

We recommend that you use the HOSDEF.ACM file to include these definitions. In general, it is advisable to use the recommended symbol definitions for all references to HDOS, and include them in one or more XTEXT decks. This will make programming easier for you, and guarantee compatibility with future releases. Although we will make every effort to keep binary compatibility, we may need to revert to "assembly language compatability," in which case you may have to change some HDOS symbol values and reassemble.

.EXIT

```
***     EXIT - EXIT USER PROGRAM.
*
*       EXIT IS CALLED TO RETURN CONTROL TO THE SYSTEM COMMAND
*       PROGRAM.
*
*       MVI     A,FLAG          (see below)
*       SCALL   .EXIT
*
*       FOR EITHER EXIT, THE CONTROL CHARACTER VECTORS
*       (SET BY .CTLC) ARE CLEARED.
*
*       IN ADDITION, THE ABORT EXIT RESETS THE DISK AND
*       CONSOLE I/O DRIVERS.
*
*       ENTRY   (A)         = FLAG ( Ø = NORMAL, 1 = ABORT )
*       EXIT    -IF-        [ SYSTEM DISK IS STILL MOUNTED ]
*                                   -or-
*                           [ STAND-ALONE IS SET ]
*               -THEN-      EXIT TO "SYSCMD.SYS"
*               -ELSE-      EXIT TO REBOOT CODE
*
```

The .EXIT SCALL is the proper way for a program to return control to HDOS. In any mode, .EXIT will close all open I/O channels. This action is equivalent to that of the .CLEAR SCALL. It is best for a program to close or clear its own channels before incurring .EXIT, as future releases may differ in this action.

It should not be necessary for a program to use abort exit unless some process was being used which affected the state of the console or disk I/O ports. The use of such processes is not recommended.

If SYØ: has been dismounted and the STAND-ALONE flag is not set, HDOS exits to re-boot. If the STAND-ALONE flag has been set and no disk is mounted on SYØ:, or SYSCMD.SYS is not found on the disk mounted on SYØ:, HDOS exits to re-boot. Thus, the only way for a program to return to the command level once SYØ: has been dismounted and remounted is for the STAND-ALONE flag to have been previously set via the SET command, and for the disk mounted on SYØ: to have SYSCMD.SYS on it.

```
**      EXAMPLES:
ALDONE  MVI     A,0             FLAG NORMAL EXIT
        SCALL   .EXIT
ABTXIT  MVI     A,1             FLAG ABORT EXIT
        SCALL   .EXIT
```

NOTE: We do not encourage this re-entrance to HDOS, and it may not be supported in future releases.

.SCIN — System Console INput

```
***     SCIN - SYSTEM CONSOLE INPUT.
*
*       SCIN TAKES A SINGLE CHARACTER FROM THE CNSOLE INPUT
*       BUFFER, IF ANY ARE AVAILABLE.
*
* L1    SCALL   .SCIN
*       JC      L1                CHARACTER NOT READY
*
*       ENTRY   NONE
*       EXIT    'C' SET IF NO CHARACTER
*               'C' CLEAR IF CHARACTER
*                (A) =CHARACTER
*       USES    A,F
```

This command is relatively obvious, and is also explained in the HEATH HDOS Software Reference Manual. Note that you can use the .CONSL SCALL to set console mode bits.

```
**      EXAMPLES:

RDCHAR  SCALL   .SCIN              TRY TO READ CHARACTER
        JC      RDCHAR             NONE READY YET
        RET                         EXIT, (A) = CHARACTER
```

NOTE: Detailed examples of .SCIN are shown in the HEATH HDOS Manual.

.SCOUT — System Console OUTput

```
***      SCOUT - SYSTEM CONSOLE OUTPUT.
*
*        SCOUT OUTPUTS A SINGLE CHARACTER TO THE CONSOLE. CURSOR
*        POSITIONING IS KEPT TRACK OF. A "NL" CHARACTER
*        INDICATES A NEW LINE. "CR" AND "LF" CHARACTERS SHOULD
*        NOT BE USED.
*
*        MVI      A,CHAR
*        SCALL    .SCOUT
*
*        ENTRY    (A) = CHARACTER
*        EXIT     (A) = CHARACTER
*        USES     NONE
```

This command is relatively obvious, and is also explained in the HEATH HDOS Software Reference Manual.

```
 **       EXAMPLES:

        MVI     A,'*'
        SCALL   .SCOUT            TYPE AN ASTERISK ON THE CONSOLE
```

NOTE: Further examples of .SCOUT are shown in the HEATH HDOS Manual.

.READ — Read From File

Use the .READ SCALL to read data from an open channel. The channel must already have been opened via a .OPENR or .OPENU SCALL (except for channel −1, as noted previously).

Currently, all device I/O under HDOS (with the exception of the console, via the .SCIN and .SCOUT calls) is "block mode". This means that you must read or write to the device in multiples of 256 bytes. If you cannot fill the last block, you should pad it out with zero bytes. The last block in all HDOS source files is padded out to 256 characters with 00 bytes.

The quoted C in the following example indicates the carry flag. This SCALL, as in all others in HDOS, returns with the carry flag set if an error or abnormal condition has occurred. The most common "error" for the .READ command is "end-of-file". The convention used above and throughout this document is that exit conditions which are predicated on the setting of a flag are discussed directly under that flag, indented one space. Thus, the (BC) register pair contains the unused byte count if, and only if, the 'C' flag is set. If 'C' is clear, then all of the bytes were read, and (BC) contains garbage. Thus, the (BC) and (DE) registers contain meaningful information only when an error condition occurred, which is normally an "end-of-file". The error codes returned by HDOS are defined in Part 7. This is simply a condensation of the error messages discussed in the HEATH HDOS Software Reference Manual. Note that you can use the .ERROR SCALL to look up an explanatory message.

```
***     READ - PROCESS READ SCALL.
*
*       READ PROCESSES READ SCALLS. IF A SERIAL DEVICE, PASS TO
*       DRIVER. IF A STORAGE DEVICE, HANDLE STORAGE MAPPING.
*
*       MVI     A,CHAN
*       LXI     B,COUNT         MUST BE MULTIPLE OF 256
*       LXI     D,ADDR
*       SCALL   .READ           READ DATA FROM FILE
*
*       ENTRY   (A) = I/O CHANNEL/NUMBER
*               (B) = COUNT OF 256-BYTE BLOCKS TO TRANSFER
*               (C) = 0
*               (DE) = DATA ADDRESS
*       EXIT   'C' CLEAR IF ALL OK
*              'C' SET IF ERROR
*               (A) = ERROR CODE
*               (BC) = UNUSED TRANSFER COUNT
*               (DE) = NEXT UNUSED ADDRESS
*       USES    ALL
```

NOTE: All read operations must be for integer multiples of 256 bytes. Thus, the last sector in a file may have been padded with 00 bytes. All ASCII (coded) files in HDOS are zero-byte filled in the last sector (if they need it). A 00 byte is considered a NULL character, and should always be ignored when encountered in an ASCII file.

```
**      EXAMPLES:

        .
        .
        .
READ    MVI     A,1         READ FROM ALREADY OPEN CHANNEL 1
        LXI     B,256       READ ONE SECTOR
        LXI     D,BUFFER
        SCALL   .READ       READ IT
        JC      READ1       ERROR
        LXI     B,256       READ 256 BYTES
        JMP      READ2

*       HAVE ERROR. SEE IF EOF, OR SOMETHING WORSE

READ1   CPI     EC.EOF      SEE IF JUST EOF
        JNE     ERROR       HAVE SERIOUS ERROR
        STA     EOFFLG      FLAG HAVE SEEN EOF
        LXI     H,256       (HL) = ORIGINAL STARTING COUNT
        MOV     A,L
        SUB     C
        MOV     C,A
        MOV     A,H
        SBB     B
        MOV     B,A         (BC) = 256-REMCNT = AMOUNT READ

*       READ COMPLETE. (BC) = BYTES AVAILABLE

READ2   ....
        .
        .
BUFFER  DS      256          SECTOR BUFFER
```

.WRITE — Write to Open File

```
***     WRITE - PROCESS WRITE SCALL.
*
*       MVI     A,CHAN
*       LXI     B,COUNT         MUST BE MULTIPLE OF 256
*       LXI     D,ADDR
*       SCALL   .WRITE          WRITE DATA TO CHANNEL
*
*       ENTRY   (A) = CHANNEL #
*               (BC) = DATA COUNT
*               (DE) = DATA ADDRESS
*       EXIT    'C' CLEAR IF ALL OK
*               'C' SET IF ERROR
*                (BC) = UNUSED TRANSFER COUNT
*                (DE) = NEXT UNUSED ADDRESS
*                (A) = ERROR CODE
*       USES     ALL
```

The .WRITE SCALL is very similar to the .READ call, except that it writes the data to the file. Once again, the count in (BC) must be an integral multiple of 256. The most typical error returned by .WRITE is "NO ROOM ON MEDIA".

NOTE: All write operations must be for integer multiples of 256 bytes. Thus, the last sector in a file may have to be filled out to 256 bytes. All ASCII (coded) files in HDOS are zero-byte filled in the last sector (if they need it). A 00 byte is considered a NULL character, and should always be ignored when encountered in an ASCII file.

```
**      EXAMPLES:

WRIDAT  MVI     A,1             CHANNEL 1 ALREADY OPEN
        LXI     B,512           WRITE 512 BYTES
        LXI     D,BUFFER
        SCALL   .WRITE          WRITE IT
        JC      ERROR           SERIOUS ERROR
        .
        .
        .
BUFFER  DS      512             BUFFER AREA FOR WRITE
```

.PRINT — Print Line on System Console

```
***      PRINT - PRINT CONSOLE LINE.
*
*        PRINT CAUSES A CODED LINE TO BE PRINTED AT THE CONSOLE.
*
*        LXI     H,LINEADDR
*        SCALL   .PRINT
*
*        THE LAST CHARACTER IN THE LINE SHOULD HAVE THE
*        200Q BIT SET.
*
*        ENTRY   (HL) = LINE ADDRESS
*        EXIT    (HL) = LWA OF MESSAGE +1
*        USES    A,F,H,L
```

.PRINT is an efficient and convenient way to print lines on the system console. Another good way is to use the subroutine "$TYPTX", as shown in Part 8. Note that the parity bit (bit 200Q) is set over the last character to be printed to notify the end-of-line to HDOS. Remember, use the NL character (012Q, same as LF) for a CRLF sequence. HDOS will automatically insert the required number of PAD characters for the console. If you prefer, you can include the NULL (00) character in a print line. It is ignored, does not cause a delay in console output, and thus cannot be used as a PAD character.

```
**      EXAMPLES:

        .
        .
        LXI     H,MSGA          TYPE OUT STARTUP MESSAGE
        SCALL   .PRINT
        .
        .
PROMPT  LXI     H,MSGB          TYPE OUT PROMPT MESSAGE
        SCALL   .PRINT
REACHA  SCALL   .SCIN           READ REPLY....
        .
        .
        .
MSGA    DB      12Q,'SET OPTIONS:'
        DB      12Q
        DB      12Q,'HELP - TYPE THIS LIST'
        DB      12Q,'CRASH - DESTROY DISK SURFACE'
        DB      12Q+200Q                NEW LINE, END OF PRINT

MSGB    DB      12Q,'YOUR COMMAND?',' '+200Q
        .
        .
```

.CONSL — Set Console Mode Bits

```
***     CONSL - SET AND CLEAR CONSOLE FLAGS.
*
*       CONSL IS CALLED TO SET, CLEAR, OR READ BITS IN THE
*       VARIOUS CONSOLE FLAGS.
*
*       THE CALLER PASSES AN INDEX INTO THE PROPER FLAG, A
*       MASK TO INDICATE THE AFFECTED BITS, AND A SET OF NEW
*       VALUES FOR THOSE BITS.
*
*       INDEX =
*
*       0       I.CSLMD
*       1       I.CONTY
*       2       I.CUSOR
*       3       I.CONWI
*       4       I.CONFL
*
*       ENTRY   (A) = INDEX
*               (B) = NEW VALUES
*               (C) = MASK ('1' BIT FOR EVERY BIT TO CHANGE)
*       EXIT    'C' CLEAR IF NO ERROR
*                (A) = NEW VALUE
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

The .CONSL SCALL is used to read and write the console control bits and bytes. These bytes are available directly in memory, but we recommend that you access them via the .CONSL command to guarantee synchronization and upward compatibility with future releases.

The caller supplies HDOS with three values: the index of the byte to be read and/or written, the bits to be altered, and the new bit values. The technique of supplying a "bits-affected" mask and a "new value" pattern allows you to alter just one bit in a byte, without having to know the values of the other bits in the byte. Since the console is an interrupt-responsive device, this also avoids synchronization probems. There are five bytes which can be read and/or written via the .CONSL function.

```
I.CSLMD - Console Mode


I.CSLMD EQU     0               I.CSLMD IS FIRST BYTE

CSL.ECH EQU     10000000B       SUPPRESS ECHO
CSL.WRP EQU     00000010B       WRAP LINES AT WIDTH
CSL.CHR EQU     00000001B       OPERATE IN CHARACTER MODE
```

These three bits are used to affect the mode in which HDOS handles characters typed at the console. They are documented in more detail in the HDOS Software Reference Manual.

```
I.CONTY - Console Type


I.CONTY EQU     1                       I.CONTY IS 2ND BYTE

CTP.BKS EQU     10000000B               TERMINAL PROCESSES BACKSPACES
CTP.MLI EQU     00100000B               MAP LOWER CASE TO UPPER ON INPUT
CTP.MLO EQU     00010000B               MAP LOWER CASE TO UPPER ON OUTPUT
CTP.2SB EQU     00001000B               TERMINAL NEEDS TWO STOP BITS
CTP.BKM EQU     00000010B               MAP BKSP (UPON INPUT) TO RUBOUT
CTP.TAB EQU     00000001B               TERMINAL SUPPORTS TAB CHARACTERS
```

The bits in the I.CONTY byte are used to describe the console's hardware characteristics. These bits are all discussed under the SET command section in the HEATH HDOS Software Reference Manual.

```
I.CUSOR - Console Cursor Position

I.CUSOR EQU     2                 I.CUSOR IS 3RD BYTE
```

The I.CUSOR byte contains the current cursor position of the console terminal cursor. Immediately after a New-Line, this byte contains 001.

```
I.CONWI - Console Width

I.CONWI EQU     3                 I.CONWI IS 4TH BYTE
```

The I.CONWI byte contains the current console width. This value is documented under the SET command in the HDOS Software Reference Manual. In brief, when the cursor reaches this value, HDOS automatically generates an NL. You can effectively disable this option by setting the width to 255.

```
I.CONFL - Console Flags

I.CONFL EQU    4                I.CONFL IS 5TH BYTE

CO.FLG  EQU    00000001B        CTL-O FLAG
CS.FLG  EQU    10000000B        CTL-S FLAG
```

The I.CONFL byte contains the current setting of the console CTL-O and CTL-S bytes. A user program may find it useful to note that the user has typed CTL-S or CTL-O. In addition, your program may want to clear the CTL-O flag immediately before an input prompt is typed, so that the typing of the prompt is guaranteed.

NOTE: If the CTL-S flag is set, and your program issues a character to the console (via .SCOUT or .PRINT) then your program will hang up in HDOS waiting for the CTL-S flag to clear. There is no way to do a "conditional" character type-out. Programs which do not want to hang up must check the CTL-S flag before every .PRINT or .SCOUT, and trust to luck that your user doesn't type the CTL-S between the .CONSL and the .SCOUT.

```
**      EXAMPLES:


*       SET CHARACTER MODE, NO ECHO

        MVI     A,I.CSLMD       (A) = BYTE INDEX
        MVI     B,CSL.ECH+CSL.CHR       SET BOTH BITS
        MVI     C,CSL.ECH+CSL.CHR       AFFECT BOTH BITS
        SCALL   .CONSL

*       SET MAP LOWER CASE TO UPPER, CLEAR BACKSPACE ON 'RUBOUT'
 KEY
        MVI     A,I.CONTY       (A) = BYTE INDEX
        MVI     B,CTP.MLI+CTP.MLO       SET MAP LOWER CASE BITS
        MVI     C,CTP.MLI+CTP.MLO+CTP.BKS       SET MAP, CLEAR
 BKS
        SCALL   .CONSL

*       READ CONSOLE CURSOR POSITION
        MVI     A,I.CUSOR
        MVI     C,0             AFFECT NO BITS, (B) MEANINGLESS
        SCALL   .CONSL          AFFECT NOTHING, JUST GET NEW
                                (SAME AS OLD) VALUE
        CPI      1              SEE IF CURSOR OVER COLUMN 1
        .
        .

*       SET CONSOLE WIDTH

        MVI     A,I.CONWI
        MVI     B,80              SET 80 COLUMNS
        MVI     C,377Q            AFFECT FULL BYTE
        SCALL   .CONSL            SET WIDTH
```

.CLRCO — Clear Console Buffer

```
***       CLRCO - CLEAR CONSOLE BUFFERS.
*
*         CLRCO CLEARS THE CONSOLE TYPE-AHEAD BUFFER.
*         CTL-O AND CTL-S FLAGS ARE ALSO CLEARED.
*
*         ENTRY   NONE
*         EXIT    NONE
*         USES    ALL
```

The .CLRCO SCALL is used to clear the console buffer, and the console CTL-S and CTL-O flags. HDOS contains a console "type-ahead" buffer, so the user may type commands before a program asks to read from the console. All typed text is stored in the type-ahead buffer; the .SCIN SCALL reads the characters from the buffer. The special control characters; CTL-A, CTL-B, and CTL-C; are not stored in the type-ahead buffer; but instead, cause an interrupt to a user service routine (if you set one up via the .CTLC SCALL). Often, a user has typed a partial line before he typed the CTL-C (or CTL-A or CTL-B). You can use the .CLRCO function to clear out any unwanted type-ahead.

NOTE: Issuing the .CLRCO function does not cause a New-Line to be sent to the console. The user is given no indication that the characters he may have typed in have been discarded. Your program should issue a new prompt immediately after the .CLRCO function, to make things clear to the user.

```
**        EXAMPLE: CLEANUP AFTER CTL-C


          (Part 6 discusses intercepting CTL-C's)

*         ASSUME CONTROL PASSES HERE AT CTL-C

CCHIT     LXI     H,CCHITA         TYPE ^C
          SCALL   .PRINT           ACKNOWLEDGE CTL-C, SETUP NEW LINE
          SCALL   .CLRCO           CLEAR TYPE AHEAD
          .
          .

CCHITA    DB      '^C',212Q                  ^C WITH NEW-LINE
```

.LOADO — Load Overlay

```
***     LOADO - LOAD SPECIFIED OVERLAY
*
*       LOADO LOADS THE OVERLAY SPECIFIED THROUGH THE INDEX
*
*               OVERLAY         INDEX
*             ------------------------
*               HDOSOVLØ          Ø
*               HDOSOVL1          1
*
*       ENTRY   (A)      = OVERLAY INDEX
*       EXIT    (PSW)    = 'C' CLEAR IF NO ERROR
*                          'C' SET IF ERROR
*                           (A)   = ERROR CODE
*       USES    ALL
```

The .LOADO system call is used to force an overlay load. Before you dismount the system disk (SYØ:),you must load both overlays "0" and "1". Quite simply, once the system disk has been dismounted, subsequent diskettes are only data diskettes. That is, the overlays may not be loaded from them. A sample program fragment follows, and further examples may be found in Part 8.

NOTE: This system call may generate an error if enough memory is not available for both your program and the indicated overlay. In such a case, either the size of your program must somehow be reduced, or you will have to forego the overlay loading.

```
**      Examples:


        MVI     A,OVL0
        SCALL   .LOADO          LOAD 'HDOSOVLØ.SYS'
        JC      FATAL           Error on attempted load

        MVI     A,OVL1
        SCALL   .LOADO          LOAD 'HDOSOVL1.SYS'
        JC      FATAL          Error on attempted load
```

.VERS — HDOS Version Number

```
***       VERS - RETURN HDOS VERSION NUMBER
*
*         VERS RETURNS THE HDOS VERSION NUMBER AS A ONE-BYTE
*         BCD NUMBER. A DECIMAL IS ASSUMED BETWEEN THE HIGH
*         AND LOW ORDER NYBBLES.
*
*         ENTRY   NONE
*         EXIT    (PSW)     = 'C' CLEAR IF NO ERROR
*                              (A) = VERSION NUMBER
*                             'C' SET IF ERROR ( VERS < 1.5)
                               (A) = ERROR CODE ( EC.ILC)
*         USES    A,F
```

The .VERS system call returns the current version number of HDOS. The primary use of this system call is to ascertain under which version of HDOS the program is running. If the program determines that the version does not support these new calls, it may exit gracefully with an error message. Versions earlier than 1.5 may be distinguished because they will return an invalid system call.

The version number is returned as one BCD byte. That is, version 1.5 will return 21, or 25Q, or 015H. (See the HDOS common deck listing for an example of the definition format).

```
        SCALL   .VERS
        JC      BADVER          No version system call
        CPI     VERS
        JNZ     BADVER          Invalid version
        .
        .
BADVER  LXI     B,MESSAG
        SCALL   .PRINT
        .
        .
MESSAG  DB      12Q,'This version of HDOS does not support'
        DB      'the required system calls.',12Q+200Q
```

Part 6

OVERLAID SCALLs

This section discusses those HDOS SCALLs which are resident in the overlaid portion of HDOS.

Overlay Management

When an overlaid request is issued, HDOS checks the status of the overlay area. If it is already in memory, the request is processed. If it is not in memory, HDOS then checks the LWA of your user program. If there is not enough room past the end of your user LWA, then some of the last bytes in the user memory area are swapped to disk. Then the overlay is loaded and the function performed. After the function is performed, HDOS will reload any paged-out portion of your program.

This overlay structure affects assembly language programmers in two ways:

1. Arguments passed to HDOS for SCALL requests should not be too high up in user memory, as they might get swapped to the disk when the overlay is loaded.

2. In program where you plan on doing many overlay SCALLs, try to limit your memory requests so that the overlay area can remain resident. Currently, the best way of doing this is to use the .SETTOP SCALL to find the maximum allowable allocation; then subtract the overlay size, kept in HDOS's low memory (see Part 7). Also subtract a 10-byte margin of error. You may request memory up to this new limit without causing the overlay area to be swapped out.

File Names

Since many overlaid SCALLs require file names as arguments, this is a good time to discuss HDOS file names.

In general, when you supply a file name as an argument to a SCALL, you point to an ASCII string containing the file descriptor just as the user would have typed it. The line should be terminated with a delimiter of some sort, usually a comma, blank, or ØØ character. For example, the following are examples of valid file names:

```
DB        'SYØ:MYFILE.TMP',Ø
DB        'TEMP',Ø
DB        'BASIC.SAV,'     (',' delimits name)
```

Of course, these names are all shown being assembled into the program. You might just as well have read them from the user's console, or generated the names somehow. They must not have embedded ØØ bytes or blanks in the names.

Also note that some of the examples shown do not specifiy an extension or a device. All SCALLs that take file names as arguments also require a default block. This block is a 6-byte area, containing the default device specification and a default extension specification. A typical default block is:

```
DB        'SYØTMP'
```

which yields a default device of SYØ: and extension of TMP. Another common default block is:

```
DB        'SYØ',Ø,Ø,Ø
```

which indicates that there is no default extension. File descriptors not specifying a name will generate a file with a null extension.

.OPENR — Open File for Read

```
***     OPENR - OPENR SCALL PROCESSOR.
*
*       OPENR IS CALLED TO OPEN A CHANNEL FOR READ.
*
*       THE CALLER SUPPLIES A FILE NAME, A DEFAULT BLOCK
*       FOR THE DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'           DEFAULT DEVICE
*       DB      'XXX'           DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .OPENR SCALL to open files for read access. This means that you may then read the file, but HDOS will not allow any write requests to it. You may open an individual file for read access on as many channels as you wish.

The channel number supplied must be a legal one (−1 to 5), and must not already have a file open on it.

HDOS will not allow any one file to be open for both read and write at the same time, nor may any one file be open for write on more than one channel. Attempting to do so will cause a "usage conflict" error. This means that you may not open a file via .OPENR if it is already open for write (or update) on another channel.

```
**        EXAMPLES:

*         OPEN PRE-DETERMINED FILE NAME ON CHANNEL 1

          MVI      A,1             CHANNEL 1
          LXI      D,DEFALT        POINT TO DEFAULT BLOCK
          LXI      H,FNAME         POINT TO FILE DESCRIPTOR
          SCALL    .OPENR          OPEN FOR READ
          JC       ERROR           SOME ERROR
          .
          .

*         READ FILE NAME FROM USER, OPEN ON CHANNEL 2

          LXI      H,MSGA
          SCALL    .PRINT          PROMPT HIM
          LXI      H,BUFFER
REA1      SCALL    .SCIN
          JC       REA1            NO CHARACTER
          MOV      M,A             STORE IN MEMORY
          INX      H
          CPI      012Q            SEE IF NEWLINE (USER HIT CR KEY)
          JNE      REA1            NOT YET
          DCX      H
          MVI      M,0             TERMINATE LINE WITH ØØ, INSTEAD
                                   OF 12Q

          LXI      H,BUFFER
          LXI      D,DEFALT        POINT TO DEFAULT BLOCK
          MVI      A,2             CHANNEL 2
          SCALL    .OPENR          OPEN FILE
          JC       ERROR
          .
          .

MSGA      DB       12Q,'FILE NAME?',' '+200Q
DEFALT    DB       'SYØTMP'        DEFAULT DEVICE AND EXTENSION
BUFFER    DS       20              FILE NAME BUFFER
FNAME     DB       'SY1:MYFILE.NEW',Ø      FILE NAME FOR CHANNEL 1
          .
          .
```

.OPENW — Open File for Write

```
***     OPENW - OPEN FILE FOR WRITE
*
*       OPENW IS CALLED TO OPEN A CHANNEL FOR WRITE.
*
*       THE FILE IS ENTERED IN THE CHANNEL TABLE, BUT NOT ON THE
*       DISK. IT WILL BE ENTERED IN THE DIRECTORY AT CLOSE TIME.
*
*       THE CALLER SUPPLIES A FILE NAME, A DEFAULT BLOCK FOR THE
*       DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'           DEFAULT DEVICE
*       DB      'XXX'           DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .OPENW SCALL to open a file for writing. When HDOS processes the .OPENW SCALL, the file is opened with a "temporary" name, which does not appear in the directory. When the channel is closed, HDOS will then enter the name in the directory. If any previous file by that name existed, it will be deleted at that time. This procedure has three implications:

1. You cannot modify an existing file by means of the .OPENW SCALL, .OPENW is intended for creating new files, or replacing old ones.

2. If you are replacing an existing file, there must be enough free space to hold both the new version and the old one, as the old one will not be deleted until the new one is closed. You might want to manually delete (via .DELETE) the old one first.

3. If you do not properly close the channel, the new file will be lost. This is intended as a safety factor; a previously existing file by that name will not be destroyed until the new one has been successfully completed. If you should start to write a file by some name, then realize that you already have a useful file by that name, you can CTL-Z out and still retain the old file.

HDOS will not allow any one file to be open for both read and write at the same time, nor may any one file be open for write on more than one channel. If you attempt to do so you will cause a "usage conflict" error. This means that you cannot open a file with .OPENW if it is already open for write or update, or if it is open for read.

The examples shown above for .READ are applicable to .WRITE as well. The following example illustrates opening a file on a non-disk device, "AT:". Note that exactly the same procedure is followed. In fact, in the above example where the user types in a file name, he may just as well have typed in "TT:" or "AT:" for a device specification.

```
**       EXAMPLES:

         .
         .
         MVI      A,3           OPEN ON CHANNEL 3
         LXI      D,DEFALT      POINT TO DEFAULT BLOCK
         LXI      H,FNAME
         SCALL    .OPENW
         JC       ERROR         ERROR
         .
         .
         .
DEFALT   DB       'SYØ',Ø,Ø,Ø   UNUSED, BUT REQUIRED
FNAME    DB       'AT:',Ø       NAME AND EXTENSION MEANINGLESS
         .
         .
```

.OPENU — Open File for Update

```
***     OPENU - OPEN FILE FOR UPDATE.
*
*       OPENU IS CALLED TO OPEN A CHANNEL FOR UPDATE.
*
*       UPDATE IS JUST LIKE READ, BUT THE FILE MAY BE WRITTEN
*       ALSO.
*
*       THE CALLER SUPPLIES A FILE NAME, A DEFAULT BLOCK FOR THE
*       DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'           DEFAULT DEVICE
*       DB      'XXX'           DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use .OPENU to open a file for update. This means that a previously existing disk file is opened for both read and write. When opened, the file is positioned at sector Ø.

If the channel is positioned over an existing sector and you issue a .WRITE, then that sector will be re-written. If the channel is positioned at the end of the file, the file will be extended. You can use the .POSIT SCALL to position the channel at the end of the file. Thus, the .OPENU and .POSIT combination allows you to append information onto an existing file.

NOTE: Always close a file that was opened for update. Failure to do so causes undefined results. Failing to close the channel properly can also cause "orphaned" sectors, which are not being used by a file, nor are they in the free list. HDOS will automatically recover these orphans when the disk is next mounted (or booted) and return them to the free list.

The examples used for .OPENR on Page 31 also apply to .OPENU . Of course, there are some differences:

1. The file opened must already exist.

2. The file must reside on a mass storage device, which can be both read and written (i.e., not write protected).

.CLOSE — Close Channel

```
***     CLOSE - PROCESS CLOSE SCALL.
*
*       CLOSE PROCESSING DEPENDS UPON THE FILE AND DEVICE TYPE.
*
*       FOR A WRITE/DIRECTORY TYPE, THE DIRECTORY IS SEARCHED
*       FOR A PREVIOUS ENTRY. IF FOUND, IT IS DELETED. THE NEW
*       ENTRY IS THEN INSERTED.
*
*       FOR A UPDATE/DIRECTORY TYPE, THE PREVIOUS ENTRY IS
*       UPDATED.
*
*       FOR ALL FILES, THE DRIVER IS CALLED WITH THE DC.CLO
*       FUNCTION. THE CHANNEL IS RELEASED.
*
*       ENTRY   (A) = CHANNEL #
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = CODE
*       USES    ALL
```

Use the .CLOSE SCALL to close a channel when you are done with it. Always close all the channels your program has opened, with two exceptions:

1. HDOS enters your program with channel −1 open on your program load file. If you do not use this channel you need not close it -- HDOS will perform the close on it automatically.

2. Scratch files which were created via .OPENW, which are no longer needed need not be closed. See ".CLEAR", Page 57.

```
**      EXAMPLES:

        .
        .
        MVI     A,1
        SCALL   .CLOSE          CLOSE CHANNEL 1
        JC      ERROR
        MVI     A,2
        SCALL   .CLOSE          CLOSE CHANNEL 2
        JC      ERROR           IF ERROR
        .
        .
```

.RENAME — Rename Disk File

```
***      RENAME - PROCESS RENAME FUNCTION.
*
*        RENAME RENAMES A FILE ON A DIRECTORY DEVICE.
*
*        * NOTE * RENAME DOES NOT CHECK TO SEE IF THE NEW NAME
*        ALREADY EXISTS--THIS IS CURRENTLY THE RESPONSIBILITY
*        OF THE CALLER !
*
*        ENTRY    (HL) = NAME STRING
*                 (DE) = DEFAULT BLOCK
*                 (BC) = NEW NAME STRING
*        EXIT     'C' CLEAR IF OK
*                 'C' SET IF ERROR
*                  (A) = CODE
*        USES     ALL
```

Use the .RENAME SCALL to change the name of a file on disk. A renaming is considered a form of writing on a file, so the same "usage conflict" restrictions apply: the file to be renamed must not be open on another channel. Two other restrictions exist:

1. A file with the "new name" must not already exist on that device. RENAME unfortunately does not check for this currently, so you must check yourself by trying to .OPENR the file, before doing the .RENAME . Currently, RENAME will allow you to create two files on a disk with the same name. The results of this will be disastrous.

2. The "name string" and the "new name string" must both specify the same device (SYØ:, SY1:, SY2:, DKØ:, DK1:, or DK2:). Alternatively, both files may use the default device, which may be any valid HDOS drive name.

NOTE: The default block device and extension applies only to the old file name, not to the new name. The new file name must be fully specified, including device, file name, and extension, if there is to be one.

```
**      EXAMPLES:

*       RENAME 'SY1:SORT.ASM' TO 'SY1:SORT.BAK'
        LXI     B,NEWNAM
        LXI     D,DEFALT
        LXI     H,OLDNAM
        SCALL   .RENAME
        JC      ERROR
        .
        .
NEWNAM  DB      'SY1:SORT.BAK',Ø        NO DEFAULTS ALLOWED
OLDNAM  DB      'SORT',Ø        USE DEFAULT DEVICE AND EXTENSION
DEFALT  DB      'SY1ASM',Ø      DEFAULT DEVICE AND EXTENSION
```

.DELETE — Delete Disk File.

```
**      DELETE - PROCESS DELETE COMMAND.
*
*       ENTRY    (HL) = NAME STRING
*                (DE) = DEFAULT BLOCK
*       EXIT     'C' CLEAR IF OK
*                'C' SET IF ERROR
*                 (A) = CODE
*       USES     ALL
```

Use the .DELETE SCALL to delete a disk file. The format of the call is similar to that of .OPENW, except that no channel number is specified. Note that deleting a file is considered a form of writing, so the file must not be open on any channel for reading or writing, as that would cause a "file usage conflict".

```
**      EXAMPLE: DELETE FILE "SYØ:TEMP.TMP"
DELTEMP LXI     H,NAME
        LXI     D,DEFALT
        SCALL   .DELETE
        JC      ERROR
        .
        .
NAME    DB      'TEMP.TMP',Ø       FILE NAME
DEFALT  DB      'SYØXXX'           DEFAULT DEVICE, DEFAULT EXTENSION
```

.CHFLG — Change File Flags

```
***     CHFLG - CHANGE FILE FLAGS.
*
*       CHFLG IS CALLED TO CHANGE THE FILE DESCRIPTION FLAGS
*       FOR A MASS STORAGE FILE. ONLY CERTAIN FLAGS MAY BE
*       CHANGED:
*
*       FLAG    BIT      MEANING
*
*       DIF.SYS 200Q     IS SYSTEM FILE
*       DIF.LOC 100Q     LOCKED FOR CHANGE (SETABLE ONLY)
*       DIF.WP  040Q     IS WRITE PROTECTED
*
*       CHFLG WILL REFUSE THE OPERATION IF THE DIF.LOC BIT
*       IS SET.
*
*       ENTRY   (B) = NEW BIT VALUES
*               (C) = CHANGE MASK (BIT SET FOR EVERY BIT
*                     TO REPLACE FROM (B))
*               (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = FILE NAME
*       EXIT    'C' CLEAR, CHANGE DONE
*               'C' SET, ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .CHFLG SCALL to change the attribute flags on a file. These flags are discussed in detail in the Heath HDOS "Software Reference Manual," under the program "FLAGS". The arguments are similar to the .OPEN SCALLs. Note that a two-byte "bits to effect" and "new bit values" scheme is used, just as described earlier for the .CONSL SCALL.

NOTE: You can use the .CHFLG SCALL to set the DIF.LOC (LOCKed) flag on a file, but you cannot use to clear the flag. Once the DIF.LOC flag is set, no other flag changes may be made, including clearing the DIF.LOC flag. If the file is not write-protected (DIF.WP not set), you can copy it to a temp file, delete the old LOCKed version, and rename the temp file back. If the file is both LOCKed and write protected, then it is there "forever", or until the volume is re-initialized via INIT.

```
**      EXAMPLE: WRITE-PROTECT 'OUTPUT.DAT'

WRIPRO  MVI     B,DIF.WP        EFFECT WRITE PROTECT
        MVI     C,DIF.WP        SET WRITE PROTECT
        LXI     D,DEFALT
        LXI     H,NAME
        SCALL   .CHFLG
        JC      ERROR           ERROR
        .
        .
        .
NAME    DB      'SY1:OUTPUT.DAT',Ø      FILE NAME
```

.POSIT — Position Disk File

```
***     POSIT - POSITION FILE.
*
*       LXI     B,POSITION
*       MVI     A,CHANNEL NUMBER
*       SCALL   .POSIT
*
*       ENTRY   (A) = CHANNEL NUMBER
*               (BC) = SECTOR NUMBER TO POSITION BEFORE
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*                (A) = EC.EOF IF OFF END
*                  (BC) = SECTORS UNSKIPPED (REMAINDER OF COUNT)
*                  FILE POSITIONED AT EOF
*       USES    ALL
```

Use the .POSIT SCALL to position the "channel cursor". Since each read or write on a file (via a channel) must transfer in sector (or multi-sector) lots, the channel's current position in the file is simply the logical sector number next to be read or written. This sector number has no relation to actual physical sector numbers; the first sector in a file is sector 0, the next is sector 1, the last sector in an n sector file is n-1.

NOTE: The .POSIT SCALL positions the channel (file) before the specified sector. Thus, a .POSIT to Ø positions the channel before sector Ø, so that a one-sector read will return sector Ø. To position the channel at the end of a file, .POSIT to n, where n is the number of sectors in the file. If you do not know how long the file is, .POSIT to 65535 (377377A), verify that an EC.EOF error was flagged, and then compute the file size as SIZE = 65535-(BC).

Thus, when a file is first opened, via .OPENR, .OPENW, or .OPENU, it is positioned at sector Ø. The first read or write of m sectors will read or write sectors Ø through m-1. This is a normal sequential access. For example, when reading, each one-sector read will return the next sector in the file.

You can use the .POSIT SCALL to set this "sector cursor" at any spot in the file. Positioning a file at sector Ø is the equivalent of rewinding it. A file may be positioned at its end, so a read will return end-of-file, and a write will extend the file. It may not be positioned after the last sector+1 in an attempt to extend the file size--files may be extended only via .WRITE SCALLs.

Note that the .POSIT SCALL strengthens the similarity between .OPENW and .OPENU. If you have opened a file via .OPENW, you may use .POSIT to position the channel cursor to allow you to re-write any sector in the file, at any time. If you then wish to add some more sectors to the end, you can position to the end of the file and .WRITE some more. Also note that you can change the value of any byte or bytes in a file open for write or update by positioning before the proper sector, reading the sector, modifying it, repositioning over it again, and writing the sector back.

```
**      EXAMPLE 1: REWINDING A FILE AFTER READING IT

        <OPEN AND READ A FILE ON CHANNEL 1>

        .
        .
        LXI     B,Ø             BEFORE SECTOR 0
        MVI     A,1             CHANNEL 1
        SCALL   .POSIT          POSITION
        JC      ERROR

        <READ THE FILE OVER AGAIN>
```

```
**      EXAMPLE 2: REPLACING A SECTOR IN A FILE BEING WRITTEN

        <OPEN THE FILE VIA .OPENW >
        MVI     A,2             CHANNEL 2
        LXI     B,256*10        WRITE 10 SECTORS
        LXI     D,BUFFER        FROM BUFFER
        SCALL   .WRITE
        JC      ERROR
        LXI     B,1             PREPARE TO RE-WRITE 2ND SECTOR
                                IN FILE
        MVI     A,2
        SCALL   .POSIT
        JC      ERROR
        MVI     A,2             CHANNEL 2
        LXI     B,256
        LXI     D,BUFFER2
        SCALL   .WRITE          WRITE DIFFERENT DATA
        MVI     A,2             CHANNEL 2
        LXI     B,-1            POSITION AT END OF FILE
        SCALL   .POSIT          WILL RETURN EOF ERROR
        CPI     EC.EOF
        JNE     ERROR           OTHER ERROR

        < FURTHER WRITES WILL APPEND TO END OF FILE >
```

```
**      EXAMPLE 3: INCREMENTING BYTE 7423 IN FILE "DATA.RAW"

        MVI     A,0             OPEN ON CHANNEL 0
        LXI     D,DEFALT
        LXI     H,FNAME
        SCALL   .OPENU          OPEN FOR UPDATE
        JC      ERROR

*       POSITION FOR READ

        LXI     H,7423          (H) = SECTOR NUMBER,
                                (L) = BYTE INDEX
        MOV     C,H
        MVI     B,0             (BC) = SECTOR NUMBER
        PUSH    H               SAVE (HL)
        MVI     A,0
        SCALL   .POSIT           POSITION
        JC      ERROR

*       READ SECTOR INTO WORK BUFFER

        MVI     A,0             (A) = CHANNEL
        LXI     B,256
        LXI     D,BUFFER
        SCALL   .READ
        JC      ERROR

*       INCREMENT BYTE

        POP     B               (B) = SECTOR, (C) = BYTE INDEX
        LXI     H,BUFFER
        MOV     A,C             (A) = BYTE INDEX
        CALL    $DADA           ADD (A) INTO (HL) (ROUTINE IN
                                H17 ROM)
        INR     M               INCREMENT BYTE IN BUFFER

*       POSITION FOR RE-WRITE

        MOV     C,B
        MVI     B,0              (BC) = SECTOR NUMBER
        MVI     A,0
        SCALL   .POSIT
        JC      ERROR
```

```
*       WRITE BACK OUT

        MVI     A,0               (A) = CHANNEL
        LXI     B,256
        LXI     D,BUFFER
        SCALL   .WRITE
        JC       ERROR

*       CLOSE FILE

        MVI      A,0
        SCALL    .CLOSE
        JC       ERROR
        .
        .
        .
DEFALT  DB       'SYØ',Ø,Ø,Ø
FNAME   DB       'SYØ:DATA.RAW',Ø
BUFFER  DS       256
```

.DECODE — Decode File Name

```
***     DECODE - PROCESS DECODE SCALL.
*
*       DECODE DECODES THE SUPPLIED FILE NAME
*       INTO A BLOCK IN THE FORM:
*
*       DS      1               RESERVED
*       DS      2               DEVICE NAME
*       DS      1               DEVICE UNIT
*       DS      8               FILE NAME
*       DS      3               FILE EXTENSION
*       DS      4               RESERVED
*
*       ENTRY   (BC) = AREA FOR TABLE TO BE WRITTEN
*               (DE) = DEFAULT LIST
*               (HL) = NAME ADDRESS
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .DECODE SCALL to decode an ASCII file descriptor into a formatted block. The fields in the block contain the device, unit, name, and extension values from the file descriptor. The fields are 0 filled. This function is useful for programs which wish to in some way examine the file name, extension or device specification without going to the work of manually cracking the file descriptor. For example, if your program reads a file descriptor from the console, then wants to know if the extension is "ABC", it might use the .DECODE SCALL to crack out the extension field.

```
**      EXAMPLE: SEE IF USER TYPED DEVICE CODE 'TT:'

        <READ LINE FROM CONSOLE INTO *LINE* >

        LXI     B,BUFFER
        LXI     D,DEFALT
        LXI     H,LINE
        SCALL   .DECODE         DECODE SUPPLIED FILE NAME
        JC      ERROR           ILLEGAL NAME
        LXI     B,3             COMPARE 3 BYTES
        LXI     D,BUFFER+1      (DE) = SUPPLIED DEVICE NAME
        LXI     H,TTSTR
        CALL    $COMP           COMPARE STRINGS (ROUTINE
                                IN H17 ROM)
        JNE     NOTTT           FILE NOT ON TT:
        JMP     GOTTT           NAME DID SPECIFY TT:

BUFFER  DS      19              ROOM FOR REPLY DATA
LINE    DS      80              USER-SUPPLIED FILE NAME
TTSTR   DB      'TT',0          NAME AND UNIT IF DEVICE WAS 'TT:'
```

.NAME — Get File Name from Channel

```
***     NAME - PROCESS NAME SCALL.
*
*       THE NAME SCALL RETURNS THE DEVICE, FILE NAME, AND
        FILE EXTENSION OF AN OPEN CHANNEL.
*
*       THE INFORMATION IS OBTAINED FROM THE CHANNEL TABLE,
*       WHICH WAS SET UP UPON FILE OPEN.
*
*       ENTRY   (A) = CHANNEL NUMBER
*               (DE) = ADDRESS FOR DEVICE AND EXTENSION (DEFAULT
*                      BLOCK FORMAT)
*               (HL)  ADDRESS FOR NAME (8 CHARACTERS, FOLLOWED
*                      BY 00 BYTE)
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .NAME SCALL to recall the name which was supplied to HDOS when the channel was opened. This is mainly used when an error message is prepared after HDOS has flagged an error on a channel operation.

```
**      EXAMPLE: ERROR PRINTING PROGRAM.
*
*       THIS ROUTINE PRINTS AN ERROR MESSAGE FOR A FILE
*       OPERATION GONE WRONG.
*
*       ENTRY   (A) = ERROR NUMBER
*               (CURCHAN) = CHANNEL NUMBER USED IN
*                           FAILED OPERATION
*       EXIT    ...

ERROR   PUSH    PSW             SAVE ERROR CODE
        LXI     H,ERRORA
        SCALL   .PRINT          PRINT 'ERROR - '
        POP     PSW             (A) = CODE
        MVI     H,07Q           BELL AFTER ERROR CODE
        SCALL   .ERROR          PRINT ERROR
        LXI     H,ERRORB
        SCALL   .PRINT          PRINT ' ON FILE '
        LDA      CURCHAN        (A) = CHANNEL NUMBER
        LXI      D,ERRDFB       (DE) = ADDRESS FOR DEVICE AND
                                       EXTENSION
        LXI      H,ERRNAM       (HL) = ADDRESS FOR NAME
        SCALL    .NAME          GET FILE NAME

*       MANIPULATE DEVICE, NAME, AND EXTENSION INTO
*       PRESENTABLE FORMAT, AND PRINT ON CONSOLE.

        .
        .
        .

ERRORA  DB      012Q,'ERROR -',' '+200Q
ERRORB  DB      ' ON FILE',' '+200Q
ERRDFB  DS      6               DEVICE AND EXTENSION FOR BAD FILE
ERRNAM  DS      9               NAME FOR BAD FILE
```

.LINK — Link to Another Program

```
***     LINK - PROCESS LINK SCALL.
*
*       LINK LOADS IN AND RUNS ANOTHER PROGRAM. THE OPEN FILES,
*       SYSTEM TABLES, AND STACK ARE NOT DISTURBED.
*
*       ENTRY   (HL) = ADDRESS OF PROGRAM FILE DESCRIPTOR
*       EXIT    TO LINKED PROGRAM, IF OK
*                (A) UNCHANGED
*                (SP) = VALUE AT 'LINK' SCALL
*               TO CALLER IF ERROR
*                'C' SET
*                (A) = ERROR CODE
*       USES    ALL
```

The .LINK SCALL is used to pass control to another program.

```
**      EXAMPLE: TRANSFER CONTROL TO PROGRAM 'CLEANUP.ABS'

XFER    MVI     A,-1            CHANNEL -1 OPEN ON LOADED FILE

*       GET DEVICE WE WERE LOADED FROM, SO THAT WE CAN
*       RUN 'CLEANUP.ABS' FROM THAT SAME DISK

        LXI     D,DEVCODE               AREA FOR DEVCODE
        LXI     H,BUFFER        PUT NAME INTO SCRATCH AREA
        SCALL   .NAME

*       BUILD NAME TO LINK TO...

        LXI     B,XFERAL        (BC) = NUMBER OF BYTES TO MOVE
        LXI     D,XFERA         FROM XFERA
        LXI     H,DEVCODE+3     PUT AFTER DEVICE SPECIFICATION
        CALL    $MOVE           PUT NAME AFTER DEVICE (ROUTINE
                                IN H17 ROM)

*       CALL PROGRAM

        LXI     H,DEVCODE
        SCALL   .LINK           TRY TO EXECUTE IT
        JC      ERROR           FAILED

XFERA   DB      ':CLEANUP.ABS',0        NAME
XFERAL  EQU     *-XFERA         AMOUNT TO MOVE

DEVCODE DS      3+XFERAL        ROOM FOR ENTIRE FILE SPECIFICAION
        .
        .
```

.CTLC — Set Up Handlers for Control Characters

```
***     CTLC - SET CONTROL CHARACTER ADDRESS
*
*       THE .CTLC SCALL IS USED TO SET UP HANDLING FOR
*       THE CONTROL CHARACTERS CTL-A, CTL-B, AND CTL-C.
*
*       A SEPARATE ADDRESS IS SPECIFIABLE FOR EACH CHARACTER. IF
*       AN ADDRESS OF 0 IS SPECIFIED, PROCESSING OF THAT
*       CHARACTER IS SUSPENDED.
*
*       THE PROCESS ADDRESS MUST BE > 255A.
*
*       ENTRY   (A) = CONTROL CHARACTER WHOSE PROCESS ADDRESS IS
*                       TO CHANGE (CTL-A, CTL-B, OR CTL-C)
*               (HL) = NEW ADDRESS (=0 TO CLEAR PROCESSING)
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES.   A,F,H,L
```

The .CTLC SCALL allows you to set up interrupt service subroutines for the handling of CTL-A, CTL-B, and CTL-C. You may set up a separate service routine for each character.

When a service routine has been set up and the specified character has been struck, your routine will be entered at interrupt-time, with interrupts enabled.

Upon entry to your routine, the registers B, C, D, E, H, and L have whatever contents were in them at the time of the control character interrupt. The stack contains:

((SP)+0) = Return Address into HDOS
((SP)+2) = Interrupted PSW
((SP)+4) = Interrupted PC

Your routine can do some interrupt-time work (having saved the registers first, of course) and then do a RET to HDOS, in which case HDOS will take care of the rest. Or, if you wish, you may ignore the HDOS return address and jump back into your program's command loop, or whatever.

```
**      EXAMPLE 1: SETTING AN 'INTERRUPT OCCURRED' FLAG

        LXI     H,CCINT         SET UP CTL-C INTERRUPT PROCESSOR
        MVI     A,003           (A) = CTLC
        SCALL   .CTLC           SET UP CTL C
        .
        .
        .
LOOP    SCALL   .SCIN
        JNC     GOTONE          GOT A CHARACTER
        LDA     CCHIT
        ANA     A
        JZ      LOOP            NO CTL-C HIT
        JMP     PROCC           PROCESS CTL-C
        .
        .
        .

*       CTL-C CAUSES THIS ROUTINE TO BE ACTIVATED

CCINT   MVI     A,1             PSW IS ALREADY SAVED
        STA     CCHIT           SET CC HIT
        RET                     RETURN TO INTERRUPTED CODE VIA
HDOS
        .
        .
        .
CCHIT   DB      0               SET =1 WHEN CTL-C TYPED


**      EXAMPLE 2: RETURNING CONTROL TO MAIN COMMAND LOOP.

        .
        .
        LXI     H,CBHIT
        MVI     A,002           (A) = CTLB
        SCALL   .CTLB
        .
        .
START   LXI     SP,STACK        CLEANUP STACK
LOOP    .                       DO WHATEVER WE DO...
        .
        .

*       ENTERED HERE IF CTL-B HIT

CBHIT   JMP     START           RESTART COMMAND LOOP
```

.SETTOP — Set Top of User Memory

```
***     SETTOP - SET TOP OF USER MEMORY.
*
*       SETTOP IS CALLED TO NOTIFY THE SYSTEM OF A NEW
*       MEMORY LIMIT ADDRESS. IF NECESSARY, THE OVERLAYS
*       WILL BE UNLOADED.
*
*       ENTRY   (HL)    = NEW ADDRESS
*       EXIT    (PSW)   = 'C' CLEAR IF OK
*                         'C' SET IF TOO HIGH
*                          (A) = ERROR CODE
*                          (HL) = MAXIMUM ADDRESS
*       USES    ALL
```

Use the .SETTOP SCALL to set the top of the user memory area. Since HDOS sets the top of memory to the last address in your program, most programs do not need to use .SETTOP. Programs which need large buffer areas should not declare them with DS statements, since the generated binary file will be excessively large. Instead, they should define the areas via EQU statements, and use the .SETTOP SCALL to request the needed space from HDOS.

Note that, by requesting the impossible (65535 bytes), you can determine the actual maximum memory available from the error return.

If you want to request maximum memory but avoid swapping the overlays, the approved method is to first load both overlays (see .LOADO) and then make the memory request.

```
**      EXAMPLE 1: GETTING MAXIMUM MEMORY WITHOUT SWAPPING

        MVI     A,OVL0          LOAD OVERLAY 0
        SCALL   .LOADO
        JC      ERROR
*
        MVI     A,OVL1          LOAD OVERLAY 1
        SCALL   .LOADO
        JC      ERROR
*
        LXI     H,-1            CAUSE DELIBERATE ERROR
        SCALL   .SETTOP         . .TO GET MAX IN (HL)
        LXI     D,-10           SUBTRACT 'SLOP' FACTOR
        DAD     D
        SHLD    MAXMEM          SAVE MAX MEMORY
        SCALL   .SETTOP         NOW ASK FOR THE MAX ALLOWABLE
        JC      ERROR           SHOULD NOT HAPPEN
        .
        .
MAXMEM  DS      2               MEMORY LIMIT
```

```
**        EXAMPLE 2: GETTING ABSOLUTE MAXIMUM MEMORY
*                   ( ENTER HERE WITHOUT LOADING OVERLAYS )

          LXI      H,-1                IMPOSSIBLE AMOUNT
          SCALL    .SETTOP             WILL FAIL. .
          SHLD     MAXMEM              SAVE RESULT
          SCALL    .SETTOP             ASK FOR MAX
          JC       ERROR               SHOULD NOT HAPPEN
          .
          .
MAXMEM    DS       2                   MEMORY LIMIT

*         REMEMBER THAT IF THE .SETTOP IS SUCCESSFUL,
*         THE CONTENTS OF (HL) ARE MEANINGLESS.
```

.CLEAR — Clear I/O Channel

```
***     CLEAR - CLEAR I/O CHANNEL.
*
*       CLEAR IS CALLED TO CLEAR AN I/O CHANNEL. IF THE
*       CHANNEL IS CLOSED, NO ACTION IS PERFORMED. IF THE
*       CHANNEL IS OPEN, IT IS FLAGGED CLOSED. THE RESULTS
*       OF THIS OPERATION DEPEND UPON THE TYPE OF FILE:
*
*       OPEN FOR                 ACTION
*
*       READ            SAME AS .CLOSE
*
*       WRITE           FILE IS FORGOTTEN. ANY WRITTEN
*                       DISK BLOCKS ARE RESTORED TO THE
*                       FREE POOL.
*
*       UPDATE          REPLACED SECTORS REMAIN REPLACED.
*                       APPENDED SECTORS ARE LOST UNTIL
*                       NEXT BOOT. FILE STAYS AT PREVIOUS
*                       LENGTH.
*
*       THE DEVICE DRIVER IS NOT INFORMED OF THE CLOSING.
*
*       SCALL   .CLEAR
*
*       ENTRY   (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Use the .CLEAR SCALL to free up a channel without closing it. The actions discussed above merely document the current results of the .CLEAR SCALL; they may not stay the same for future releases. There is only one supported use of the .CLEAR SCALL, which is to delete temp files. A temp work file is created by means of a .OPENW SCALL. You need not worry about name conflicts, as any pre-existing file will not be disturbed by the .OPENW. However, when you are done, you do not want to .CLOSE then .DELETE the file, since this would destroy any pre-existing file by that name. In that case, use .CLEAR on the channel to free up the channel and release the used disk sectors.

```
**      EXAMPLE: CREATING, USING, AND DESTROYING A SCRATCH FILE

        MVI     A,0             USE CHANNEL 0
        LXI     D,DEFALT
        LXI     H,SCRNAME
        SCALL   .OPENW          OPEN SCRATCH FILE
        JC      ERROR
        .
        .
        < WRITE DATA ON SCRATCH FILE >
        .
        .
        MVI     A,0
        LXI     B,0
        SCALL   .POSIT          REWIND SCRATCH FILE
        .
        .
        < READ DATA FROM SCRATCH FILE >
        .
        .
        MVI     A,0
        SCALL   .CLEAR          DESTROY SCRATCH FILE
        .
        .
        .
SCRNAME DB      'SYØ:TEMP.TMP',0        ANY PRE-EXISTING TEMP.TMP
                                        NOT AFFECTED
```

.ERROR — Print Error Message

```
***     ERROR - PRINT ERROR MESSAGE.
*
*       ERROR IS CALLED TO PRINT AN ERROR MESSAGE.
*
*       THE HDOS SYSTEM RETURNS ERROR CODE NUMBERS WHEN
*       IT DETECTS AN ERROR. THE ERROR FUNCTION MAY BE
*       USED TO TYPE AN ALPHABETIC EXPLANATION OF THE ERROR.
*
*       THE ERRORS ARE STORED IN THE FILE 'ERRORMSG.SYS'
*       ON THE SYSTEM DISK, ONE MESSAGE PER LINE. THE
*       LINES LOOK LIKE:
*
*       NNNTEXT
*
*       FOR EXAMPLE,
*
*       002END OF MEDIA
*
*       IF THE ERROR MESSAGE FILE CANNOT BE READ, OR THE
*       MESSAGE DOES NOT APPEAR, THE ERROR IS TYPED AS
*
*       'SYSTEM ERROR # NNN'
*
*       ENTRY   (A) = ERROR CODE
*               (H) = TRAILING CHARACTER (TYPED AFTER MESSAGE)
*       EXIT    NONE
*       USES    ALL
```

Use the .ERROR SCALL to look up an error message in the system error message file "ERRORMSG.SYS". Since HDOS returns all error messages as numbers, this function allows you to easily inform the user, in English, just what went wrong. Also note that if you have a program which needs to generate a large number of messages, you can add them to ERRORMSG.SYS. Of course, this is not a supported use of .ERROR, and may not work with future releases.

An example of the use of the .ERROR SCALL is shown in the example of the .NAME SCALL.

.LOADD — Load Device Driver

```
***      LOADD - LOAD DEVICE DRIVER
*
*        LOADD LOADS THE SPECIFIED DEVICE DRIVER
*
*        ENTRY   (HL)    = DEVICE DRIVER DESCRIPTOR STRING
*        EXIT    (PSW)   = 'C' CLEAR IF OK
*                          'C' SET IF ERROR
*                            (A)   = ERROR CODE
*
*        USES    ALL
```

Use the LOADD system call to load a specified device driver in memory without opening a file on the device. Like the .LOADO system call, this system call is not to be used when SYØ: is to be dismounted. If a device driver is not in memory at the time SYØ: is dismounted (because it was not loaded and no channel is currently open on the device), subsequent references to the device will generate unknown device errors. Examples of the use of this call are found in Part 8, and below.

```
                 LXI      H,DEVICE
                 SCALL    .LOADD
                 JC       ERROR
                 .
                 .
        DEVICE   DB       'LP:',0
```

.MOUNT — Mount Disk

```
***     MOUNT - MOUNT DISK
*
*       MOUNT DISK ON SPECIFIED UNIT OF SELECTED DEVICE
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                          (A) = ERROR CODE
*                         'C' CLEAR IF NO ERROR
*                          'Z' CLEAR IF AN ABORT
*       USES    ALL
```

Use the .MOUNT system call to mount additional devices. The device specified must not have a volume already mounted on it. If it does, a successful dismount must be issued before a .MOUNT may be processed. The devices currently supported are SYØ:, SY1:, SY2:, DKØ:, DK1:, and DK2:. This system call also prints a message informing the user that a volume has been mounted, as per the format of the HDOS "MOUNT" command. This call will also verify that the disk structure is not corrupt. If the disk structure is corrupt, the volume will not be successfully mounted and an error will be returned. If you do not want the message, you may issue the .MONMS system call.

For a detailed example of .MOUNT, see Part 8.

.DMOUN — Dismount Disk

```
***     DMOUN - DISMOUNT DISK
*
*       DISMOUNT DISK ON SELECTED DRIVE
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                          (A)   = ERROR CODE
*
*       USES    ALL
```

Use the .DMOUN system call to dismount diskettes. After the volume has been successfully dismounted, it will also print a message verifying that the volume has, in fact, been dismounted. The device to be dismounted must have a volume currently mounted. If it does not, .DMOUN returns an error.

If the volume to be dismounted is the system volume, you must observe several precautions. Since HDOS will no longer be able to overlay itself, the overlays must be loaded via the .LOADO SCALL. Similarly, device drivers not currently in memory at the time of the dismount will be considered nonexistent. Subsequent references to drivers so marked will generate unknown device errors. You may load a device driver by opening a channel on the device, or by ".LOADD"ing it. Even if the current program will not use the device, the device must be loaded before you dismount the system volume if any subsequent programs are to use it.

Before you dismount a disk, you must clear all of the I/O channels open to that disk. Remember that the program itself is left open on channel -1 (377Q), and this channel must be closed before you dismount the system disk (SYØ:).

.MONMS — Mount Disk with No Message

```
***      MONMS - MOUNT/NO MESSAGE
*
*        MOUNT SPECIFIED UNIT OF SELECTED DEVICE WITHOUT ISSUING
*        A MOUNT MESSAGE
*
*        ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*        EXIT    (PSW)   = 'C' SET IF ERROR
*                           (A)  =ERROR CODE
*                          'C' CLEAR IF NO ERROR
*                           'Z' CLEAR IF AN ABORT
*        USES    ALL
```

In versions of HDOS later than Version 1.5, the .MONMS system call is identical to the .MOUNT system call, except that .MONMS prints no mount message. In the future, this may not be the case. In all likelihood, this will be changed to a "quick" mount which neither prints the message nor verifies the disk structure. Therefore, we do not recommend that you use .MONMS for the present.

For a detailed example, see Part 8.

.DMNMS — Dismount Disk with No Message

```
***     DMNMS - DISMOUNT DEVICE/NO MESSAGE
*
*       DISMOUNT SELECTED UNIT OF SPECIFIED DEVICE WITHOUT
*       ISSUING MESSAGE
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                          (A)   = ERROR CODE
*       USES     ALL
```

The .DMNMS system call is virtually identical to the .DMOUN call, except for the printing of the dismount message. In future releases, this will probably be changed to some form of quick dismount. For the present, we do not suggest that you use it.

.RESET — Mount/Dismount Disk

```
***      RESET - RESET DEVICE
*
*        RESET THE SPECIFIED UNIT OF THE SELECTED DEVICE
*        BY ISSUING A DISMOUNT FOLLOWED BY A MOUNT.
*        THE DEVICE NAME SHOULD BE IN THE SAME FORMAT AS
*        THAT EXPECTED BY MOUNT AND DISMOUNT.
*
*        ENTRY    (HL)     = ADDRESS OF DEVICE SPECIFICATION
*        EXIT     (PSW)    = 'C' CLEAR IF NO ERROR
*                            'C' SET IF ERROR
*                             (A)   = ERROR CODE
*
*        USES     ALL
```

If a disk is mounted on the specified device, the .RESET SCALL is equivalent to a .DMOUN, a disk change prompt, and a .MOUNT. You must verify the prompt by opening the drive door (so that the diskette stops spinning) and then closing it. If no volume is mounted, the call is equivalent to a .MOUNT and no prompt message is printed. This call may be interrupted between the .MOUNT and .DMOUN by means of control characters (^C, etc.), in which case the device will be left without a volume mounted on it.

For a detailed example, see Part 8.

Part 7

HDOS SYMBOL DEFINITIONS

As we stressed in earlier sections, there are numerous advantages to using symbolic definitions when you are interfacing to the operating system. This section lists suggested common decks which contain the appropriate symbolic definitions.

To obtain access to these definitions, simply insert the pseudo-ops

```
XTEXT   HOSDEF           XTEXT ASCII
XTEXT   HOSEQU           XTEXT ECDEF
```

into the initial statements of your program. This will cause the assembler to process, as required, the statements in the file HDOS.ACM, thus defining those symbols for that assembly.

Note that the assembler will not normally list the contents of any file read by XTEXT. However, by using the

```
LON   C
```

pseudo-op, or the

```
/LON:C
```

switch when you are using the assembler, you can cause a listing of all files read by XTEXT to be written to the listing file.

Recommended HDOS Common Deck Contents

RECOMMENDED HOSDEF.ACM CONTENTS

```
HOSDEF   SPACE   3,10
**       HOSDEF - DEFINE HOS PARAMETER.
*

VERS     EQU     2*16+0          CURRENT VERSION = 2.0

         ORG     0

*        RESIDENT FUNCTIONS

.EXIT    DS      1               EXIT (MUST BE FIRST)
.SCIN    DS      1               SCIN
.SCOUT   DS      1               SCOUT
.PRINT   DS      1               PRINT
.READ    DS      1               READ
.WRITE   DS      1               WRITE
.CONSL   DS      1               SET CLEAR CONSOLE OPTIONS
.CLRCO   DS      1               CLEAR CONSOLE BUFFER
.LOADO   DS      1               LOAD AN OVERLAY
.VERS    DS      1               RETURN HDOS VERSION NUMBER
```

```
*       HDOSOVL0.SYS FUNCTIONS

                ORG     40A
.LINK           DS      1       LINK (MUST BE FIRST)
.CTLC           DS      1       CTL-C
.OPENR          DS      1       OPENR
.OPENW          DS      1       OPENW
.OPENU          DS      1       OPENU
                DS      1       RESERVED
.CLOSE          DS      1       CLOSE
.POSIT          DS      1       POSITION
.DELETE         DS      1       DELETE
.RENAME         DS      1       RENAME
.SETTOP         DS      1       SETTOP
.DECODE         DS      1       NAME DECODE
.NAME           DS      1       GET FILE NAME FROM
                                CHANNEL
.CLEAR          DS      1       CLEAR CHANNEL
                DS      1       RESERVED
.ERROR          DS      1       LOOKUP ERROR
.CHFLG          DS      1       CHANGE FLAGS
                DS      1       RESERVED
.LOADD          DS      1       LOAD DEVICE DRIVER


*       HDOSOVL1.SYS FUNCTIONS

        ORG     200Q     .MOUNT (MUST BE FIRST)

.DMOUN  DS      1        DISMOUNT
.MONMS  DS      1        MOUNT/NO MESSAGE
.DMNMS  DS      1        DISMOUNT/NO MESSAGE
.RESET  DS      1        RESET = DISMOUNT/MOUNT OF UNIT

*       OVERLAY INDICES
*

OVL0    EQU     0        HDOSOVL0.SYS
OVL1    EQU     1       HDOSOVL1.SYS
```

RECOMMENDED HOSEQU.ACM CONTENTS

```
HOSEQU  SPACE   4,10
**      HDOS Equates
*

USERFWA EQU     42200A          FIRST WORD ADDRESS OF USER
                                PROGRAMS
STACK   EQU     42200A          SYSTEM STACK ADDRESS

ESVAL   SPACE   4,10
**      SYSTEM RAM CELL DEFINITIONS.
*
*       THESE VALUES ARE LOCATED IN THE RESERVED HDOS RAM AREA.
        ORG     040277A

S.DATE  DS      9               SYSTEM DATE (IN ASCII)
S.DATC  DS      2               CODED DATE
        DS      4               RESERVED
S.HIMEM DS      2               HARDWARE HIGH MEMORY ADDRESS+1

S.SYSM  DS      2               FWA RESIDENT SYSTEM

S.USRM  DS      2               LWA USER MEMORY

S.OMAX  DS      2               MAX OVERLAY SIZE FOR SYSTEM


**      THE FOLLOWING SYMBOLS ARE USED BY THE .CONSL SCALL.

CSL.ECH EQU     10000000B       SUPPRESS ECHO
CSL.WRP EQU     00000010B       WRAP LINES AT WIDTH
CSL.CHR EQU     00000001B       OPERATE IN CHARACTER MODE

I.CSLMD EQU     0               CONSOLE MODE

CTP.BKS EQU     10000000B       TERMINAL PROCESSES BACKSPACES
CTP.MLI EQU     00100000B       MAP LOWER CASE TO UPPER ON INPUT
CTP.MLO EQU     00010000B       MAP LOWER CASE TO UPPER ON OUTPUT
CTP.2SB EQU     00001000B       TERMINAL NEEDS TWO STOP BITS
CTP.BKM EQU     00000010B       MAP BKSP (UPON INPUT) TO RUBOUT
CTP.TAB EQU        00000001B    TERMINAL SUPPORTS TAB CHARACTERS

I.CONTY EQU     1               S.CONTY IS 2ND BYTE
I.CUSOR EQU     2               S.CUSOR IS 3RD BYTE
I.CONWI EQU     3               S.CONWI IS 4TH BYTE

CO.FLG  EQU     00000001B       CTL-O FLAG
CS.FLG  EQU     10000000B       CTL-S FLAG

I.CONFL EQU     4               S.CONFL IS 5TH BYTE
```

RECOMMENDED ASCII.ACM CONTENTS

```
ASCII   SPACE   2,10
**      ASCII CHARACTER EQUIVALENCES

CR      EQU        15Q           CARRIAGE RETURN
LF      EQU        12Q           LINE FEED
NULL    EQU     0   200Q         PAD CHARACTER
BELL    EQU        7             BELL CHARACTER
RUBOUT  EQU        177Q
BKSP    EQU        10Q           CTL-H
C.SYN   EQU        26Q           SYNC
C.STX   EQU        2             STX
QUOTE   EQU        47Q
TAB     EQU        11Q
ESC     EQU        33Q
NL      EQU        12Q           NEW LINE (HDOS SYSTEMS)
ENL     EQU        NL+200Q       NL + END-OF-LINE FLAG
FF      EQU        14Q           FORM FEED
CTLA    EQU        01Q           CTL-A
CTLB    EQU        02Q           CTL-B
CTLC    EQU        03Q           CTL-C
CTLD    EQU        04Q           CTL-D
```

RECOMMENDED ECDEF.ACM CONTENTS

```
ECDEF   SPACE   3,10
**      ERROR CODE DEFINITIONS.

        ORG     0
EC.HIN  DS      1       HDOS ISSUE NUMBER
EC.EOF  DS      1       END OF FILE
EC.EOM  DS      1       END OF MEDIA
EC.ILC  DS      1       ILLEGAL SYSCALL CODE
EC.CNA  DS      1       CHANNEL NOT AVAILABLE
EC.DNS  DS      1       DEVICE NOT SUITABLE
EC.IDN  DS      1       ILLEGAL DEVICE NAME
EC.IFN  DS      1       ILLEGAL FILE NAME
EC.NRD  DS      1       NO ROOM FOR DEVICE DRIVER
EC.FNO  DS      1       CHANNEL NOT OPEN
EC.ILR  DS      1       ILLEGAL REQUEST
EC.FUC  DS      1       FILE USAGE CONFLICT
EC.FNF  DS      1       FILE NAME NOT FOUND
EC.UND  DS      1       UNKNOWN DEVICE
EC.ICN  DS      1       ILLEGAL CHANNEL NUMBER
EC.DIF  DS      1       DIRECTORY FULL
EC.IFC  DS      1       ILLEGAL FILE CONTENTS
EC.NEM  DS      1       NOT ENOUGH MEMORY
EC.RF   DS      1       READ FAILURE
EC.WF   DS      1       WRITE FAILURE
EC.WPV  DS      1       WRITE PROTECTION VIOLATION
EC.WP   DS      1       DISK WRITE PROTECTED
EC.FAP  DS      1       FILE ALREADY PRESENT
EC.DDA  DS      1       DEVICE DRIVER ABORT
EC.FL   DS      1       FILE LOCKED
EC.FAO  DS      1       FILE ALREADY OPEN
EC.IS   DS      1       ILLEGAL SWITCH
EC.UUN  DS      1       UNKNOWN UNIT NUMBER
EC.FNR  DS      1       FILE NAME REQUIRED
EC.DIW  DS      1       DEVICE IS NOT WRITABLE (OR WRITE LOCKED)
EC.UNA  DS      1       UNIT NOT AVAILABLE
EC.ILV  DS      1       ILLEGAL VALUE
EC.ILO  DS      1       ILLEGAL OPTION
EC.VPM  DS      1       VOLUME PRESENTLY MOUNTED ON DEVICE
EC.NVM  DS      1       NO VOLUME PRESENTLY MOUNTED
EC.FOD  DS      1       FILE OPEN ON DEVICE
EC.NPM  DS      1       NO PROVISIONS MADE FOR
                        REMOUNTING MORE DISKS
EC.DNI  DS      1       DISK NOT INITIALIZED
EC.DNR  DS      1       DISK IS NOT READABLE
EC.DSC  DS      1       DISK STRUCTURE IS CORRUPT
EC.NCV  DS      1       NOT CORRECT VERSION OF HDOS
EC.NOS  DS      1       NO OPERATING SYSTEM MOUNTED
EC.IOI  DS      1       ILLEGAL OVERLAY INDEX
EC.OTL  DS      1       OVERLAY TOO LARGE
```

HDOS Symbol Values

This section contains a list of byte-octal values for the HDOS symbols discussed in this document. These values are presented as a double-check, so you can compare them to the values generated when you assemble the common decks. Once again, it is important that you use the common decks and use symbolic values rather than using the octal values directly.

HOSDEF SYMBOL DEFINITIONS

.CHFLG	=	000060A	.EXIT	=	000000A	.PRINT	=	000003A
.CLEAR	=	000055A	.LINK	=	000040A	.READ	=	000004A
.CLOSE	=	000046A	.LOADD	=	000062A	.RENAME	=	8000051A
.CLRCO	=	000007A	.LOADO	=	000010A	.RESET	=	000204A
.CONSL	=	000006A	.MONMS	=	000202A	.SCIN	=	000001A
.CTLC	=	000041A	.MOUNT	=	000200A	.SCOUT	=	000002A
.DECODE	=	000053A	.NAME	=	000054A	.SETTOP	=	000052A
.DELETE	=	000050A	.OPENR	=	000042A	.VERS	=	000011A
.DMNMS	=	000203A	.OPENU	=	000044A	.WRITE	=	000005A
.DMOUN	=	000201A	.OPENW	=	000043A	VERS	=	000026A
.ERROR	=	000057A	.POSIT	=	000047A			

HOSEQU SYMBOL DEFINITIONS

USERFWA	=	42200A
STACK	=	42200A

CO.FLG	=	000001A	CTP.MLO	=	000020A	I.CUSOR	=	000002A

CS.FLG	=	000200A	CTP.TAB	=	000001A	S.DATC	=	040310A
CSL.CHR	=	000001A	CTP.2SB	=	000010A	S.DATE	=	040277A
CSL.ECH	=	000200A	I.CONFL	=	000004A	S.HIMEM	=	040316A
CSL.WRP	=	000002A	I.CONTY	=	000001A	S.OMAX	=	040324A
CTP.BKM	=	000002A	I.CONWI	=	000003A	S.SYSM	=	040320A
CTP.BKS	=	000200A	I.CSLMD	=	000000A	S.USRM	=	040322A
CTP.MLI	=	000040A						

ECDEF SYMBOL DEFINITIONS

EC.CNA	=	000004A
EC.DDA	=	000027A
EC.DIF	=	000017A
EC.DIW	=	000035A
EC.DNI	=	000045A
EC.DNR	=	000046A
EC.DNS	=	000005A
EC.DSC	=	000047A
EC.EOF	=	000001A
EC.EOM	=	000002A
EC.FAO	=	000031A
EC.FAP	=	000026A
EC.FL	=	000030A
EC.FNF	=	000014A
EC.FNO	=	000011A
EC.FNR	=	000034A
EC.FOD	=	000043A
EC.FUC	=	000013A
EC.HIN	=	000000A
EC.ICN	=	000016A
EC.IDN	=	000006A
EC.IFC	=	000020A
EC.IFN	=	000007A
EC.ILC	=	000003A
EC.ILO	=	000040A
EC.ILR	=	000012A
EC.ILV	=	000037A
EC.IOI	=	000052A
EC.IS	=	000032A
EC.NCV	=	000050A
EC.NEM	=	000021A
EC.NOS	=	000051A
EC.NPM	=	000044A
EC.NRD	=	000010A
EC.NVM	=	000042A
EC.OTL	=	000053A
EC.RF	=	000022A
EC.UNA	=	000036A
EC.UND	=	000015A
EC.UUN	=	000033A
EC.VPM	=	000041A
EC.WF	=	000023A
EC.WP	=	000025A
EC.WPV	=	000024A

Part 8

Programming Examples

Menu Prologue for MBASIC

```
MENU Prologue
                                                    HEATH ASM #104.06.00
                                                    15-Oct-80  Page    1

                      00002  ***   MENU Prologue
                      00003  *
                      00004  *     COPYRIGHT 1980, HEATH CO.
                      00005  *
                      00006  *     This Prologue:
                      00007  *             Loads device drivers (if present)
                      00008  *                     LP:
                      00009  *                     LT:
                      00010  *                     LD:
                      00011  *                     AT:
                      00012  *             Runs MBASIC establishing 5 file buffers
                      00013  *             Runs the MBASIC program "MENU.BAS"
                      00014  *
                      00015  *     Note:   The command line may be easily modified to
                      00016  *             accomodate other files, etc.,  by changing
                      00017  *             the line pushed  on the  stack at  "PROB".
                      00018  *
                      00019
042.200               00020        XTEXT   ASCII
042.200               00049        XTEXT   HOSDEF
000.205               00117        XTEXT   HOSEQU
                      00123
030.252               00124  $MOVE  EQU     30252A                 These are routines in the H-17 ROM
031.136               00125  $TYPTX EQU     31136A
                      00126
                      00127
042.200               00128         ORG     USERFWA
                      00129
042.200               00130  START  EQU     *
                      00131
                      00132  *      Load the device drivers
                      00133
042.200  041 027 043  00134  LOAD1  LXI     H,PROAA
042.203  377 062      00135         SCALL   .LOADD                 Load the device driver
042.205  332 226 042  00136         JC      LOAD2                  Can't load, skip message
042.210  315 136 031  00137         CALL    $TYPTX
042.213  114 120 072  00138         DB      'LP: Loaded',ENL
                      00139
042.226  041 033 043  00140  LOAD2  LXI     H,PROAB
042.231  377 062      00141         SCALL   .LOADD
042.233  332 254 042  00142         JC      LOAD3
042.236  315 136 031  00143         CALL    $TYPTX
042.241  114 104 072  00144         DB      'LD: Loaded',ENL
                      00145
042.254  041 037 043  00146  LOAD3  LXI     H,PROAC
042.257  377 062      00147         SCALL   .LOADD
042.261  332 302 042  00148         JC      LOAD4
042.264  315 136 031  00149         CALL    $TYPTX
042.267  114 124 072  00150         DB      'LT: Loaded',ENL
                      00151
042.302  041 043 043  00152  LOAD4  LXI     H,PROAD
042.305  377 062      00153         SCALL   .LOADD
042.307  332 330 042  00154         JC      PSTACK
042.312  315 136 031  00155         CALL    $TYPTX
042.315  101 124 072  00156         DB      'AT: Loaded',ENL
                      00157
                      00158  *      Push the pseudo command line on the user stack for MBASIC to find
```

```
MENU Prologue                                                HEATH ASM #104.06.00
                                                             15-Oct-80  Page    2

                               00159
   042.330  041 000 000 00160  PSTACK  LXI     H,0
   042.333  071              00161          DAD     SP                  HL = current stack value
   000.012                   00162  .       SET     PROBE-PROB+1
   042.334  021 366 377 00163          LXI     D,-.                DE = - ( Number of bytes to push )
   042.337  031              00164          DAD     D
   042.340  371              00165          SPHL                        Reserve the stack space
                               00166
   042.341  001 012 000 00167          LXI     B,PROBE-PROB+1
   042.344  021 047 043 00168          LXI     D,PROB
   042.347  315 252 030 00169          CALL    $MOVE               Move the stuff onto the stack
                               00170
                               00171  *       Link to MBASIC
                               00172
   042.352  041 061 043 00173          LXI     H,PROC
   042.355  377 040          00174          SCALL   .LINK               Try to run MBASIC
                               00175
   042.357  315 136 031 00176          CALL    $TYPTX
   042.362  007 105 122 00177          DB      BELL,'ERROR - Unable to Execute MBASIC',ENL
                               00178
   043.024  257              00179  EXIT    XRA     A
   043.025  377 000          00180          SCALL   .EXIT               Normal EXIT
                               00181
   043.027  114 120 072 00182  PROAA   DB      'LP:',0
   043.033  114 104 072 00183  PROAB   DB      'LD:',0
   043.037  114 124 072 00184  PROAC   DB      'LT:',0
   043.043  101 124 072 00185  PROAD   DB      'AT:',0
                               00186
   043.047  040 115 105 00187  PROB    DB      ' MENU/F:5',0
   043.060                   00188  PROBE   EQU     *-1
                               00189
   043.061  123 131 060 00190  PROC    DB      'SY0:MBASIC.ABS',0
                               00191
   043.100  000              00192          END     START

00192 Statements Assembled
32007 Bytes Free
 No Errors Detected
```

INDEX