OS-9 GMX™ III Manual Addenda
Version 1.2, 09/14/83

This document describes features specific to OS-9 GMX™ III, Version 1.2 (the GIMIX version of OS-9 Level II, for the GMX™ 6809 CPU III board), that are not documented in the standard OS-9 USERS and/or SYSTEM PROGRAMMER'S manuals.

CONTENTS

Differences Between Version 1.1 and 1.2 2

* RBF Changes 3

DEFS Files 3

* Interrupts 3

* Memory Addressing 4

DAT and Memory Attributes 4
Write Protect 4
Unallocated Memory 5
Watchdog Counter 5

Attribute Test Utilities 6

* Device Descriptor Changes 7

* X-ON/X-OFF 7

* G68 (DMA floppy disk) Driver 7

* G68 Device Descriptors 8

Hard Disk Drivers & Descriptors 9

Hard Disk Cluster Sizes 9

Formatting Hard Disks 10

Booting from the Hard Disk 12

Building a customized boot file (OS9Boot) 12

New Error Codes 17

* Switch Configuration Drawings .... Appendix a

WPROT Command b-1

INIZ and INIZP Commands c-1

HDERR Command for OMTI hard disk controllers d-1

* Denotes IMPORTANT variations from versions 0.0 and 0.5 or from the information in the standard OS-9 manuals.

***** Differences Between Versions 1.1 and 1.2 *****

Except for the changes listed in this section, OS-9 GMX III, V1.2 is fully compatible with Version 1.1.

KERNEL: Memory modules may now be loaded into memory space that is not contiguous. This allows better utilization of available memory.

A "Suspend State" has been added to the possible process states. This allows (but does NOT require) drivers to "Suspend" the current process while awaiting an interrupt. In earlier versions, the process was "put to sleep" and a signal sent to "awaken" it. The Suspend State decreases the amount of overhead during interrupt processing. NOTE: Older drivers, using the "sleep" method will still work properly with Version 1.2; however, drivers using the "suspend" method will not work with the older versions.

RBF: A byte-locking style of record locking is now used in place of the sector-locking method used with Version 1.1. Several other minor corrections and speed enhancements have also been made to RBF.

DEVICE DRIVERS: The device drivers (with the exception of G68 floppy disk driver) have been modified to take advantage of the "Suspend State" described above. G68 does not use the "Suspend State" in order to retain the "drive motor time-out" protection described in the G68 Driver section. G68 has been modified to eliminate the "Error #210" problem encountered on earlier versions.

UTILS: In addition to minor changes to some of the utilities, MDIR and PROCS have been modified to work with non-contiguous memory modules. The new versions of these programs MUST be used with Version .1.2. The new versions will NOT work properly with the old Kernel.

PROGRAM DEBUGGING: When a UAM or WPT trap (error #199 or #198) is caused by a program running under the OS-9 debugger, control will now return to the debugger to facilitate location of the problem that caused the trap. A WDC trap (error #197) will still cause the system to return to "shell".

***** RBF Changes *****

Due to changes in RBF affecting the bitmap and directory handling, disks created under GMX™ III, V1.1 or V1.2 (or Level I, V1.2/Level II V1.1/2) CANNOT be written on by the versions of RBF used in Versions 0.x of OS-9 Levels I or II. V1.2 disks can always be READ by these older versions of RBF. V1.2 can read and write disks created by old RBFs; however, once written on by V1.2 RBF, the disks should not be written on by the old RBF. Note: This change affects all current implementations of OS-9 Levels I and II and is not specfic to the GMX™ III version.

OS9defs Files

The "OS9defs" files have been revised. They have been split into several different files to allow selective inclusion in assembler programs, depending on the intended application. A special file (li.equates) may be included in existing programs, that use the old short-form names, to equate the old names to the new, longer names used in the new "OS9defs" files.

***** Interrupts *****

Since OS-9 is an interrupt driven system, all I/O devices (including the time-of-day clock, disk controller(s), and serial/parallel I/O boards for terminals and printers) must have their IRQ interrupt outputs enabled.

Most devices do not actually generate an interrupt on the bus until initialized by OS-9; and a system reset (or turning off the power) clears any interrupt output that may be pending: once the device has been initialized. However, the 6850 ACIAs (used on GIMIX 1-port, 2-port and 8-port serial interfaces) have no direct connection to the system reset line. They can only be reset by the software, or by turning the power off. Because of this, it is possible to reset a running system and leave unserviced interrupts (from the 6850s) on the bus. These "leftover" interrupts can interfere with proper operation of the system when it is rebooted after a reset. Since OS-9 does not automatically initialize all of the serial ports when it is booted (normally only "TERM" is initialized), any "left-over" interrupts must be cleared by some other means. Turning the system off before booting will insure that the interrupts are clear, but this is not always practical. Two special utilities "INIZ" and "INIZP" can be used in the "startup file" to clear any unserviced interrupts when the system is rebooted. See appendix c for information on INIZ and INIZP.

***** Memory Addressing *****

OS-9 GMX™ III and the GMX™ CPU III allow a full 64K of RAM to be installed on each of the first 15 memory banks ($0-$E). The 16th bank ($F) can have up to 56K of RAM installed; the upper 8K are reserved for I/O and the operating system PROMs. RAM should be installed in contiguous 64K banks beginning at address $0000 on bank $0. All I/O devices including the motherboard and disk controllers must have extended address decoding enabled and set to decode bank address $F (see appendix a).

Since the GMX™ CPU III automatically switches to the system state when servicing interrupts, the interrupt vectors need not appear in each of the 16 memory banks as in previous systems. This means that up to 64K of RAM may be allocated to a user or task with no system overhead; the ENTIRE 64K is available to the user/task.

Dynamic Address Translation and Memory Attributes

The GMX™ CPU III includes an expanded Dynamic Address Translator (DAT) which, in addition to providing translation in 2K segments for more efficient memory usage, allows the assignment of "memory attributes" on a block by block (2K block) basis. These memory attributes are: Write Protect, Single Step, and Unallocated Memory. They may be selected separately or in combinations for each individual 2K memory segment in the address space. The purpose of these attributes is to provide protection for the system and to provide enhanced debugging capabilities. Currently only the Write Protect and Unallocated Memory attributes are implemented in OS-9 GMX™ III.

Write Protect Attribute

The WPT attribute is used to protect areas of memory, such as those containing sharable OS-9 modules or user programs, from unauthorized and/or unintentional modification. This greatly increases the security of the system by preventing, for example, one user from corrupting the copy of BASIC09 that is being shared by with users. OS-9 determines the intended status of a program (protected or unprotected) by testing a bit in the module header when the module is loaded into RAM. Once loaded, a protected module cannot be written to or modified, except by replacing it with another module.

Normally, programs supplied by GIMIX and other software vendors are write-protected. Since the system prevents even intentional modifications, the Debugger can not be used to change or patch write-protected modules. Source code for most device descriptors and for certain other modules is provided on the system disk. The source can be edited and reassembled to modify these modules. Also included on the system disk is the utility program "WPROT" (see appendix b) that changes the write-protect status of a module. WPROT can be use to temporarily write-enable a module so it can be modified with the Debugger.

To facilitate modification and debugging, modules can be run without write protection. However, to maintain maximum system security, they should be write-protected after modification and testing are completed. Note: To protect the system from crashes (especially important if there are other users on the system) and help detect certain types of bugs, initial testing should be done with write protection enabled.

The write protect status of a module is determined by bit 6 (previously undefined) in the module header's attributes/revision byte at module offset 7. (Bit 7 of this byte is the reentrant/sharable bit, see the System Programmer's Manual.) If this bit is cleared (0) the module will be write-protected when loaded by OS-9, if set (1) the module will be write-enabled.

If the GMX™ CPU III detects an attempt to write to write-protected location in memory, a trap occurs and, as a result of the trap, OS-9 will terminate the offending task (the task that initiated the write operation) and issue an error message (Error #198).

Unallocated Memory Attribute

The UAM attribute is used to prevent a user/task from making memory accesses (read or write) outside the memory area assigned to it by the operating system. Like the WP attribute, this increases the security of the system by preventing unauthorized access to selected areas of memory. The UAM attribute is automatically controlled by OS-9. When a task is initiated, all memory requested for the task is "allocated" to that task, all memory outside the requested area is "unallocated" and therefore inaccessible to that task.

If the GMX™ CPU III detects an attempt to access (read or write) unallocated memory, a trap occurs and, as a result of the trap, OS-9 will close down the offending task (the task that initiated the access) and issue an error message (Error #199).

Watchdog Counter

An additional function on the GMX™ CPU III board, the Watchdog Counter, is also implemented in OS-9 GMX™ III. The WDC is controlled by the operating system and, when enabled, limits the length of time that interrupts may remain masked and therefore unserviced. The WDC is enabled by OS-9 when the system is switched from the system state (operating system) to the user state (user program running). It begins counting CPU clock cycles when an interrupt occurs and, if the interrupt is not serviced by the CPU in a specified number of cycles (hardware selected: 128 standard,32 optional) a trap occurs. When a WDC trap occurs, OS-9 will close down the offending task and issue an error message (Error #197).

The WDC trap provides two types of protection. It protects against a user program that masks IRQ interrupts; necessary to the system for task switching and I/O processing. It also protects against the execution of certain illegal opcodes which can lock the 6809 in a state in which it will not execute further instructions or respond to any interrupts. To recover from this processor state, the WDC generates a special reset (only the 6809 is reset). The reset is processed through a special trap-vector instead of the normal reset vector, allowing OS-9 to resume processing of all but the task which caused the trap.

Test Utilities

Three utility programs are provided with the OS-9 GMX™ III system to demonstrate and test the function of the WPT, UAM, and WDC traps. The utilities (test199uam, test198wpt, and test197wdc) are provided in both source and object form to help the user understand the function of the traps. Execution of one of the utilities will cause OS-9 to close down the task and issue the appropriate error message.

TEST199UAM

Test199uam tests the function of the Unallocated Memory trap by reading from a memory location beyond the end of its normally allocated data area. Note: if test199uam is run with a memory-size modifier in the command line (i.e. test199uam #6K) a trap will not occur and the program will loop until "KILL"ed.

TEST198WPT

Test198wpt tests the function of the Write Protect trap by writing to itself. Since bit 6 of the program's attributes field is clear (0), the module is write-protected and a trap will occur.

TEST197WDC

Test197wdc tests the function of the Watchdog Counter by setting the IRQ mask and entering a wait loop. Once the watchdog count expires (normally 128 CPU cycles) the trap will occur. Note: in order for this test to function, an IRQ must occur after the program is executed; to start the watchdog count. If the system clock has been started (by running "setime") the clock interrupts will cause the trap to occur. If the system clock is not generating interrupts, an interrupt can be caused by hitting a key on the terminal.

In addition to providing a considerable degree of protection for the system, the attributes and watchdog traps also assist in software debugging. In many cases they will detect common problems (caused by uninitialized pointers, incorrect addressing modes, etc.) that might otherwise go unnoticed or be difficult to identify.

Device Descriptors

The device descriptors provided with OS-9 GMX™ III follow the conventions established in earlier versions of OS-9 from GIMIX. The header on the catalog printout, included with the system disk, lists the configuration of the disk, terminal, and printer descriptors provided. Since the descriptors are write-protected by the GMX™ III system when they are in memory, the only way to modify them for user-specific requirements is to modify the source code and reassemble them. The GMX™ III system does not permit "hot patching" with debug as in previous versions. The source code to all of the standard device descriptors is provided in the "SOURCE" directory on the system disk(s).

***** Device Descriptor Changes *****

Due to changes in the Device Descriptors (Device Controller Address field), device descriptors from earlier versions of OS-9 (Level II V0.5 or earlier and OS-9 level I prior to version 1.2) CAN NOT BE USED, without modification, with OS-9 GMX™ III. In order to use existing descriptors, the extended address byte (at module offset $E) must be set to $0F. Without this modification, the system will appear to function, but performance will be seriously degraded!

***** X-ON/X-OFF *****

In addition to the address change noted above, the descriptors for ACIA devices (TERM, T1, T2, ... , and P1) have additional bytes added to control the X-ON/X-OFF functions (module offset $2A = X-ON, offset $2B = X-OFF). These bytes can be set to the desired ASCII codes for these functions if they are required. Setting these bytes to $00 disables X-ON/X-OFF (terminal descriptors shipped with GMX™ III have these bytes set to $00). X-ON/X-OFF can also be enabled using TMODE. When X-ON is set to $11 (control-Q), the QUIT character, normally control-Q, (module offset $23) must be set to some other control character. The recommended substitute character is $05 (control-E). Note: When terminals that generate X-ON/X-OFF sequences are used, X-ON/X-OFF must be enabled in the system.

G68 DMA Disk Controller Drivers

OS-9 GMX™ III uses an enhanced version of the G68 device driver. This G68 supports 3 ms. stepping rates for 5.25" disk drives, allowing faster access times with drives that are capable of stepping at this rate. (All 5.25", 80 track (96TPI) drives currently supplied by GIMIX are capable of stepping at 3 ms.) This option is controlled by a separate bit in the floppy disk device descriptors as described below.

The new G68 prevents the drive motors from timing-out if a drive with no disk (and no "Drive Ready" line) is accessed. On earlier versions of G68, if the drive motors timed-out the system would hang and usually require re-booting. This feature will only function if the system interrupt clock has been started by using the "SETIME" command. We recommend that "SETIME" be included in the "STARTUP" file ("setime 00" will start the clock running, without prompting for new values). Note: To prevent the system from hanging if a non-existant drive is accidentally accessed, a system disk should be created that does not have device descriptors for the non-existant drive(s).

G68 Device Descriptors (D0,D1, etc.)

The device descriptors for the G68 driver follow the definitions given in the OS-9 manuals with the following exception.

MODULE OFFSET	NAME	DESCRIPTION
$14	IT.STP	Bits 0 (LSB) and 1 determine the basic stepping rate as shown in the 179x table in the manual. Bit 7 (MSB) controls "fast stepping" for 5.25" floppy drives

The fast stepping bit (Bit 7) when set (1) in a descriptor for 5.25" drives causes the stepping rate (determined by bits 0 and 1) to be doubled. The 5.25" drive will be stepped at the rate shown in the 179X, 8" drive column in the table. When Bit 7 is clear (0), the drives will be stepped at the normal 5" rate shown in the 179X, 5" column. The fast stepping bit has no effect on 8" drives and should be clear (0) in 8" descriptors.

Caution: Be sure the drives are capable of stepping at the selected stepping rate. In general the only 5.25" drives capable of stepping at 3 ms. (IT.STP = $83) are the newer 80 track (96TPI) drives. Various combinations of the step rate and fast stepping bits can be used to match the stepping rate to the drive(s) used. See the disk drive manufacturer's literature to determine the stepping capabilities of the drive(s). Attempting to step a drive faster than its specified maximum rate will cause excessive disk errors and/or prevent the drive from working at all.

Hard Disk Drivers and Descriptors

Like floppy disk drivers, the hard disk drivers consist of two parts: a device driver that interfaces the hardware to the Random Block File manager (RBF) and the device descriptor(s) that describe the characteristics of the hard disk drives. Hard disk systems may be supplied with either one of two brands of hard disk controller. Depending on the controller used, the device driver will be called either "XBC" (for XEBEC controllers), or "OMTI" (for controllers manufactured by OMTI). The drivers are customized to the particular controller and are NOT interchangable. The device descriptors for the hard disks are named "H0", for the first drive, and "H1" for the second. While the descriptors for both controllers have the same names, they are also NOT interchangable. The driver and descriptors are factory configured for a specific controller and drive(s) and are not normally modified by the user. The configuration printout supplied with the master system disk lists controller and drives that the disk is configured for.

NOTE: Version 1.2 of OS-9 GMX™III uses a different driver/descriptor format than V1.1. Drivers/Descriptors from V1.1 should not be mixed with those of V1.2.

Hard Disk Cluster Sizes

OS-9 keeps track of disk space allocation with a "bit-map" which is kept on each disk. The bit-map occupies one or more sectors, depending on the total capacity of the disk and its cluster size. Each bit in the bit-map represents a group of 256-byte sectors on the disk, called a cluster. A cluster may represent a single physical sector (cluster size 1), or a group sectors (cluster size n). The number of sectors in a cluster is determined when the disk is formatted, and is always an integral power of 2 (1, 2, 4, 8, ...).

The minimum number of sectors that OS-9 can allocate to a file is one (1) cluster. Normally, to maximize disk usage, a cluster size of 1 is used. However, due to internal requirements of OS-9, this limits the maximum size of an individual file to approximately 24 Megabytes. This limitation does not affect floppy disks or the low capacity hard disks, with total capacities less than 24 Mbytes. However, with larger capacity hard disks, it may be desirable to have individual files larger than 24 Mbytes. In order to permit files larger than 24 Mbytes, the disk must be formatted with a cluster size greater than 1.

The "Format" utility included with OS-9 GMX III prior to Dec. 1, 1984 uses a default value of one (1) sector per cluster for both floppy and hard disks. Copies of "Format" shipped after 12/01/84 use a default of one (1) for floppy disks, and eight (8) for hard disks. Both versions of "Format" allow the user to select a different cluster size by specifying the desired size on the command line when format is invoked. (This option is not documented in the current OS-9 manuals; however, it is documented in the program itself, and can be seen by invoking format without a drive specification.)

The cluster size is specified as a decimal number, which must be an integral power of 2 (1, 2, 4, 8, ...), delimited by slashes (e.g. /8/). If an improper value is used, format will revert to the default value.

For example, the command:

```
format /h0 /8/
```

would format the device "/h0" with 8 sectors per cluster.

The "free" utility can be use to determine the number of sectors per cluster on a previously formatted disk.

NOTE: Once a disk has been formatted, the cluster size can only be changed by completely reformatting the disk. If files larger than 24 Megabytes are anticipated, the disk MUST be formatted for a cluster size greater than 1 before the disk is used. If it is necessary to change the cluster size of a disk containing files, the data must be copied to another disk(s) and the original disk must be reformatted.

There are several factors to consider when determining the appropriate cluster size for a particular application. A cluster size of one (1) generally makes the most efficient use of available disk space (at most only a fraction of one sector is wasted per file), but limits the maximum file size to approximately 24 Megabytes, and tends to increase fragmentation of the disk. (Fragmentation is when individual files are stored in groups of non-contiguous sectors (clusters) scattered on the disk, rather than a single contiguous group.) Fragmentation increases the disk access time by increasing the number of disk seeks required to access the entire file. Larger cluster sizes increase the maximum file size, and tend to decrease fragmentation; however, large cluster sizes increase wasted disk space by as much as (clustersize - 1) sectors per file.

Formatting Hard Disks

The same "FORMAT" program is used to format both floppy and hard disks. Format determines the type of disk being formatted from the device descriptor. There are several differences in the options available when formatting hard disks, as described below. Hard disks are normally formatted only once; when the drive is first used. Once the hard disk is formatted it does not require reformatting unless the file structure is damaged by a hardware or software fault, or if it becomes desirable to erase ALL of the files on the disk at once.

CAUTION !!

A physical format (see below) destroys all data recorded previously on the disk. A logical format, while it does not completely destroy previously recorded data, makes the data very difficult if not impossible to recover!

Systems are normally shipped with the hard disk(s) already formatted, as part of the testing performed on the system. The hard disk may contain copies of the files provided on the system (floppy) disk, and in some cases, additional "special" files, not included on the floppy disks. Before formatting the hard disk(s) use the directory (DIR) command to check the contents of the hard disk(s), and copy any "special" files from the hard disk to a floppy to preserve them. If the disk(s) are already formatted they can be used as-is or reformatted as desired. Using the logical-only format option (see below) will save time and accomplish the same results as a complete physical and logical reformatting. Note: the logical-only format option can be used to reformat disks that were previously formatted for a different operating system (such as FLEX). If the hard disk(s) do not appear to be formatted (as evidenced by read or seek errors when attempting to perform the "dir") they should be formated using both the physical and logical format as described below.

When FORMAT is called with a hard disk (/H0 or /H1) as the device to be formatted, it first prints a list of the parameters that will be used to format the hard disk. Unlike the floppy disk format, these parameters are fixed by the requirements of the drives and cannot be changed from the FORMAT utility. Entering either "N" (NO) or "Q" (QUIT) at this time will abort the format program. If "Y" is entered, FORMAT will ask :

```
Both Physical and Logical Format ?
```

Answering "Y" (YES) will cause FORMAT to perform both types of format. If "N" (NO) is entered only the logical format will be performed.

PHYSICAL FORMAT

The physical format is normally only required when formatting a new hard disk that has not been formatted previously. The physical format records the information required by the controller to divide the disk into sectors and to locate a particular track and sector on the disk. The operating system does not modify this information once it is written.

LOGICAL FORMAT

The logical format records the information that OS-9 requires to store and keep track of the data files that will be written to and read from the disk. Some of this information is modified by the operating system as files are written to and deleted from the disk. Performing a logical format has the same effect as deleting ALL files on the disk. The logical format takes considerably less time than the physical format.

After the type of format desired is entered, FORMAT will prompt for a disk name, which must be entered following the same syntax as the name for a floppy disk. The next prompt is:

```
Physical Verify Desired ?
```

Answering "Y" will cause a physical verify to be performed. Each sector is read and the information written there is verified. As the verify is performed, FORMAT prints the number of each completed track on the standard output device. Note: Before performing a physical verify, use TMODE to turn the pause function off or FORMAT will stop at the end of each page of track numbers. Answering "N" causes the physical verify to be skipped.

Booting From The Hard Disk

It is possible to boot OS-9 GMX III from a hard disk in one of two ways, depending the type of OS-9 used. The Support ROM version of OS-9 (GMX IIIs) can be booted directly from the hard disk, once an "OS9Boot" file and the other necessary files have been installed on the hard disk. (See the Support ROM documentation for more information.) The original OS-9 GMX III (without Suppport ROM) requires that OS9Boot reside on a floppy disk. However, OS9Boot can automatically transfer to the hard disk once it has loaded from the floppy. (This method can also be used with the Support ROM version.) The initial directories will be /HO and /HO/CMDS, the system will attempt to execute the "STARTUP" file from /HO, and "Login" will expect to find the "PASSWORD" and "MOTD" files in the directory /HO/SYS.

In either case, a special OS9Boot file is required. OS9Boot must include the hard disk version of the system initialization module "init" in place of the normal "init". A copy of this module is included on the OS-9 system disk in a file called "Init.hd".

Building a customized boot file (OS9Boot)

Before attempting to create a new boot file, use one of the procedures outlined in the OS-9 manuals to create backups of the original system disk and store the original in a safe place. If the system only has one floppy disk drive, backups can be made by using the single drive option of either the "backup" or "copy" commands, or by first copying the entire floppy disk to the hard disk and then copying from the hard disk to another floppy. Copies made using the "copy" command can not be used to boot the system unless the "OS9gen" utility is first used to install a bootfile on the disk. Copies made using "backup" can be used to boot the system, without the need for using "OS9gen". Note: In order to use "backup", the disk being created must have the same format as the original disk (number of tracks, density, sides, etc.). The format of the supplied system disk is listed on the printout provided with the disk.

When OS-9 is bootstrapped from disk (floppy or hard), it loads a file called OS9Boot into memory from the boot disk. OS9Boot is created by the "OS9Gen" command, and contains all of the modules normally required to run OS-9. Depending on the system, this file includes all or part of the OS-9 kernel, as well as the file managers, device drivers, and device descriptors. The boot file (OS9Boot) supplied on the original system disk contains everything necessary to run the system in a minimum configuration (see the printout included with the disk). However, it is often necessary for the user to create a customized boot file to suit a particular application. A new boot file is required to take advantage of certain features, such as the ability to boot from a hard disk, or to enable the use of a hardware CRC board. A customized boot file allows the user to install device descriptors for additional devices (terminals, printers, etc.) and possibly remove some unnecessary modules in order to conserve memory. For example, if parallel I/O is not required, the parallel driver "PIA" and parallel descriptor "P" can be omitted. If the system will use only intelligent I/O boards, with no standard ACIA or PIA devices, the drivers and descriptors for these devices, as well as the Sequential Character File manager (SCF) can be omitted from the boot file. The intelligent I/O devices have their own file manager (IOPman) and do not need SCF. Device descriptors for non-existant disk drives should also be omitted.

In order to facilitate the creation of customized boot files, several special files are included on the supplied system disk. Other necessary files are obtained by using the OS-9 "Save" command to create copies of them from memory. The source code for various device descriptor types is also included on the system disk. These source files can be edited and assembled by the user to generate additional device descriptors or to make modifications to the standard descriptors.

Note: The system module "SysGo", on the supplied system disk, has its attributes set to non-reentrant. Since it is non-reentrant, SysGo can not be copied from memory by using "save". If a separate copy of SysGo is needed, it must be assembled from the source code included on the system disk.

Included on the system disk are the files "Init.hd", OS9Boot.core, and, with Support ROM versions only, "OS9p1.hcrc_6" and "OS9p1.hcrc_7". Init.hd is a version of the "Init" module configured for booting from a hard disk (see the preceding section). OS9Boot.core includes most of the modules normally included in OS9Boot, with the exception of the device descriptors (the "Pipe" descriptor is included), the hard disk driver (XBC or OMTI), and the initialization module "Init". (Use the Ident command, -s option, to list the modules included in OS9Boot.core.) OS9p1.hcrc_6/7 are special versions of the kernel module "OS9p1", required when a GIMIX CRC GENERATOR BOARD is installed.

The minimum requirements for a usable boot file are listed below. Additional modules, such as device descriptors can be added as necessary; however, the listed modules must be included or the system will not boot.

Note: In systems using the OS-9 GMX III Support ROM, OS9p1 must be included in the boot file, in systems without the Support ROM, OS9p1 is in ROM and should not be included in the boot file.

***** Important *****

In OS-9 GMX III systems, as with any OS-9 Level II system, the "shell" should NOT be incorporated in the bootfile, as is normally the case with OS-9 Level I.

Minimum boot file requirements

Support ROM systems

OS9p1 †
OS9Boot.core §
INIT[.hd]
DO
TERM
<XBC or OMTI> *
HO *

Standard systems

OS9Boot.core §
INIT[.hd]
DO
TERM
<XBC or OMTI> *
HO *

† This can be the standard OS9p1 "saved" from memory or one of the hardware CRC board versions. OS9p1 must be the first module in the boot file.

§ OS9Boot.core may be replaced by the equivalent individual modules.

* Hard disk systems only, driver and descriptor must match the controller used.

The following examples outline the procedure for a system that includes one floppy disk (DO), one hard disk (HO), two terminals (TERM and T1), and a parallel printer (P). Where the hard disk driver is specified use either XBC or OMTI as appropriate to the system. The files OS9Boot.core and Init.hd are assumed to exist in the current working directory.

Note: If the destination for the OS9Boot file is not a freshly formatted disk, and already contains files, OS9Gen may fail because of disk fragmentation and the requirement that the OS9Boot file reside on contiguous sectors. An OS9Boot can usually be created on a disk (floppy or hard) that already contains files; however, it may be necessary to "use up" existing small groups of non-contiguous sectors by creating dummy files, until a large enough block of contiguous sectors is available. If OS9gen aborts because of insufficient contiguous sectors, a file called tempboot will be left on the disk. This temporary file can be renamed before using OS9gen again, causing OS9gen to use a different location on the disk the next time it is called. This process may need to be repeated several times, renaming "tempboot" to a different name each time, before a successful OS9gen is run. Once OS9gen is successful, the renamed tempboot file(s) can be deleted to restore the disk space. Note: OS9gen will automatically delete an existing OS9Boot file on the destination disk; however, it will not delete the "tempboot" file if one exists. The tempboot file must be renamed (or deleted) before OS9gen can be run again, or a "File Already Exists" error (#218) will be generated.

EXAMPLE #1: system without Support ROM

```
OS9:save  DO           Create a file containing the descriptor "DO"

OS9:save  HO           Create a file containing the descriptor "HO"

OS9:save  Term         Create a file containing the descriptor "TERM"

OS9:save <XBC or OMTI> Create a file containing the appropriate
                        disk driver.

OS9:save  T1           Create a file containing the descriptor "T1"

OS9:save  P            Create a file containing the descriptor "P"

OS9:build bootlist     Create a text file containing a list of the
                       files to be included in the new OS9Boot file.

    ? OS9Boot.core
    ? Init.hd
    ? Term             Additional descriptors or modules can be
    ? <XBC or OMTI>    included in the list as necessary.
    ? HO
    ? DO
    ? T1
    ? P
    ? [return]

OS9:OS9gen /DO  <bootlist   create an OS9Boot on the floppy

OS9:
```

EXAMPLE #2: system with Support ROM

```
OS9:save  OS9p1        Create a file containing the kernel   "OS9p1"
                        (omit this step and use one of the hard-
                         ware CRC versions (OS9p1.hcrc_x) if the
                         system includes a CRC generator board.)

OS9:save  DO           Create a file containing the descriptor "DO"

OS9:save  HO           Create a file containing the descriptor "HO"

OS9:save  Term         Create a file containing the descriptor "TERM"

OS9:save <XBC or OMTI> Create a file containing the appropriate
                        disk driver.

OS9:save  T1           Create a file containing the descriptor "T1"

OS9:save  P            Create a file containing the descriptor "P"
```

```
OS9:build bootlist    Create a text file containing a list of the
                      files to be included in the new OS9Boot file.

    ? OS9p1[.hcrc_x]  Must be the first module in the list
    ? OS9Boot.core
    ? Init.hd
    ? Term            Additional descriptors or modules can be
    ? <XBC or OMTI>   included in the list as necessary.
    ? HO
    ? DO
    ? T1
    ? P
    ? [return]

OS9:OS9gen /HO <bootlist    create an OS9Boot on the hard disk

        -or-

OS9:OS9gen /DO  <bootlist   create an OS9Boot on the floppy


OS9:
```

In systems that have intelligent serial I/O boards, the device descriptors for terminals other than "Term" are normally called IT1, IT2, etc.

Customized boot files that boot to the floppy disk, rather than the hard disk, can be created by "SAVE"ing a copy of the standard Init module from memory and using it in place of Init.hd in the above examples.

The disk created by this procedure can now be used to boot the system. The root directory of the hard disk (HO) must contain a CMDS directory, which becomes the execution directory, and the "STARTUP" file if one is to be used. The system will also search "HO" for the "SYS" directory containing the "PASSWORD" and "MOTD" files.

Note: Much of the available software for OS-9, especially assemblers, compilers, etc., expect to find common files such as "DEFS" or "SYS" files in a particular directory on the system boot device. Some programs of this type assume that the system boot device is always "/DO", and will still look for the files on "/DO", even if the system is booted from a hard disk. Programs that check the "Init" module to determine the name of the system boot device, rather than assume it is "/DO", will locate these files on the hard disk when it is used as the boot device.

OS-9 ERROR CODES

The following error codes should be added to the error code list in the OS-9 manuals. They are generated by hardware "traps" on the GMX III 6809 CPU board.

HEX	DEC	
$C5	197	WATCHDOG COUNTER TRAP - The CPU was unable to respond to an interrupt before the Watchdog count expired. The interrupts were masked for too long in a user task or the CPU executed an illegal instruction and was unable to respond.
$C6	198	WRITE PROTECT TRAP - An attempt has been made to write to a module in memory that has the write-protect bit in its header cleared (0), enabling write-protection. (See page 5 for more information.)
$C7	199	UNALLOCATED MEMORY TRAP - An attempt was made to access (read or write) memory not allocated to the program.

We would appreciate information from users on any other problems encountered while using this version of OS-9 GMX™ III. To be most useful, the information should be submitted in writing, and should include a complete description of the problem and the conditions under which it occurs, including the Edition number(s) of any programs involved (use the IDENT command). A brief description of the hardware configuration should also be included so that we can attempt to duplicate the problem if necessary. Information regarding problems with the software should be addressed to:

GIMIX Inc.
1337 W 37th Place
Chicago Il 60609

Attn: Mike Magnus

SWITCH CONFIGURATION DRAWINGS FOR OS-9 GMX II & III

The following drawings show the standard DIP-switch configurations for the GIMIX 64K RAM board(s), #68 DMA Disk Controller, Hard Disk Interface (SASI), and Mother Board; when used with OS-9 GMX II & III.

The disk controller(s) must be set to both drive and decode extended addressing (both "ENA" switches ON). The motherboard must be set to decode extended addressing. The boards are addressed so they appear only on bank $F (A16, 17, 18, 19 = ON), at the appropriate base address.

The memory boards are addressed for either 64K banks (OS-9 GMX III systems or OS-9 GMX II systems with modified #05 CPU boards) or 56K banks (unmodified OS-9 GMX II systems). The configuration of switch S3 will normally be the same on all boards in the system, with section-7 ON for 64K banks or OFF for 56K. Both extended address enable switches ("XON"-sections 1 and 6) must be "ON" on all boards. The remaining eight sections of S2 determine the bank address of the board. The boards are divided into two halves, with sections 2,3,4, and 5 used to set the bank address for one half and sections 7,8,9, and 10 the other. In this application both halves are set to the same bank address. The switches are set in a binary pattern with section-2(7) being the least significant bit and section-5(10) the most significant. See the drawing for examples. The boards should be addressed on consecutive banks with the first board on bank $0, the second on bank $1, etc.

***** CAUTION *****

In addition to the boards shown in the drawings, any other memory-mapped boards installed on the 50 pin bus (GIMIX 8-port Serial Interfaces, Intelligent Parallel Interfaces, PROM/ROM boards, etc.) must be capable of extended addressing. Normally, I/O-type boards must be addressed on bank $F, with a base address in the $E000-$EFFF range. Memory-type boards (PROM/ROM) can be addressed as appropriate to the application, as long as extended addressing is enabled. Note: Standard versions of OS-9 only search the lower 56K of bank $F for PROM/ROM memory modules. In order to be located by OS-9, boards containing OS-9 modules in PROM/ROM must be addressed on bank $F.

***** NOTE *****

The switch configuration shown for the #68 DMA board assumes that the system will be booted from a 5.25" drive (D0). If D0 is an 8" drive, S2 section-9 must be OFF.

64K RAM BOARD

SWITCH CONFIGURATION

FOR OS-9 GMX II & OS-9 GMX III

BANK ADDRESS (S2)

	$Ø	$1	$2	$3	$F	
1	ON ←				→ ON	XON
2	OFF	ON	OFF	ON	ON	A16
3	OFF	OFF	ON	ON	ON	A17
4	OFF	OFF	OFF	OFF	ON	A18
5	OFF	OFF	OFF	OFF	ON	A19
6	ON ←				→ ON	XON
7	OFF	ON	OFF	ON	ON	A16
8	OFF	OFF	ON	ON	ON	A17
9	OFF	OFF	OFF	OFF	ON	A18
1Ø	OFF	OFF	OFF	OFF	ON	A19

BOARDS SHOULD BE ADDRESSED ON SUCCESSIVE BANKS - (1ST BOARD, BANK Ø; 2ND BOARD, BANK 1; 3RD BOARD, BANK 2; ETC.)

S3

1	2	3	4	5	6	7	8	9	1Ø
ON	ON	ON	ON	ON	OFF	*	ON	ON	ON

$ØØØØ-$DFFF ($FFFF)

* ON=64K OFF=56K

USE 64K FOR ALL BOARDS (EXCEPT BANK $F) IN GMX III AND MODIFIED GMX II SYSTEMS.

USE 56K FOR UNMODIFIED GMX II SYSTEMS.

DMA FLOPPY DISK CONTROLLER
OS-9 GMX II & GMX III CONFIGURATION

ADDRESS=$FE3BØ BOOT DRIVE (0)=5 1/4"

S1

EXTENDED ADDRESS					REGISTER BASE ADDRESS				
ENA	A16	A17	A18	A19	A3	A4	A5	A6	A7
ON	ON	ON	ON	ON	OFF	ON	ON	OFF	ON
1	2	3	4	5	6	7	8	9	10

ON=1

OFF=Ø

S2

REGISTER BASE ADDRESS									
A8	A9	A10	A11	A12	A13	A14	A15	5"	ENA
ON	ON	OFF	OFF	OFF	ON	ON	ON	ON	ON
1	2	3	4	5	6	7	8	9	10

S1 1 — EXT. ADDRESS DECODE

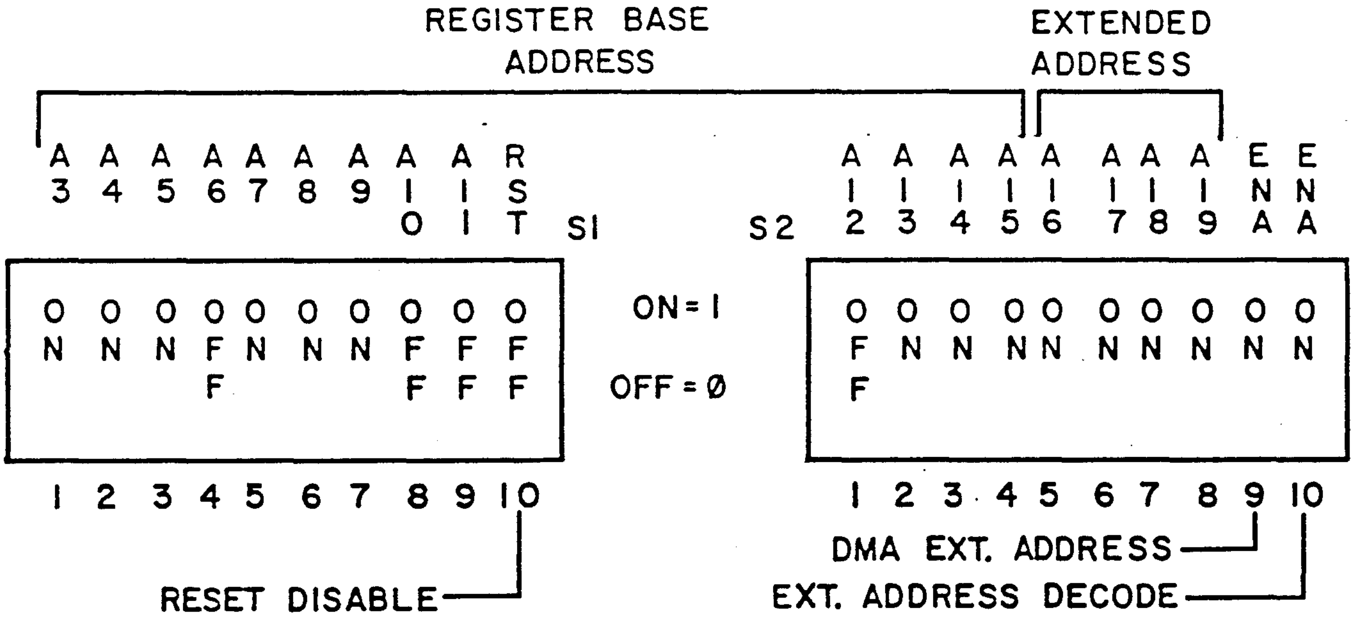
S2 9 — 5 1/4" OR 8" BOOT SELECT

S2 10 — DMA EXT. ADDRESS

HARD DISK INTERFACE
SWITCH CONFIGURATION
OS-9 GMX II & GMX III

FOR XEBEC CONTROLLERS

ADDRESS=$FE3B8

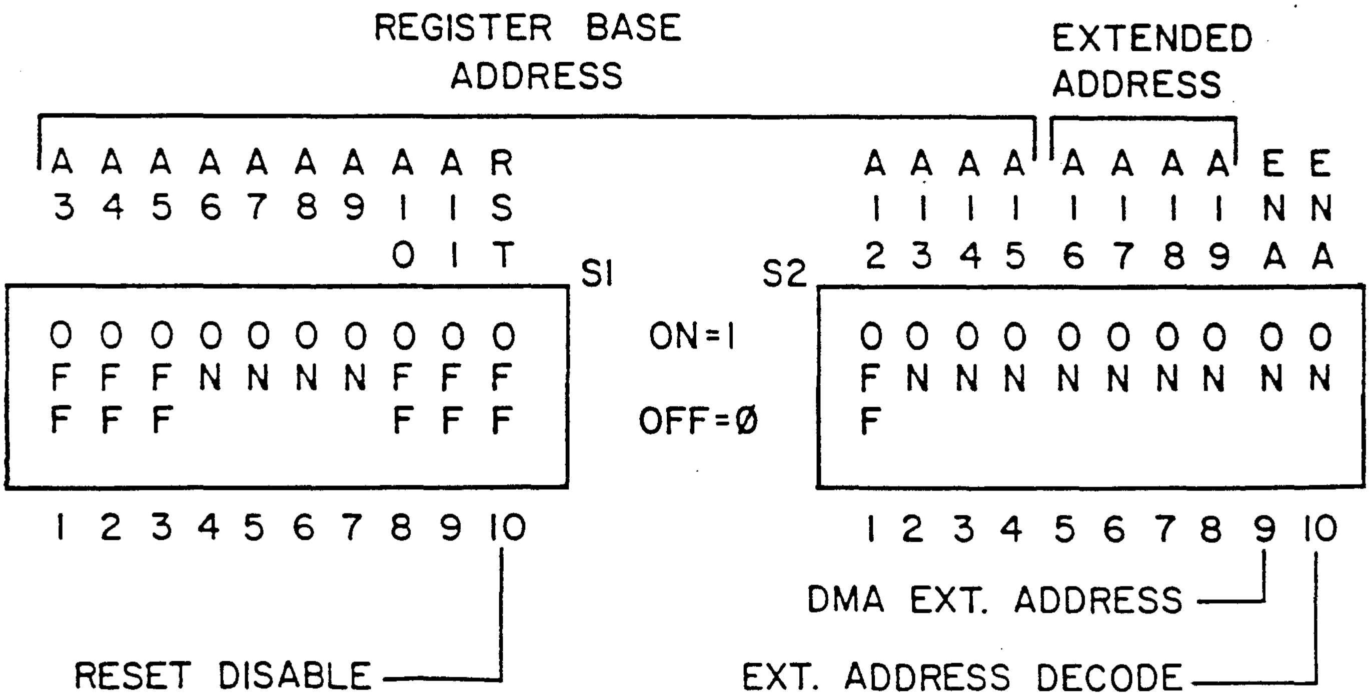


S1	A3	A4	A5	A6	A7	A8	A9	A10	A11	RST
	ON	ON	ON	OFF	ON	ON	ON	OFF	OFF	OFF
	1	2	3	4	5	6	7	8	9	10

S2	A12	A13	A14	A15	A16	A17	A18	A19	ENA	ENA
	OFF	ON	ON	ON	ON	ON	ON	ON	ON	ON
	1	2	3	4	5	6	7	8	9	10

FOR OMTI CONTROLLERS

ADDRESS=$FE3CØ



S1	A3	A4	A5	A6	A7	A8	A9	A10	A11	RST
	OFF	OFF	OFF	ON	ON	ON	ON	OFF	OFF	OFF
	1	2	3	4	5	6	7	8	9	10

S2	A12	A13	A14	A15	A16	A17	A18	A19	ENA	ENA
	OFF	ON	ON	ON	ON	ON	ON	ON	ON	ON
	1	2	3	4	5	6	7	8	9	10

MOTHER BOARD SWITCH CONFIGURATION

OS-9 GMX II & GMX III

ADDRESS = $FEØØØ

OFF(O PEN) Ø	S2	ON (CLOSED) 1	
	OFF	4	I-O SLOT
	OFF	8	
	ON	16	
	OFF	A5	
	OFF	A6	
	OFF	A7	LOW ORDER
	OFF	A8	STARTING
	OFF	A9	ADDRESS BITS
	OFF	A10	
	OFF	A11	

OFF(OPEN) Ø	S1	ON(CLOSED) 1	
	OFF	DIS	I-O ENABLE/DISABLE
	OFF	A12	
	ON	A13	HIGH ORDER STARTING
	ON	A14	ADDRESS BITS
	ON	A15	
	ON	EXON	EXTENDED ADDRESS ENABLE/DISABLE
	ON	A16	
	ON	A17	EXTENDED ADDRESS
	ON	A18	
	ON	A19	

WPROT

Change module's write-protect status

SYNTAX: WPROT

FUNCTION: WPROT is used to invert the hardware write-protect bit (bit 6) in a module's Attributes/Revision byte (module offset 7). By inverting this bit, a write-protected module can be write-enabled or a write-enabled module can be write-protected. Write-enabling a module allows it to be "patched" in memory using DEBUG. Once the module has been patched, WPROT is used to write-protect it again.

WPROT reads a file containing a module or group of modules from standard input and writes the module(s), with the write-protect bit inverted, to standard output. A message indicating the write-protect status of each output module, and any error messages, are written to the standard error path.

Note: WPROT does not verify or update the module's header parity or CRC. The "VERIFY" utility must be used to update these values before a module processed by WPROT can be loaded or used.

WARNING: Modules should only be write-enabled for debugging or patching purposes. Once the module has been debugged or patched, it should be write-protected to preserve system security. Disabling write-protection bypasses the protection provided by the memory management hardware on the GMX 6809 CPU III.

For more information see the "OS-9 GMX III MANUAL ADDENDA" and page 4-5 of the OS-9 "SYSTEM PROGRAMMERS MANUAL" (revision H).

In the following examples, "file_name" is any valid OS-9 pathlist ending with the name of a file containing the module(s) to be processed. "new_file_name" is a pathlist ending in a name for the output file that will be created.

EXAMPLES:

```
 OS9: WPROT <file_name  >new_file_name

      Output module is write-enabled.

 OS9:


 OS9: WPROT <file_name ! VERIFY U >new_file_name

      Output module is write-protected.

      Output module is write-protected.

      Output module is write-enabled.

 OS9:
```

In the first example, "file_name" contains a single module that is write-protected. A new file called "new_file_name" is created, which contains a copy of the module with its status changed to write-enabled. Note: The header parity and CRC of the output module are incorrect and must be updated using "VERIFY" before the module can be used.

In the second example, the input file contains three different modules. The first two are write-enabled, while the third is write-protected. Since WPROT simply inverts the existing write-protect bit, the first two are changed from enabled to protected and the third from protected to enabled. In this example, the output of WPROT is "piped" directly to VERIFY (with the "U" option enabled) in order to update the module's header parity and CRC. This eliminates the separate verification step required in the first example and produces an output file that is ready to use.

The program WPROT is copyright 1984 by GIMIX, Inc. All rights reserved. Any questions concerning its use should be directed to GIMIX.

INIZ

Attach device(s)

SYNTAX: INIZ <devicename> [<devicename> <devicename> ...]

FUNCTION: INIZ is used to clear pending interrupts from serial I/O devices that use the 6850 ACIA (such as GIMIX 1, 2, or 8-port serial boards). If the system is RESET while an interrupt from one of these devices is pending, the interrupt is not cleared by the reset. If the system is rebooted, the operating system cannot identify the source of the interrupt. Unidentified interrupts can prevent the system from booting or cause a reduction in system performance (see the INTERRUPTS section of the OS-9 GIMIX III MANUAL ADDENDA).

INIZ clears the interrupt by "attaching" (I$Attach) each device in a list of devices given on the command line. If no list is given, INIZ attempts to read a list of devices from the standard input path.

INIZ should be included in the "startup" file of any system that includes 6850 type serial devices. The device list should include all such devices, for which device descriptors are present in the boot file (OS9Boot); with the exception of "TERM" and any non-sharable devices (usually printers). It is not necessary to initialize "TERM", and the command "INIZP" is used to initialize non-sharable devices (see the description of the INIZP command).

INIZ can also be used to permanently "attach" any sharable I/O device(s) (parallel ports, intelligent ports, etc.) to the system. Using INIZ in this way prevents an unnecessary "termination" and re-initialization of the device that would normally occur, for example, each time a user logs off of the system.

EXAMPLE:

```
 OS9: INIZ  T1  T2  T3
```

If INIZ is unable to open one of the devices listed, an error is returned, and the remainder of the device list is not processed.

INIZP

Initialize device(s)

SYNTAX: INIZP </devicename> [</devicename> </devicename> ...]

INIZP, like INIZ, is used to clear pending interrupts from I/O ports that use a 6850 ACIA (such as GIMIX 1, 2, or 8-port serial boards). However, INIZP does not permanently attach the device as INIZ does. INIZP must be used if the device is non-sharable, since attaching a non-sharable device would render it inaccessible.

Generally, device descriptors for printer ports are made non-sharable to prevent more than one user from accessing the printer at the same time. The printer device descriptors supplied by GIMIX (P, P1, IP1, etc.) are non-sharable.

INIZP should be included in the "startup" file of any system that has non-sharable device descriptors for 6850 ACIA type I/O ports included in its "OS9boot". (For sharable devices, and ports that do not use the 6850 ACIA, see the INIZ command.)

INIZP clears the interrupts by opening a path to each device in the list of devices given on the command line, and then immediately closing the path. If no devices are listed, INIZP attempts to read a device list from the standard input path.

EXAMPLE:

```
 OS9: INIZP /P1 /P2 /P3
```

If INIZP is unable to open one of the devices listed, an error is returned, and the remainder of the device list is not processed.

NOTE: The INIZP command requires the "/" character before each device name as shown in the above example, while the INIZ command does not.

HDERR

Report Errors From OMTI Hard Disk Controller

SYNTAX: HDERR

FUNCTION: HDERR is used to return information from an OMTI type hard disk controller's built-in error log. NOTE: This facility is only available in systems using the OMTI controller. XEBEC controllers DO NOT support this function, and HDERR should NOT be used in systems with XEBEC controllers. Systems with the OMTI controller can be identified by the "OMTI" device driver in the boot file or module directory. XEBEC controllers use a different driver: "XBC".

HDERR first attempts to open a path to the controller through the device "/H0". Once a path is opened, HDERR issues the OMTI controller's "request logout" command twice, once for each of the two possible drives that the controller can support. It then prints (in decimal) the information returned by the controller.

The OMTI controller returns two counts for each of the two drives or Logical Unit Numbers (LUN 0 = /H0, LUN 1 = /H1). The first count is the number of retries that have been performed and the second is the number of "hard" errors that have occurred. Note: The error log is cleared when the system or controller is reset, or when the error log is read by HDERR.

The information supplied by the error log can be useful in determining the condition of the hard disk(s). OS-9 does not perform any retries when accessing the hard disks, the controller is programmed to perform its own retries if an error occurs. When an error occurs, the controller uses re-reads, restore operations, and error correction if necessary, to try and correct it. After 8 unsuccessful "retries", the error is considered permanent, the hard error count is incremented, and the appropriate error is returned to the driver.

HDERR should be used before problems become apparent through errors reported by OS-9. To be most useful, HDERR should be used on a regular basis, and a record kept of the reported errors. A certain number of retries is normal, and is usually caused by soft errors that are re-read correctly; however, a drastic rise in the number of retries and errors, over a comparable period of time and system use, may indicate potential problems with the drive(s) or controller.

For more information, see the OMTI controller manual.

EXAMPLES:

```
 OS9: HDERR

      Retries for LUN 0: 0     Permanent Errors: 0

      Retries for LUN 1: 5     Permanent Errors: 0

 OS9:
```