DATA GENERAL CORPORATION
Southboro, Massachusetts 01772
(617) 485-9100

DOS OPERATOR'S GUIDE

January, 1972

093-000076-00

CONTENTS

I. WHAT IS DOS?

II. SECTIONS OF DOS

III. LAYOUT OF DISK

IV. DOS BOOTSTRAP

V. FILE NAMES

VI. CLI

VII. CLI COMMANDS

VIII. PROGRAMMED DOS COMMANDS

IX. SYSTEM GENERATION

X. MULTIPLE FILE DEVICES

- Devices Providing Multiple File Access
- Directory Devices
- Magnetic Tape
- Initializing a Tape Drive
- Releasing a Tape Drive
- Referencing a File On Magnetic Tape
- Writing Files To Magnetic Tape
- Creating a System Tape For Bootstrapping DOS
- User Serviced Interrupts
- Panics

XI. BATCH PROCESSING

I. WHAT IS DOS?

DOS is a highly sophisticated utility program with which the user can perform numerous operations with simple commands. All commands to DOS request the system to perform some operations on files. This is the key to the power of DOS. Files may be pre-named storage areas on disc, or files may be I/O peripheral device drivers. To DOS, both look alike. A file is simply an area in some storage medium, where information can be retained. This storage medium may be paper tape, punched cards, magnetic tape, or magnetic disc. Files may be manipulated, edited, assembled, loaded, etc. by DOS. In fact, programs, while running may utilize DOS with special call instructions. DOS eliminates the need for the user to concern himself with disc storage housekeeping and I/O driver routines.

II. SECTIONS OF DOS

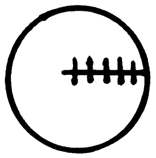
DOS comprises two (2) distinct sections: The Disc Operating System (DOS), and the Command Line Interpreter (CLI). In a system with 16K of core, 256K fixed head disk, reader, and punch, DOS requires about 4650 decimal words of store. The CLI requires about 5.5K of store, when in core. When the system is in use, the DOS section is always residing in the upper 4.5K of core and locations 0 thru 15. DOS is the section that executes all commands, updates the directories of the files that exist on disk, handles all I/O servicing, and controls the I/O interrupt facilities.

Commands to DOS can originate from user programs, or from the program called CLI. CLI is a 5.5K program that enables a user to communicate with DOS from the Teletype Keyboard. The CLI is stored on disk and is brought into core when needed. Commands entered from the console to the CLI are formatted and stored on disk in the file called COM.CM. When the operator types carriage return or form feed, the CLI's job is through and it simply tells DOS to overlay the requested program, which then reads the COM.CM file.

Resident on disk is a directory of the files that exist on disk. This directory resides on disk along with a duplicate of the DOS and CLI programs. Anytime a file is altered, DOS immediately updates the disk file directory. This assures the user that the system and file directory can always be recovered and restored in the event of system shutdown. The only critical time for a system shutdown to occur is during the actual disk file updating. This failure is highly improbable.

III. LAYOUT OF DISK

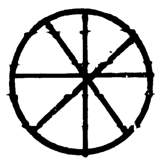
Our fixed head, head-per-track disk is a single disk platter with 32, 64, or 128 heads mounted on a radial line. In actuality, however, space limitations require the heads to be staggered.



The circumferential line that travels beneath a head is called a track. Thus a 32-head disk has 32 tracks, etc. By convention, the outermost track is track 0, and the track numbers increase inward.

Each track has a storage capacity of 2048 16-bit words. For ease of file keeping, each track is segmented into 8 sectors, or blocks, numbered from 0 thru 7. Thus, a 64K disk has 32 tracks and 256, 256-word blocks of storage. A 128K disk has 64 tracks and 512, 256-word blocks of storage, and a 256K disk has 128 tracks and 1024, 256-word blocks of storage.

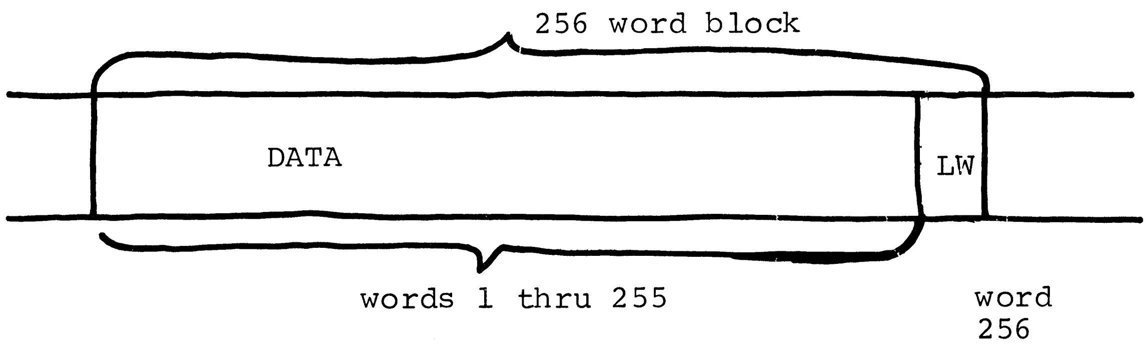
The disk layout is compatible with the disk interface since the interface is designed to do a complete 256-word block transfer whenever a disk transfer is initiated.



This block layout of the disk is the basis of the file directory, retained by DOS. The file directory contains, <u>in essence</u>, only two (2) entries for every unique file on disk. One entry is the unique file name itself, and the other entry is the disk address of the first 256 word block where that file is stored.

File Directory
FOO
disk address 3
JOE
disk address 12
EDIT.SV
disk address 6
ASM.SV
disk address 8

"
"



Within that 256-word block of the file, the first 255 words are the file contents; whether they be data, source, or binary information is immaterial at this time. The 256th word in the block contains a pointer to the next 256 word disk block where this file is stored. This pointer is known as a <u>link word</u> (LW). This link word is used by DOS to determine the next disk address where this file is stored. DOS computes this next address as follows:

$$\text{last disk address} \oplus \text{LW} = \text{next disk address}$$

Since the relationship:

$$\text{next disk address} \oplus \text{LW} = \text{last disk address}$$

also holds true, DOS can read <u>back</u> through a file as well as <u>forward</u>.

A number of blocks on the disk are needed to store DOS, the CLI, a disk map dictionary, and a DOS bootstrap program. The CLI and DOS utilize approximately 62 blocks of disk space. The disk map dictionary is variable in size, depending upon the disk size in the system. This dictionary utilizes 1 bit of disk space for every block of disk. Thus, the dictionary for a 64K disk uses 1 block (256 bits), for a 128K disk it uses 1 block (512 bits), and for a 256K disk it needs 1 block (1024 bits) of disk space. This map dictionary (MAP.DR) on the disk is utilized to keep track of the free storage areas on the disk. Another block is utilized to store a special DOS bootstrap program whose function it is to reload the complete system into core, from the disk.

The minimum disk overhead required by DOS is, therefore:

USE	# BLOCKS REQ'D
Bootstrap	1
File Directory	1 (min)
Map Directory	1 (min)
Reserved blocks for overlays	3
DOS + CLI	62
	68 = 17408 words

A number of programs are supplied with, and will operate under, DOS: Editor, Relocatable Assembler, Relocatable Loader, Debug III, Fortran IV Compiler, Algol 60 Compiler, an octal editor (OEDIT), etc. These programs may be left on disk, or may be loaded only when needed, thus freeing the disk space for user storage.

When residing on disk, these programs utilize the following number of blocks, in addition to those reserved for DOS as indicated above:

EDITOR	9
RELOCATABLE ASSEMBLER	18
RELOCATABLE LOADER	10
DEBUG III	11
FORTRAN COMPILER	43
FORTRAN RUN TIME LIBRARIES	122
ALGOL COMPILER	80
ALGOL RUN TIME LIBRARIES	121
ALGOL TRACE (DEBUG) PROGRAM	37
OCTAL EDITOR (OEDIT)	4

Until now, only the fixed head disk has been referenced as the bulk storage medium. Also available as bulk storage devices are removable pack, moving arm disks. These disks differ from the fixed head disks in operation, but not in principle. The fixed head disk has two surfaces, with magnetic pick-up heads permanently positioned over the disc. Switching from one track to the other is done electronically. The moving arm disks usually have more than one storage surface. A moving arm, containing one magnetic pick-up head per surface, is moved along the disk surface to position itself over a particular track. Thus, switching from one track to another on the same surface is a mechanical process. However, switching from one surface to another is electronically accomplished. The combination of equivalent tracks on all the surfaces is known as a cylinder.

The differences in the discs appear to DOS, essentially, only in the form of access speed variations. The following comparisons can be made:

	HEADS	A SURFACES	B TRACKS	C BLOCKS PER TRACK	D CAPACITY (BLOCKS) A*B*C	CAPACITY (WORDS) D*256
4019A	fixed	2	32	8	256(B*C)	65,536
4019B	fixed	2	64	8	512(B*C)	131,072
4019C	fixed	2	128	8	1024(B*C)	262,144
4047A	moving	2	203	12	4872	1,247,232
4047B*	moving	4	203	12	9744	2,494,464
4048A	moving	10	203	6	12180	3,118,080

*This is 2 4047A disks with no door on one of them, thus prohibiting pack removal.

If more than one fixed head disk is in the system (up to 8), it appears to DOS as only one disk (DKØ) of expanded size, and the disk overhead is still a minimum of 68 blocks. Each moving arm disk (up to 4) appears to DOS as a unique device (DPØ, DP1, DP2, DP3). If one of the DP disks is used to retain the system (the bootstrap device), it must be DPØ, and DPØ has a minimum disk overhead of 68 blocks. DP disks used only for file storage (do not have copies of CLI and DOS), have a minimum disk overhead of only 6 blocks. DK and DP disks may be mixed in a system.

IV. DOS BOOTSTRAP

In the event that the system is turned off, or programs are executed without DOS, thus destroying the core resident DOS, it becomes necessary to restore the DOS to core by loading it from the disk. As explained in section III, there is 1 block on the disk which contains a special bootstrap program whose function it is to restore DOS to core from the disk. By executing this bootstrap program, an automatic loading, or bootstrapping procedure is initiated.

DOS can be bootstrapped only from the disk which was specified as the "bootstrap device" at sysgen, and only if that disk was given the INSTALL command, causing the Bootstrap Program to be written onto track #0, sector #0 of that disk. In addition, the only bootstrappable disks are the fixed head disk (DKØ) or DPØ. The reason for bootstrapping being available only on sector #0 of track #0 of unit #0 will become evident.

To understand the process, it is necessary to understand the disk interface. In order to accomplish a disk transfer of a block of data, the processor must know the two end points of the transfer: The memory address and the disk block address. The processor obtains these 2 pieces of information from the A and B buffers in the disk interface.

It is important to note at this time, that pressing the RESET switch on the front console resets the entire system, including the A and B buffers on the disk interface. If a disk transfer were initiated at this time, the disk block address would be unit 0, track 0, sector 0, and the memory address would be location 00000. For this reason, the bootstrap program is always stored on TRACK 0 of SECTOR 0 of UNIT 0 and is always loaded into the first 256 core locations: Addresses 00000 thru 00377.

To do a BOOT of DOS on the Supernova:

1. Press RESET
2. Set data switches to 000020 (Fixed head disk device code) or 000033 (Moving arm disk device code)
3. Press CHANNEL START
4. Message "DOS REV Ø4
   R"
   should be typed by the CLI.

To do a BOOT of DOS on the NOVA 800 or NOVA 1200 with the program load option:

1. Press RESET
2. Set data switches to 100020 (Fixed head disk device code) or 100033 (Moving arm disk device code)
3. Press PROGRAM LOAD
4. Message "DOS REV Ø4
   R"
   should be typed by the CLI.

To do a BOOT of DOS on the Nova, Nova 800, or Nova 1200 without the program load option:

1. Press RESET
2. Set data switches to 000376
3. Press EXAMINE
4. Set data switches to 060120 for the fixed head disk. (Set data switches to 060133 for the moving arm disk.)
5. Press DEPOSIT
6. Set data switches to 000377

7. Press DEPOSIT NEXT
8. Set data switches to 000376
9. Press START
10. Message "DOS REV Ø4
    R"
    should be typed by the CLI.

If a magnetic tape unit exists in the system, it is possible to make the magnetic tape unit the bootstrap device, thus relieving the 62 block DOS+CLI disk overhead, or just to be used as a system back-up medium.

1. Mount special system tape (see Chapter X) onto magnetic tape unit #0 and position the tape to the load point.
2. Press RESET
3. A. SUPERNOVA: Set data switches to 000022 (MTA device code) and press CHANNEL START
   B. N1200/N800 with Program Load: Set data switches to 100022 (MTA device code) and press PROGRAM LOAD.
   C. Nova and N1200/N800 without Program Load: Follow steps 2 thru 9 above, changing step 4 to 060122.
4. Message "FULL (0) OR PARTIAL (1)?" is typed
5. Type "1"

The disk overhead is now a minimum of 6 blocks. If it is desired to copy DOS+CLI onto the disk to make it a bootstrap device, type the command:

```
    XFER   MTØ:1   SYS.SV;   CHATR   SYS.SV  SP;  INSTALL   SYS.SV↵
```

V. FILE NAMES

A complete file name may take either of the following forms:

```
FILE NAME
FILE NAME.FILE NAME EXTENSION
```

The FILE NAME may be up to 10 characters in length. The characters may be any of the 37 symbols A thru Z, 0 thru 9, or $.

The FILE NAME EXTENSION, when used, must be separated from the FILE NAME by a period (.). If the extension is used, it becomes part of the complete file name. The extension may be up to 2 characters in length, and these characters may be any of the 37 symbols A thru Z, 0 thru 9, or $.

The extension provides the user with a means, if he desires, of writing a code which identifies the file as being of a certain type. Except for the extensions SR, RB, SV, and LS, the user is free to make his own assignments.

Some file names reserved by DOS at this time are:

```
COM.CM
ASM.SV
EDIT.SV
RLDR.SV
SYS.LB
$TTI                   ;(EOF is the character CTRL Z)
$TTR                   ;(EOF is CTRL Z for ASCII, and
$TTO                     time-out for binary or ASCII)
$TTP                   ;(no leader/trailer with $TTO)
$PTR                   ;(leader/trailer, no tab simu-
$PTP                     lation or line feeds with $TTP)
$CDR
$LPT
$PLT
MTn.ff
```

For protective purposes, files may be given attributes which, when employed, limit the use of these files. The five (5) possible attributes are:

P - Permanent file - (cannot be renamed or deleted)
S - Save file (absolute core image)
W - Write protected file (cannot be written)
R - Read protected file (cannot be read)
A - Attribute protected (attributes cannot be changed)

If more than one unique disk exists in the system, each such disk contains a file directory of the information contained in its storage. Since more than one file directory can exist at any one time, DOS must be aware of which directory it is to reference. At sysgen time the master storage device is specified when the response DSK or DKP is given to the query

```
ENTER     MASTER     DEVICE
```

Initially, and after a bootstrap, it is the file directory on the master storage device that DOS references. If any commands are given which refer to a file, DOS searches for that file on the master storage device. If it is desired to have DOS search for files on another disk, which has its own file directory also, DOS must be given the command to "change directory devices." This is done by giving the command DIR followed by the device we wish to become the directory device.

```
example:     DIR   DP3↵          (DOS will now search for files
                                  on disk pack 3)
```

Changing directory devices causes DOS to search for <u>all</u> files on that new disk. It is possible to reference individual files on <u>any</u> disk or magnetic tape without changing the directory device. This is done by preceeding the filename with the device at where it is stored.

```
example:    DP2:BOB   (references file BOB on disk pack 2)
            DK0:BOB   (references file BOB on the fixed head disk)
```

In the case of disk pack units, it is possible to remove packs containing files and to insert new packs (either virgin packs or packs containing older files). Before a disk pack (or magnetic tape) is removed, type the command RELEASE followed by that device name.

```
example:    RELEASE DP1↵   (allows removal of the disk pack from
                            disk pack drive 1)
            RELEASE MT2↵   (rewinds the tape on magnetic tape
                            unit 2 for removal)
```

If the newly inserted disk pack contains useful files, give the command:

```
            INIT  DP1↵
```

This initializes disk pack unit 1 and retains the free storage map directory and the file directory that were on it.

If the newly inserted disk pack is a virgin pack, give the command:

```
            INIT/F   DP1↵
```

This initializes disk pack unit 1 and builds a new free storage map directory and file directory.

VI. CLI

The CLI is in core and available for commands when the ready message "R" has been printed by the CLI.

DOS doesn't execute any commands until the CLI has received a Carriage Return or Form Feed. Within a command line there may be several commands, each must be separated by a semi-colon (;). No commands are executed until a CR or FF is received. The CLI builds a specially edited file of the command line, deposits this into the file COM.CM, and then turns control over to DOS.

If the command line is to be carried over onto the next line of the Teletype, an up arrow (↑) should be typed prior to pressing the CR. The ↑ causes the next CR to be ignored as the end of command line delimiter.

The command line itself may be edited by utilizing the RUBOUT key for character deletion and the back slash (\) (shift + L) for entire line deletion.

Any program can be interrupted (including the CLI) by typing CTRL A. The CLI then types "INT." This form of interrupt destroys the interrupted program in core, making it impossible to CONTINUE the interrupted program.

Interrupting a program by typing CTRL C causes the CLI to type "BREAK." This form of interrupt leaves the interrupted program in a state of possible continuance, since the program is not interrupted until a logical breaking point is encountered.

Arguments within a command line may be repeated any number of times by use of the numeric switches. The argument to be repeated is followed by a slash (/) then by the number of times it is to be repeated. This is known as a numeric switch.

A letter switch is unique to the command word involved, however, if used, it follows the (/) as does the numeric switch. If the letter switch is appended to the command word, it becomes a global switch which applies to all the arguments. If the letter switch is appended to an argument, it becomes a local switch which applies to only that argument.

An asterisk (*) used in place of any letter of a file name, within a command, instructs DOS to refer to every file name that matches, independent of the character that appears in the position of the (*). A single (*) may refer to the entire file name or extension.

Files containing command sequences may be built and executed. The command string is entered into a file, and to execute this command string the file is referred to indirectly by enclosing its file name within the symbol (@).

VII. CLI COMMANDS

XFER

"transfer the contents of one file to another file"

```
Globals:   A
example:   XFER $PTR BOB.SR↵
```

TYPE

"XFER/A file $TTO"

```
example:   TYPE BOB.SR↵
```

PRINT

"XFER/A file $LPT"

```
example:   PRINT BOB.SR↵
```

BPUNCH

"XFER file $PTP"

```
example:   BPUNCH $PTR↵
```

PUNCH

"XFER/A file $PTP"

```
example:   PUNCH BOB.SR↵
```

DISK

"LEFT: #blocks USED: #blocks"

```
example:   DISK↵
```

LIST

"gives file names, number of bytes in the file, and the file attributes"

```
Globals:   A   L   B
example:   LIST BOB.*↵
```

CREATE

"makes an entry into the file directory"

example:
```
CREATE AL.XX↵
```

DELETE

"deletes an entry from the file directory"

Globals: V

example:
```
DELETE DP3:JOE*.*↵
```

APPEND

"creates a new file by appending several existing files"

example:
```
APPEND MT1:3  DP1:BOB  DP0:AL   SAM   $PTR/3↵
```

RENAME

"change the name of a file in the file directory"

example:
```
RENAME   BOB   JOE↵
```

CHATR

"Change the attributes of a file"

example:
```
CHATR   SYS.SV   *SP↵
```

DUMP

"transfer files from disk, in special DOS format"

Globals: A V

example:
```
DUMP/V   MT1.12   BOB.SV↵
```

LOAD

"reload DOS formatted files onto disk"

Globals: A V

example:
```
LOAD/V   $PTR↵
```

SAVE

"rename the contents of BREAK.SV"

```
example:  SAVE JOE.SV↵
```

EDIT

"bring the Editor into core for execution"

```
Input file information is supplied by the command   *GRBOB1$$
Output file information is supplied by the command  *GWBOB2$$
When the input and output files are no longer needed,
     give the command  *GC$$
To return to the CLI, give the command  *H$$
Other Editor commands are similar to those of the DGC stand
     alone Text Editor.
```

```
example:  EDIT↵
```

ASM

"bring the Extended Assembler into core for execution"

```
Globals:  L   N   U   E   S   T   X
Locals:   B   L   S   N
example:  ASM /U/E/X   $PTR/2   AL   $LPT/L   JOE   BOB.RB/B↵
```

ALGOL

"bring the Algol compiler into core for execution"

```
Globals:  L   N   E   S   A   B
Locals:   B   L   S
example:  ALGOL/E   BOB   $LPT/L↵
```

FORT

"bring the Fortran compiler into core for execution"

```
Globals:  L   N   E   S   A   B   F   X
Locals:   B   L   S
example:  FORT/E   BOB   $LPT/L↵
```

RLDR

"bring the Relocatable Loader into core for execution"

```
Globals:  A    D    L    S    Z
Locals:   U    S    N
example:  RLDR/A/D   JOE.SV/S   $PTR/3   BOB   $LPT/L   SAM↵
```

FILENAME

"bring this file into core for execution"

```
example:  BOB↵
```

DEB

"bring a file into core but execute at the Debugger starting address"

```
example:  DEB   BOB↵
```

MKABS

"make an absolute binary object file from a save file"

```
Globals:  Z
Locals:   S
example:  MKABS   BOB.SV   $PTP   400/S↵
```

MKSAVE

"make a save file from an absolute binary object file"

```
Globals:  Z
example:  MKSAVE   $PTR   BOB.SV↵
```

OEDIT

"bring the Octal Editor into core for execution"

```
example:  OEDIT BOB.SV↵
```

LFE

"bring the Library File Editor into core for execution"

```
Arguments:  A   A/M   D   I   M   N   R   X
Locals:     A   B   L   O
example:    LFE   A   SYS.LB   $LPT/L↵
```

VIII. PROGRAMMED DOS COMMANDS

Commands may be passed to DOS from user programs by including the calling sequence within the program.

```
________
________
________
.SYSTM                     ;DOS Call-assembles as JSR @2
command or command n       ;DOS Command-n=0thru7,77
abnormal return            ;error return location
normal return
________
________
________
```

Since the .SYSTM call is actually a JSR instruction, the contents of AC3 are destroyed. On a normal return, DOS restores AC0, AC1, AC2, and C as they were, and AC3 gets restored with the contents of location 16. If the user program stores the (AC3) into location 16 prior to doing the .SYSTM call, then DOS will restore C and all AC's as they were. On an error return (AC2) contains the error code. If n=0thru7, DOS uses that number; if n=77, DOS uses (AC2).

.INIT

"analagous to the CLI command INIT"

(AC0) = BPTR to device name
(AC1) = 1 for /F
      ≠ 1 for partial

.DIR

"analagous to the CLI command DIR"

(AC0) = BPTR to device name

<u>.RLSE</u>

"analagous to the CLI command RELEASE"

(AC0) = BPTR to device name

<u>.INST</u>

"analagous to the CLI command INSTALL"

(AC0) = BPTR to device name

<u>.CREAT</u>

"analagous to the CLI command CREATE"

(AC0) = BPTR to file name

<u>.DELET</u>

"analagous to the CLI command DELETE"

(AC0) = BPTR to file name

<u>.RENAME</u>

"analagous to the CLI command RENAME"

(AC0) = BPTR to old file name
(AC1) = BPTR to new file name

<u>.OPEN n</u>

"links a file name to a channel number n, such that all further references to that file may be made through the channel number"

```
(AC0) = BPTR to file name
(AC1) = file characteristic inhibit (set=0 in normal cases)
        n=0 thru 7 for desired channel number.  If n=77,
        then DOS takes (AC2) as the channel number.
```

<u>.CHATR n</u>

"analagous to the CLI command CHATR"

(AC0) = new attributes

.GTATR n

"obtains the present attributes of a file"

(AC0) receives the file's attributes

Input/Output (I/O) under DOS may be done on a Line, Sequential, or Random Basis.

Line I/O:	ASCII character string ending with a CR or FF
Sequential I/O:	byte image format with a byte count required
Random I/O:	image format by record number

.APPEND n

"analagous to the .OPEN n command. Normally data written to a file begins at the start of the file and overwrites existing data. The .APPEND n command causes data written to be appended to the existing contents of the data file."

(AC0), (AC1) n = (see .OPEN n command)

.CLOSE n

"breaks the link between a file name and a channel number"

.RESET

"breaks all channel number links"

.RDL n

"reads a line from channel n"

On input from $CDR, DOS does a Hollerith to ASCII conversion.
Normal return occurs on a CR or FF.
Abnormal return occurs if 132 characters are input.

(AC0) = BPTR to user input buffer area
(AC1) receives byte count of number of characters input

.RDS n

"reads a string of bytes from channel n"

(AC0) = BPTR to user input buffer area
(AC1) = byte count of number of bytes to be read.
On input from $CDR, this count must be even (2 bytes/col).
If EOF occurs, (AC1) receives byte count of number of bytes input.

.RDR n

"reads one 64-word record from channel n"

(AC0) = Memory address of user input buffer area
(AC1) = Record number (record numbers start at 0)

.WRL n

"writes a line to channel n"

Output is terminated by a null, CR, FF, or 132 characters.

(AC0) = BPTR to user output buffer area
(AC1) receives byte count of number of characters output

.WRS n

"writes a string of bytes to channel n"

(AC0) = BPTR to user output buffer area
(AC1) = byte count of number of bytes to be written

.WRR n

"writes one 64-word record to channel n"

(AC0) = Memory address of user output buffer area
(AC1) = Record number (record numbers start at 0)

.GCHAR

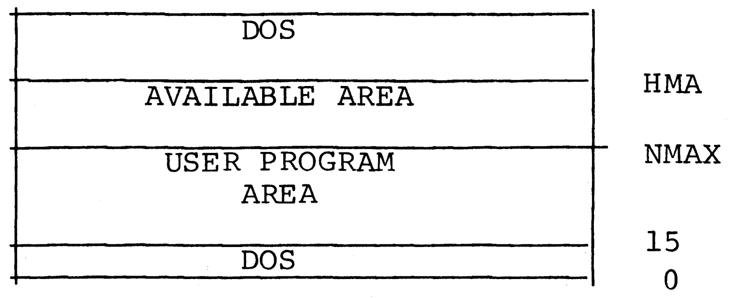
"inputs one character from $TTI"

($AC0_{0-8}$) = 0
($AC0_{9-15}$) = ASCII character input

.PCHAR

"outputs one character to $TTO"

($AC0_{9-15}$) = ASCII character to be output



.MEM

"find NMAX and HMA"

(AC0) receives HMA
(AC1) receives NMAX
SUB 1,0 returns size of Available Area

.MEMI

"adjust NMAX"

(AC0) = ± D, where new NMAX = old NMAX ± D

.EXEC

"saves core and overlays another program"

There are up to four levels of overlay:

	CLI	A	B	C
level:	0	1	2	3

The core saved is: (location 16) thru the higher of
(location SST) or NMAX

(AC0) = BPTR to file name of program to be brought into core
(AC1) = Starting Address of new program:
= 0, DOS starts at the program's starting address
= 1, DOS starts at the Debugger address

.RTN

"returns previous level program to core and begins execution at the normal return address"

DOS restores C, AC0, AC1, AC2, and AC3 (gets (16)).

<u>.ERTN</u>

"returns previous level program to core and begins execution at the abnormal return address"

(AC2) receives (AC2) from terminated overlay.

If the return is to the CLI, the CLI prints the error message associated with that error code in AC2.

<u>.BREAK</u>

"analagous to CTRL C in the CLI"

IX. SYSTEM GENERATION

(see Appendix B of the DGC "Disk Operating System User's Manual" #093-000048-03)

X. MULTIPLE FILE DEVICES

Devices Providing Multiple File Access

At present there are three possible types of devices on which system and user files can be stored that the Disk Operating System can readily access for multiple reading and writing. These are:

1. Fixed head disk - Usually one per system configuration, having the mnemonic:

   DK0

   If there were more than one fixed head disk controller, the second controller would be designated DK1.

2. Disk pack of the movable head disk - There can be up to four disk packs per disk and they are designated:

   DP0, DP1, DP2, DP3

3. Magnetic tape units - Up to eight magnetic tape drives are permitted per system, and they are designated:

   MT0, MT1, . . . MT7

Directory Devices

Directory devices are those devices that have their own file directory, containing the names, attributes, and byte counts of all files stored on the device. Only disk packs and the fixed head disk can maintain such a file directory.

Files are stored on magnetic tape by file number, the number indicating the order in which they were written onto the tape. DOS accesses files on magnetic tape by their file number, not by referencing a file name in a directory.

Magnetic Tape

DOS has access to files on magnetic tape. The DOS system will support up to eight 9 track magnetic tape drives. The system allows up to 100 files to be written onto a given tape. Reading and writing is at high density (800 bpi). If the unit specified is a 7 track drive or is selected to low density, the message:

```
UNIT IMPROPERLY SELECTED
```

will be given.

Initializing a Tape Drive

Initializing a tape drive causes the tape on that drive to be rewound. Full initialization (/F switch) will cause the tape to be rewound and two EOF's to be written.

Releasing a Tape Drive

To rewind a tape drive, the RELEASE command can be given.

Referencing a File on Magnetic Tape

Files are placed on tape in numeric order, beginning with the file 0. Up to 100 files may be put on a given tape; the last permissible file is 99.

A given file is referenced in a command by a tape drive specifier followed by file number. Either a one-digit or two-digit number may be used to reference the first ten file numbers, i.e.

MT1:04 and MT1:4 are equivalent.

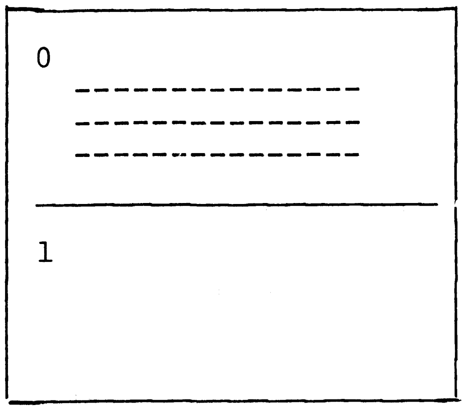
Both the tape drive specifier and the file number must be given. The file number must be in the range 0-99 and the tape drive unit number must be in the range 0-7.

Writing Files to Magnetic Tape

Files must be placed on magnetic tape in numeric order. For example, suppose the user transfers a file to tape that has just been initialized:

```
XFER FILE0   MT0:0↵
```

FILE0 will be the first file on the tape. The tape on drive 0 will now contain the following:



First file, containing contents of FILE0.

Once a file is written, the file number of the next file is assigned. File 1 is a null file.

An attempt to place a new file on the tape above with one of the following commands:

```
XFER FILEX MT0:2↵
```

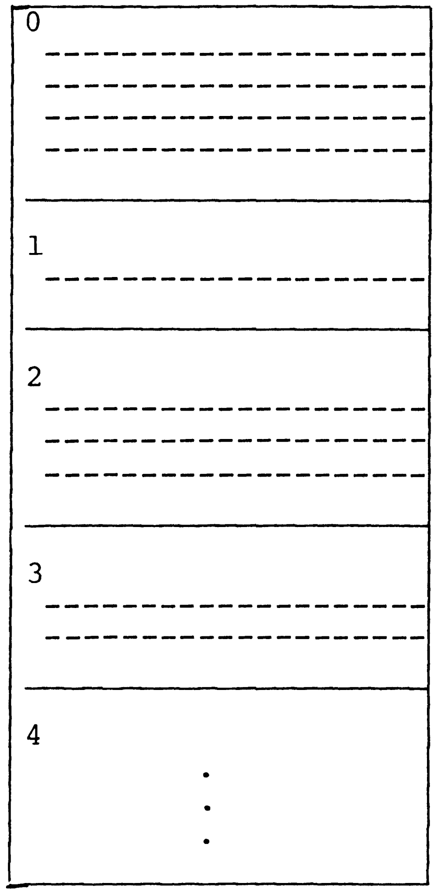
where only file 0 has been written to tape

```
XFER FILEY MT0:4↵
```

will result in an error message:

```
ILLEGAL FILE NAME
```

It is possible to overwrite a magnetic tape file. For example, assume a tape on drive 0 contains four files:

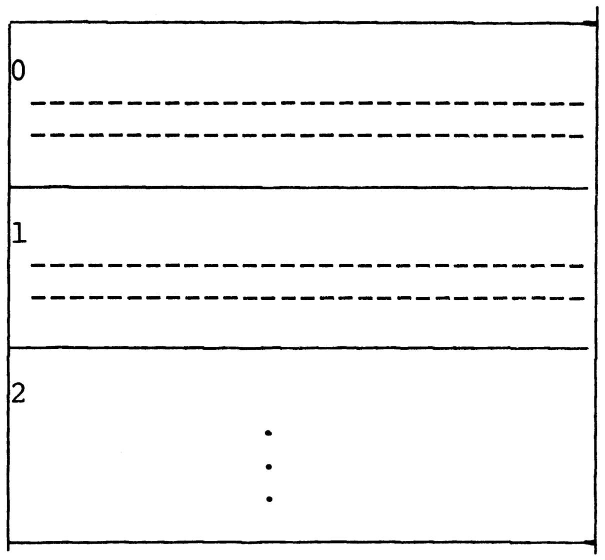


Null file.

The command:

```
XFER MYFILE MT0:1↲
```

will cause the contents of MYFILE to overwrite the tape beginning at the file 1 position. When a tape file is written in this manner, all subsequent files on tape are lost. In the example, the tape will contain:



Creating a System Tape for Bootstrapping DOS

1. Select an unused magnetic tape, mount it on the tape drive, and give the command:

```
INIT/F   MTn↵     (where n is the tape drive number)
```

2. Load tape #088-000015 into the $PTR, and give the command:

```
LOAD/V $PTR↵
```

3. Give the command:

```
XFER  TBOOT.SV  MTn:0↵
```

4. Give the command:

```
XFER  SYS000.SV  MTn:1↵
```

With the presence of magnetic tape in the system, it is usually desirable to dump the DGC system programs onto the system for reloading following a bootstrap.

User Serviced Interrupts

A facility has been provided for the user to service his own interrupts using DOS. The procedure and restrictions are outlined on the next page.

Two parameters are defined mnemonically by the Disk Operating System parameter tape 090-000176. The first, UIS, defines a page zero address where the user must store the address of his own interrupt service routine. The second, UMSK, contains the address of a DOS subroutine that properly maintains the interrupt mask word and enables interrupts for the user.

If the user does not change the word at location UIS, a PANIC with code 210 is given when an interrupt is detected that is not recognized by DOS. However, if the user initializes the word at UIS to contain an address within his own program, control is transferred to that address via JSR when an interrupt not recognized by DOS is detected.

When the user receives control, interrupts are OFF and AC0 contains the device code of the interrupting device. The user may examine this code or, alternatively, skip on the appropriate DONE flip flops of the devices from which he expects interrupts. It is the user's responsibility to save AC3 and to return to this address after completion of his service routine, but all other accumulators and carry are saved for him by the system.

If the user can complete his servicing of his device with interrupts disabled and is not concerned about the other devices running under DOS interrupting, he need not make use of the subroutine provided for maintaining the current mask word. In general, however, the user should use this subroutine to mask out his device and all lower priority devices and to turn on interrupts again. To use this routine, the user must supply in AC0 a bit corresponding to the interrupt disable mask bit for every device, including his own, that he considers of lower priority. Note that as a minimum the user _must_ set a bit in AC0 corresponding to his own device interrupt disable flip flop. The calling sequence for this routine is simply,

```
            JSR  @UMSK
```

The system will maintain the mask word properly, turn on interrupts, and return control to the user with AC0 containing the new mask word.

Panics

There are a number of hardware malfunctions that may cause the system to "PANIC." Should a PANIC occur, the contents of the accumulators will be printed on the TTY, followed by a PANIC code. The output will appear as follows:

```
            000015 177777 000011 037500 000210
              AC0    AC1    AC2    AC3   PANIC CODE
```

The PANIC codes are:

210 - Unknown interrupt. Offending device code in AC0.

220 - System stack overflow.

230 - Repeated critical disk write errors.

240 - Repeated critical read errors.

250 - Repeated critical disk read or write errors.

260 - Runaway tape reader. (An NIOC to an input device did not stop its forward motion.)

270 - Fatal magnetic tape hardware status. AC0 contains the magnetic tape controller status.

XI. BATCH PROCESSING

Utilizing the technique of referencing command files, it is possible to produce a batch processing mode of operation.

example: A system containing a card reader ($CDR) is to be used for batch processing student programs written in Fortran, Algol, and Assembly Language.

* the underlined dialogue is generated by the operator

```
R
XFER/A  $TTI  FORTRAN↲
CREATE XX.XX;DELETE XX.* ;XFER/A  $CDR  XX.FS;↑↲
FORT XX.FS  ;RLDR   XX   FORTLIB1  FORTLIB2   FORTLIB3   FORTLIB4;↑↲
XX↲
↑Z ←——— (CTRL Z)
R
XFER/A  $TTI  XALGOL↲
CREATE XX.XX;DELETE  XX.*;  XFER/A  $CDR  XX.AS;↑↲
ALGOL XX.AS;  RLDR XX ALGOLLIB1 ALGOLLIB2 ALGOLLIB3;↑↲
XX↲
↑Z
R
XFER/A  $TTI  ASSEMBLY↲
CREATE XX.XX;DELETE XX.*;  XFER/A  $CDR  XX.SR;↑↲
ASM XX.SR;RLDR XX ; XX↲
↑Z
R
```

Students' program decks should now be loaded into the card reader with one of the following cards placed at the beginning of the program:

```
@FORTRAN@
@XALGOL@
@ASSEMBLY@
```

An EOF card (12,11,0,1 multipunch) should be placed at the end of each program deck.

The computer operator then types:

```
@$CDR@↲
```