USER'S MANUAL

VER. A.4

DOSPLUS (c) (p) 1983 Micro-Systems Software Inc.
DOSPLUS User's Manual (c) (p) 1983 Micro-Systems Software Inc.

This manual is copyrighted material. Reproduction in any form,
without the written permission of the publisher, is prohibited by law.

Acknowledgements

We are very proud to present to you DOSPLUS IV. We feel it to be the premier Disk Operating System for the TRS-80 Model 4, combining speed, power, flexibility, and reliability to create a truly professional system.

We would like to take this opportunity to express our gratitude toward a few people who worked very hard to make this all happen:

System authors : Steve Pagliarulo and Todd Tolhurst
Manual authors : Mark Lautenschlager, Todd Tolhurst, and Rich Leun

If you have problems with your DOSPLUS or wish to ask a question, technical support is available at:

Micro-Systems Software Inc.
4301-18 Oak Circle
Boca Raton, FL 33431
(305) 983-3390
MicroNet: 70271,120
Source: ST8719

Support will be offered also via the Telecom SIG maintained by Micro-Systems on CompuServe. Section 4 is devoted to DOSPLUS questions.

We would request that you reserve only questions of an urgent nature for the telephone systems and send all other problem in via mail. The single number DOES rotor to other lines and there are several people here to help you. From time to time, all technical support personnel will be tied up. If you get a busy signal, please be patient and try again. The Micro-Systems WATS lines are for orders and dealers only! Technical support will NOT be provided there.

Enjoy your DOSPLUS. If you have any suggestions or comments, we would love to hear them. Please take the time to register this system. Technical support for unregistered owners is VERY difficult. If you have already registered a previous version of DOSPLUS and this is an upgrade to you, don't be concerned. You don't have to register again.

Thank you for your patronage! We hope you like our system.

Micro-Systems Software Inc.
Boca Raton, FL
July 1983

Introduction

Welcome to DOSPLUS IV! This unique Disk Operating System is based upon our belief that a program does not have to be confusing to be powerful. This is perhaps the most device independent system ever designed for the TRS-80. This manual is designed to help you get acquainted with DOSPLUS IV and is written in a user-friendly and easy to understand manner. The manual is basically divided into the following sections:

The operations manual. This portion of the manual is aimed at teaching you the concepts behind the system and introducing you to the various parts of the system. It contains information for the first-time user and also information on command syntax and how to operate DOSPLUS.

The library of commands. This portion of the manual covers the library commands. A library command is a "built-in" function of the system. In other words, it is a command that is contained within the actual system files. You will be allowed to purge whatever commands you do not need, but, for the most part, a "minimum system" will consist of these commands.

The DOSPLUS utilities. The section of the manual covers the DOSPLUS utility programs. These are programs that are included with your DOSPLUS that enhance or expand on the capabilities of your library commands. These utilties may be easily removed by the user (if not needed, of course), thus allowing the user to "customize" DOSPLUS in the interests of disk space efficiency.

The Disk BASIC manual. DOSPLUS IV has a set of programs that enhance the capabilities of the BASIC that is included with your TRS-80. This section will cover each of them in turn.

The DOSPLUS IV technical manual. This final portion of the manual contains all of the important RAM and Disk addresses for those of you seeking information on the DOS. It also has documentation of the various system calls and how they function. This area of the manual is a "must read" for the machine language programmer seeking to interface to the system.

We hope that you will be pleased with your DOSPLUS and hope that you enjoy using the system. There are some differences between DOSPLUS IV and other systems, even earlier DOSPLUS', so we strongly suggest that you read the manual before beginning to use the system.

General manual syntax

This manual uses some fairly uniform syntax and to make the use of the manual easier, we should cover that here. The first item is general syntax of entering commands.

Three terms will be used in this manual in regards to responding to the system's request for information (usually called "prompts"). They are "press", "type", and "enter".

Press means to press the single key indicated by the text. Type can either indicate a single keystroke or a series of keystrokes. In either case, when you are instructed to "type" something, you will be given exactly key by key what it is that you are supposed to type. Enter is used when the user is to respond based on a set of valid parameters for that command and fill them in as desired. You will be instructed to "enter" something when we wish YOU to make the choice regarding what is typed.

For the sake of simplicity, we have adopted certain general manual notations. These are as follows:

Capital letters	Word **must** be typed in as shown.
Lowercase letters	Information to be entered by the user based upon a list of valid values and parameters for that command.
Brackets [ ]	Indicates that the information contained within the brackets is optional and may or may not be entered depending upon the situation and the user's desire for clarity.
Parenthesis ( )	Indicates the parameter field. This field is composed of those parameters that will modify the action of the command to suit the user's needs. Please note that the parenthesis are no longer a requirement in DOSPLUS. The parameter field is now indicated as much by position as by the delimiters.
H	Indicates that the value it follows is a hexadecimal value. Used for entering data to the system in hexadecimal format.

The general form of each command will be the command itself, followed by the I/O field, followed by the parameter field. The parameter field and certain portions of the I/O field are optional and included at the discretion of the operator.

First time operation

If this is the very first time that you are using DOSPLUS, the first thing you should do is make a backup of your Master diskette. Before you can make a backup, though, you must first power up the system.

Booting up

After switching on the machine, place your DOSPLUS Master diskette in Drive 0 and press the reset button. This is the orange button in the upper right hand area of the keyboard. If the system fails to boot or reports an error, open the drive door, re-seat the diskette, and try again.

The DOSPLUS header and logo should appear. You will be prompted to enter the date and time. DOSPLUS IV stores the date of last access on every file and this is displayed from the DIR command, so we do recommend that you take the time to set the date on system power up. You may, if you wish, remove the logo, date, and time prompts (or any combination of same) by using the SYSTEM command.

DOSPLUS allows you to enter the date and time in what we call "free form format". Essentially, this means that you may enter the date and time in any form that appeals to you. "MM/DD/YY" is valid; as is "MM/DD/YYYY". You may use any valid delimiters to separate the information. For further examples or a detailed explanation, see the DATE and TIME library commands.

DOSPLUS comes with a unique and powerful program called MEDIC. MEDIC stands for Menu Environment DOS Interface Controller. That is a fancy title for a simple program. What MEDIC does is display some or all of the files from a particular diskette on the screen in alphabetical order and allows you to, with a single keystroke, perform many common functions. You may even execute other programs and return to MEDIC.

After you respond to the "Date:" and "Time:" prompts, DOSPLUS will echo the current date to the screen as confirmation that the date is correct. This is not what will always happen. We have set the DATE library command on the AUTO (see DATE and AUTO) so that as you boot the master diskette, the date will echo. You are welcome to remove this auto execute function or change it to something else at your discretion.

Once the date has been echoed to the screen, DOSPLUS will execute MEDIC. For a complete description of MEDIC, look under "MEDIC" in the utilities table of contents and read the portion of the manual describing it. Suffice it to say for now that you will find yourself in MEDIC. This will greatly simply things. When MEDIC loads, you will be presented with a list of files from the master diskette. The blinking cursor will be positioned over the filename BACKUP/CMD.

Since we wish to make a backup of the master, press ENTER to execute the program. Depending on the number of disk drives you have, follow the appropriate set of following directions.

Backing up with multiple drives

Place whatever diskette you wish to use for BACKUP in drive 1 (the top drive). It does not matter if the diskette is blank or not. After you have pressed ENTER from MEDIC, the display will clear, the backup program header will display, and you will be prompted:

Source drivespec ?

Reply to this with a "0" (a numeric "0", not an alphabetic "O"). This question is asking in which drive the diskette we are backing up FROM is located. Because we are backing up from the Master diskette located in drive 0, we answer accordingly. Following that, you will be prompted:

Destination drivespec ?

Reply to this with a "1". This question is asking you in which drive the diskette we are backing up TO is located. Since we placed this diskette in drive 1, we answer accordingly.

If the current system date is "00/00/00", which would result if you had pressed ENTER or BREAK at the date prompt and no system date was previously set, BACKUP will prompt you:

Backup date ?

At this point, you may enter the correct date or any eight characters that you wish to have appear in the diskette's "creation date" field. Any characters are valid at this prompt.

BACKUP will then read first from the source diskette, then from the destination. If the destination diskette was blank, BACKUP will format the diskette and proceed with the copy. If the diskette was NOT blank, you will be prompted:

Diskette contains data, Use or not ?

At this prompt you have three options.

(1) Abort the backup.

(2) Continue using present destination format.

(3) Continue, but after re-formatting the destination diskette.

To abort the backup, type N and press ENTER. You may also simply press BREAK to abort (at any of the prompts). BACKUP will then flash the message:

Insert System disk [ENTER]

Your DOSPLUS Master diskette should still be in drive 0 at this point, so simply press the ENTER key. You will return to MEDIC and should bring out another disk and try again.

To continue with the backup and attempt to use the current destination disk format, type Y or U and press ENTER. DOSPLUS will then examine the destination disk to determine whether or not the formats are compatible. The system will re-format as little of the destination disk as possible if they are not compatible (to save time) and then proceed with the backup. If the destination disk has a major incompatibility, BACKUP will automatically re-format the entire disk.

To continue with the backup, but force BACKUP to re-format the destination disk first, type F and press ENTER. BACKUP will then re-format the destination disk and proceed with the backup. This is useful when you are not certain of the destination disk's format or when you wish to make sure that no vestiges of the old data exist on the destination disk after the backup.

When BACKUP is actually copying the disk, it will read just as many granules as it can into memory at one time before writing them out to the destination disk. BACKUP will read only those granules that are currently allocated, so if some cylinders seem to be skipped or the numbers change more rapidly, don't be alarmed. Those granules were empty and there was no need to copy them.

Please note that BACKUP makes only "mirror image" copies. That is, the destination disk will be exactly like the source. If it encounters so many flaws that it cannot place data in exactly the same location on the destination disk as it occurred on the source, then BACKUP will abort with an error. To make a "copy by file" backup, use the library command COPY with the wildmask parameters.

BACKUP will also abort on any sort of disk read error. In the event that this should occur and you cannot backup the disk, you may use the COPY command to remove what files you can and preserve whatever data is good.

When BACKUP is finished, it will flash the message:

Insert SYSTEM disk [ENTER]

Insert your backup disk in drive 0 (as a matter of testing) and press ENTER. Your backup is complete. File your Master away in a safe location and use the disk you just made as your "working" Master. You should be returned to MEDIC and proceed to the next portion of this overview.

Backing up with a single drive

Backing up with a single drive is much the same as backing up with multiple drives, except that during the actual copy, BACKUP will be prompting you to switch between the source and destination disks. When you press ENTER from MEDIC, the BACKUP program will load, display its header and prompt:

Source drivespec ?

Reply to this with a "0" (a numeric "0", not an alphabetic "O"). This question is asking in which drive the diskette we are backing up FROM is located. Because we are backing up from the Master diskette located in drive 0, we answer accordingly. Following that, you will be prompted:

Destination drivespec ?

Reply to this also with a "0". This question is asking you in which drive the diskette we are backing up TO is located. Since we are using a single drive to make this backup, we are backing up TO drive 0 as well as FROM it and we therefore answer accordingly.

If the current system date is "00/00/00", which would result if you had pressed ENTER or BREAK at the date prompt and no system date was previously set, BACKUP will prompt you:

Backup date ?

At this point, you may enter the correct date or any eight characters that you wish to have appear in the diskette's "creation date" field. Any characters are valid at this prompt.

BACKUP will then read first from the source diskette, then from the destination. It will prompt you as to when to insert each of them. After inserting each disk as prompted, press ENTER. It is most important that you do not confuse the two disks and insert source instead of destination or vice versa. Also bear in mind that from time to time BACKUP will need to load something from the system disk. When it prompts you for the system disk, insert your Master disk and press ENTER. Please pay attention to the prompts and be careful.

If the destination diskette was blank, BACKUP will format the diskette and proceed with the copy. If the diskette was NOT blank, you will be prompted:

Diskette contains data, Use or not ?

At this prompt you have three options.

(1) Abort the backup.

(2) Continue using present destination format.

(3) Continue, but after re-formatting the destination diskette.

To abort the backup (as in when you don't expect that disk to have data on it), type N and press ENTER. You may also simply press BREAK to abort (at any of the prompts). BACKUP will then flash the message:

Insert System disk [ENTER]

Place your Master diskette in drive 0 and press ENTER. You will be returned to MEDIC and should bring out another disk and try again.

To continue with the backup and attempt to use the current destination disk format, type Y or U and press ENTER. DOSPLUS will then examine the destination disk to determine whether or not the formats are compatible. The system will re-format as little of the destination disk as possible if they are not compatible (to save time) and then proceed with the backup. If the destination disk has a major incompatibility, BACKUP will automatically re-format the entire disk.

To continue with the backup, but force BACKUP to re-format the destination disk first, type F and press ENTER. BACKUP will then re-format the destination disk and proceed with the backup. This is useful when you are not certain of the destination disk's format or when you wish to make sure that no vestiges of the old data exist on the destination disk after the backup.

When BACKUP is actually copying the disk, it will read as many granules as it can into memory at one time before writing them out to the destination disk. BACKUP will read only those granules that are currently allocated, so if some cylinders seem to be skipped or the numbers change more rapidly, don't be alarmed. Those granules were empty and there was no need to copy them.

Please note that BACKUP makes only "mirror image" copies. That is, the destination disk will be exactly like the source. If it encounters so many flaws that it cannot place data in exactly the same location on the destination disk as it occurred on the source, then BACKUP will abort with an error. There is no method to make a "copy by file" backup with only a single disk drive.

BACKUP will also abort on any sort of disk read error. In the event that this should occur and you cannot backup the disk, you may use the COPY command to remove what files you can and preserve whatever data is good.

When BACKUP is finished, it will flash the message:

Insert SYSTEM disk [ENTER]

If you have been following your prompts, you will insert the Master diskette and press ENTER. You will be returned to MEDIC. Insert your backup disk in drive 0 and re-boot the system as a matter of testing the copy you just made. If all goes well and the new disk boots, then your backup is complete. File your Master away in a safe location and use the disk you just made as your "working" Master.

Overview

In this next section of the manual, we will cover such areas as:

(1) File, drive, and device specifications.
(2) Entering commands.

For those of you interested in "customizing" your DOSPLUS, as well as those of you who have upgraded from previous systems, a discussion of the new manner of hard configuring the system is detailed in the SYSTEM command (see SYSTEM).

File, drive, and device specifications

File specifications

We will cover these three areas in that order. First, the file specifications. The only way to store data in a permanent manner and retrieve it later is to place it into a "file". A file is any group of organized data stored on the disk. It can be a program file or simply data, but all data that is stored on a disk is stored in a file. A file can store the data on the disk until you are ready to retrieve it. The data is then accessed through the filename that you assigned it when you opened or last renamed the file. In this sense, your disks are nothing more that electronic filing cabinets.

A file specification (or "filespec" for short) will be in the following general format:

filename/ext.password:ds

filename is a sequence of 1 to 8 characters used to specify which file we are referring to. A filename may contain any alphabetic or numeric characters or any of the special filespec characters. These are detailed below.

/ext is the optional extension. This consists of from 1 to 3 characters used to further specify which file we are talking about. Two files with the same filenames and different extensions are different files. These may contain the same characters as a filename.

.password is an optional file password consisting of up to eight characters. This will be used in conjunction with whatever protection level you set via the ATTRIB command to control access to your file. The password may also contain any of the legal filespec characters.

:ds is the optional drive specifier noting which drive this particular file is stored on. We will cover drive specifiers and what they are used for after filespecs. If you specify a drive specifier, it must correspond to one that us currently defined in the system.

Legal filespec characters. This area differs in DOSPLUS IV from other operating systems. Any standard filename accepted by the other systems is legal in DOSPLUS, but we have added some special characters and loosened the restrictions on filespecs.

In DOSPLUS, filespecs may contain the following:

(1) The letters A-Z.
(2) The numbers 0-9.
(3) The special characters: #, $, %, &, +,^, _, and ~.

In addition to allowing those special characters to appear in filespecs, DOSPLUS will allow you to begin any portion of the filespec with any of the legal characters. In the past, each portion of the filespec (filename, extension, and password) had to begin with an alphabetic character. That is no longer true in DOSPLUS. You may begin each portion with any character you wish.

CAUTION: While this allows you a great deal more flexibility in your filespecs within the DOSPLUS system, it can cause you to create filespecs that are incompatible with the other operating systems. Please keep this in mind when you are assigning filespecs and do not use any of our "special" conventions when creating a file that you wish to transfer to another system.

There can be no blank spaces or illegal characters within the filespec. DOSPLUS will terminate the filespec at the first blank space or illegal character that it encounters. For example, "NO GOOD/DAT" will be seen by the DOS as "NO".

Each portion of the filespec other than the filename has a specific character that indicates to the system which portion of the filespec is coming. For the extension, it is a slash mark (/), for the password a period (.), and for the drive specifier a colon (:). These are not optional. If you wish to use these areas of the filespec, you must precede them with the proper specifiers. If you omit one of these characters, an error will result.

You also may not have an ASCII 03 (end of text) or an ASCII 13 (carriage return) in your filespec as either one of these will signal the end of the command line to the DOS. If you wish to use multiple commands on the same line, append the commands with a semi colon ";" as this indicates an implied carriage return to the system and it will continue to look for more input on the command line. For more information on multiple commands, see the section Built-in features toward the end of the operations manual.

Further examples and details regarding filespecs. Throughout the system, and consequently this manual, we will be dealing with two types of files. These are program files and data files. The type of file is usually known only to the user that has created it. In most cases, DOSPLUS will never "know" what sort of data is contained in a file. The one exception to this is the Z-80 object code file. If you attempt to execute a file from the DOS command mode directly and it is not such a file, you will be receive an error message. The topic of "load file format" will be covered in the LOAD command. Consult that for further details.

As we have already stated, all files have a file specification (or filespec). This filespec may consist of from 1 to 4 parts. For instance, given the example:

PRICE/DAT.DOLLAR:1

This filespec has all four parts. The first part is the filename. In our example, this would be "PRICE". This filename may be from 1 to 8 characters in length and may contain any of the legal characters already given. Some examples are:

Legal	Illegal	Reason
MONEY	?MONEY	"?" is an illegal character
JUNSALES	JUNESALES	Too many characters

The second part of the filespec is the extension. In our example, this would be "/DAT". Note that the extension is separated from the filename by a slash mark. This is not optional! An extension may be from 1 to 3 to three characers in length and may use any of the legal filespec characters already given. The extension is a useful item that is usually implemented in indicating what sort of information is being stored in that file. The following are some examples of the extensions we have used in DOSPLUS:

ASM	Assembly language source file
BAS	BASIC language program file
CIM	A "core image" file. This file consists of data transferred directly from memory to disk. Not necessarily executable code.
CMD	Executable Z-80 object code. Usually called a "command" file.
DAT	A data file (of any type)
DVR	A driver file. This file is a peripheral "driver" that allows you to operate various types of hardware with DOSPLUS. This will be installed by the ASSIGN command.
FLT	A filter file. Contains the data needed to instruct the FILTER command regarding manipulating character I/O to the various devices.
PAT	A patch file. This file will contain the information needed to instruct the PATCH program in how to modify a file.
PDS	A Paritioned Data Set file. Also called a FILEDISK. See FILE driver in the drivers and filters section of the manual.
SYS	A system file. This file is actually part of the DOSPLUS Disk Operating System.
TXT	Any ASCII text file. This is also the extension used by the DO command as its default extension.

Extensions are certainly optional and you may whatever you wish, but these are the conventions that we suggest for the most commonly occurring types of files and in many cases are the default extensions that our commands and utilities will assume if no other extension is given.

While the extension is not a required part of the filespec, it is often used to describe more completely a file's contents. For example, we may have a number of files with the same filename, but differing in the extension:

Filespec	Contents
SALES/JAN	January sales
SALES/FEB	February sales
SALES/MAR	March sales
SALES/APR	April sales
SALES/QTD	Quarterly sales

The one exception to the optional nature of extensions is the Z-80 object file. In order for you to be able to execute a machine language program directly from the DOS command mode, it must have an extension. DOSPLUS assumes the extension "/CMD" for the name of any program entered from the DOS command mode. Therefore, if the name of your program is "TEST" and you type TEST and press ENTER from the DOS command mode, DOSPLUS will append a "/CMD" extension to that and seek to execute the file "TEST/CMD". This, of course, will not be found. Therefore, as you can see, DOSPLUS requires that you have some form of extension in order to execute a machine language file directly from the DOS command mode. It is true that you can use the LOAD command to do it or that you could assign a different extension such as "/Z80", but in each instance, executing the file becomes a tedious process. We strongly suggest that you adopt the habit of referring to your machine language files with the "/CMD" extension.

In our example, PRICE/DAT.DOLLAR:1, the third part of the filespec is the password. In this case, it is ".PASSWORD". A password can be given to any file in order to control access to it. You may, by using the file password in conjunction with the ATTRIB command's protection levels, assign a file any level of protection ranging from "full access" to "execute only". You may require other users to know the password before they can access the file at all, or you may require it only if they intend to modify the file. If you set up a file as "run only", they may run it without knowing the password, but will need the password in order to load, list, or modify the file in any way. For more information on defining a file as "run only", see the library command ATTRIB

A password may be from 1 to 8 characters in length and can consist of any of the legal filespec characters. The password is denoted by the period "." and is an optional portion of the filespec.

Once you have created a file with a password, be sure to remember what the password is. If you forget it, you will not be able to access that file again except through the use of the PROT command, and even then only if you know the Disk Master Password.

The fourth element of the filespec is the drive specification (or drivespec for short). In our example, this was ":1". This drivespec simply informs DOSPLUS that the file "PRICE/DAT" that we are referring to resides on the drive currently named ":1". We will cover drivespecs in detail later in this section.

For our purposes now, let it suffice to say that a drivespec is a one or two character name that indicates which of the drives that we are referring to. DOSPLUS was supplied with the drivespecs ":0" through ":7" as standard. You may assign these to the actual physical drives as needed and rename them as desired.

The drivespec is also an optional portion of the filespec. If you do not give the drivespec, DOSPLUS will begin with the first drive and search through all the drives currently defined in the system. This is called a "global search". It will continue until a matching filespec has been located or DOSPLUS has searched all available disk drives.

What makes a filespec unique

It is important that you understand clearly what portions of the filespec contribute to the uniqueness of the filespec. If, for instance, you have written a BASIC program and you wish to store it on the disk, you must assign it a filespec. It is important to remember that the filespec we assign does not duplicate an existing filespec, because if it did, DOSPLUS would overlay the BASIC program on top of the old file and whatever data was contained within it would be lost.

Three of the four parts of a filespec contribute to its uniqueness: the filename, the extension, and the drivespec. The password does not. What this means is that if two filespecs have the same filename and drivespecs, but a different extension, they are two distinct files. If however, two files have the same filespec, extension, and drivespec, but only different passwords THEY DENOTE THE SAME FILE! Some examples:

Filespec 1	Filespec 2	Same?
TEST/DAT.CLOUD:1	TEST/DAT.CLOUD:2	No
DATA/ONE	DATA/TWO	No
LEDGER/BAS.CASH	LEDGER/BAS.CREDIT	Yes
PAYROLL/BAS:0	PAYROLL/BAS	Yes
ALPHA/ASM	ALPHA2/ASM	No

If you bear this in mind as you are saving programs and opening data files, you can save yourself a great deal of potential problems. In this case, an ounce of caution is truly worth a pound of recovering lost data because of carelessly overwriting a previous file. To recap an important point, remember that in filespecs the filename field is mandatory in all cases. All other fields are optional.

Device and drive specifications

The DOSPLUS system has eight character devices and sixteen drive devices built into it. Following is a list of the two and whether it is an input or an output device:

Device	Default name	Class
Keyboard	KI	Input
Display	DO	Output
Printer	PR	Output
Serial port	RS	Input or output
Standard input	SI	Input
Standard output	SO	Output
User defined	U1	User defined
User defined	U2	User defined

Drives :

1st drive	0	Input or output
2nd drive	1	Input or output
3rd drive	2	Input or output
4th drive	3	Input or output
5th drive	4	Input or output
6th drive	5	Input or output
7th drive	6	Input or output
8th drive	7	Input or output
9th drive	8	Input or output
10th drive	9	Input or Output
11th drive	10	Input or Output
12th drive	11	Input or Output
13th drive	12	Input or Output
14th drive	13	Input or Output
15th drive	14	Input or Output
16th drive	15	Input or Output

A device name is a two character description assigned to that device. Whenever you access that device, you must specify the device name.

The first group, system devices, are all character orientated, which means that all I/O done to these devices is done byte by byte, one character at a time. The second group, drive devices, are what we call file orientated, which means they are used to move a file at a time.

This is not to say that a file itself cannot function as a character orientated I/O path; it can. These (files) are special cases and the file is functioning as a "channel". But a drive cannot function in this manner. Therefore the last eight devices, the drive devices, will address one file at a time, not one byte.

You may name your devices anything you wish. For the sake of conformity and standardization, we recommend that you leave the default names in effect. Within the manual, we will refer to them by their default names. To rename a device, use RENAME (see the library command RENAME). Do NOT confuse renaming a drive with re-routing the order in which the drives are searched. That is accomplished by using CONFIG (see the library command CONFIG) to alter the physical drive number for that drive device.

Some restrictions -

You may not assign two devices the same name. In order to swap two device names, you would have to temporarily rename one of the devices to a "dummy" device name.

Addressing devices

You address a system (character orientated) device via its device specification (DEVICESPEC for short). You will address a drive (file orientated) device via its drive specification (DRIVESPEC for short). A drivespec or devicespec will have two parts:

(1) The type indicator.

(2) The device name.

The type indicator is a single character that indicates whether we are giving a devicespec or a drivespec. It will be very important throughout the system to keep the two clearly separate. The type indicator for a devicespec is "@" (i.e. @KI is the keyboard). For a drivespec, this is ":" (i.e. :0 is the first drive).

The device name is any two non-reserved characters used to specify which device you are talking about. Remember, no two device names may be the same, unless if the devices are of different types (character/file).

Any time you refer to a device, no matter what sort of operation you are performing, you will use the devicespec. It is very important that if you decide to rename devices, remember what names you have assigned to what devices. To receive a list of the current device names and status, use the FORCE or JOIN commands' display ability (see the library commands FORCE and JOIN).

In most cases in DOSPLUS, you may use character orientated devices in place of filespecs. This is part of what is called "device independence".

Summary of device handling in DOSPLUS

The principle of device handling in DOSPLUS is really simple. There are only two ways that data gets from point A to point B within the system:

(1) A byte at a time (character I/O).

(2) A file at a time (file I/O).

The two of them are not the same, and as long as you remember that, you shall have no problems with specifying an illegal I/O path for the data to move on.

When specifying the I/O path, you can specify one of three things:

(1) A devicespec.

(2) A filespec.

(3) A drivespec.

Options one and two can operate in a character I/O mode. Options two and three can operate in a file I/O mode. Thus, the filespec is unique in that a file can work with both styles of I/O.

If all this device handling seems foreign and confusing, do not be concerned. The actual operation of the system is much simpler than the theories behind it. They are, however, what makes DOSPLUS work the way that it does and they deserve to be documented. As a user of DOSPLUS, you need only be concerned with "How does this command work and what can I do with it?". This is all explained clearly, command by command, in the library section of this manual. Those people who are DEVELOPING software using DOSPLUS will be able to make full use of the system's flexibility to develop new and innovative methods of performing the various tasks that make up a "program".

The next subject we will address is the explanation of the various parts of the command line and I/O field. In that discussion, we will examine how the system views a command line after it is entered, even to the point of taking a sample command line and proceeding step by step through it, detailing how the DOS will react to each portion.

Note: Before you can enter DOS commands, you must be at the "DOS command mode". To get there from MEDIC, which is where you still should be if you have followed all of the instructions so far, press the BREAK key twice. You will receive a blinking cursor and a "DOS PLUS" prompt, signalling the DOS command mode.

Detailed explanation of the command line

The command line is the means by which you communicate with DOSPLUS. When you are at the DOS command mode, you may enter up to 79 characters of text that commands DOSPLUS to do something. This line of text is called the command line and has four parts: (1) The command, (2) The I/O field, (3) The parameter field, and (4) The optional comment field. Let's look at each of these in turn.

The command. This is the actual DOSPLUS library command. This will call in the portion of the system that you wish to operate with. This command must be the first data on the line (although leading spaces will be ignored) and must be followed with either a terminator or a separator. A terminator is a carriage return, placed into the command line by pressing ENTER after typing in the command. You have "terminated" that entry. An example of this would be if you typed "LIB" and pressed ENTER. A separator, on the other hand, occurs when you follow the command name with a space prior to entering further data.

The I/O field. This is the field immediately following the command. It will specify the direction of the I/O and which files and/or devices shall be affected. The I/O field has three parts to it: (1) The source field, (2) The destination field, and (3) The wildmask field. These are indicated by the delimeter words FROM, TO, and USING respectivley. Each of these portions of the I/O field must be separated from their delimeters and each other by a space. You may omit the delimeter words if you wish, but if you desire to change the order of the various portions of the I/O field, you MUST include them. For example:

```
COPY FROM TEST/CMD:0 TO TEST1/CMD:1
```

is the same as :

```
COPY TEST/CMD:0 TEST1/CMD:1
```

But if you wanted to specify the destination file FIRST, you would have to use the delimeter words. Therefore:

```
COPY TO TEST1/CMD:1 FROM TEST/CMD:0
```

is NOT the same thing as:

```
COPY TEST1/CMD:1 TEST/CMD:0
```

Note: In DOSPLUS IV, the delimiter TO may be abbreviated with the greater than symbol (>) and the delimiter FROM may be abbreviated with the less than (<) symbol. You, therefore, may not use these in a filespec.

The wildmask field is a field that contains a filespec that has wildcard characters in it. This field is used to make the effect of a command global to several or all files. There are three wildcard characters: "?", "*", and "!". A question mark indicates that the specific character at that position is not important. An asterisk terminates that portion of the wildmask field and fills the rest of the characters with question marks.

For example:

T??T/B??

will match the files "TEST/BAS" and "TOOT/BOB" equally well. In the filename area, we used the question marks to skip two characters and then specified another character.

However, after the "B" in the extension area, we were through but wanted any and all extensions to match. In that case, we could have used the asterisk. For example:

T??T/B*

will match the same files as the previous example. The asterisk in the extension field fills the rest of the extension area with question marks. Taking it further:

T/BAS

will only match the file "T/BAS". However :

T*/BAS

will match ANY file that has a filename beginning with the letter "T" and ending with the extension "/BAS". If you do not wish to specify a filename, simply put an asterisk in the filename area. The same is also true for the extension. That will fill either area entirely with question marks and any character will match. The exclamation mark is used to indicate that BOTH fields should be filled with question marks past the point at which this character occurs in the command field so that ANY character will match. This is also used when it is necessary to perform a function, such as COPY, on all files on a drive. It saves keystrokes and is more convenient. For example:

T!

is the same as:

T*/*

because "!" is the same as "**". If you wish to use this character to replace the entire wildmask field (such as on a COPY), you would enter:

COPY !:0 :1

This tells DOSPLUS that you wish to copy ALL files from the disk in drive "0" to the disk in drive "1". This is very useful. DOSPLUS is signalled that a wildmask is present whenever: (1) the USING delimiter precedes the wildmask, (2) the wildmask appears in its proper area of the command line, or (3) the wildmask contains wildcard characters.

The parameter field. This field allows you to specify certain additional switches and values that modify the action of the command. This field need not be included at all unless you either want to use something other than the default parameters or you plan to include a comment field. The parameter field is set off from the I/O field by one of two things: (1) A comma, or (2) A left parenthesis. Within the parameter field, you must separate your parameters from each other with a separator. In the I/O field, you had to use a space as a separator because a comma would indicate the start of the parameter field. If you are using a comment field, you must conclude your parameter field with a right parenthesis, otherwise the line terminator described before will suffice. If for some reason you intended to use the comment field but had NOT included an I/O field, you would still have to place a right parenthesis in the command line prior to the start of the comment field to signal DOSPLUS that the following text was a comment and not part of the command line.

Within the parameter field, you will be entering parameters followed by expressions. These expressions will indicate what action the parameter will take in relation to the command. An expression will be one of three things:

(1) A string. This is in the case of a password or a disk name or any other input that requires you to enter a literal string for system use. These MUST be encased in quotes (single or double).

(2) A value. This is used to pass numeric data to the command about the parameter. An example of this would be setting the buffer size for the print spooler. You would specify a value at that point. Values may be expressed hexadecimal as long as you follow the value with the correct base specifier. You do NOT have to enclose a value in quotes.

(3) A switch. This is used to specify a positive or negative condition for a parameter. If you are turning something "on" or "off", you will use a switch. When using a switch, the terms "yes" and "on" are equivalent as are the terms "no" and "off". "Yes" and "No" may be abbreviated as "Y" and "N". You will not have to enclose a switch in quotes.

Remember, when you specifying parameters and expressions, you will always separate the expression from the parameter with the equals sign ("=").

The comment field. This field allows you to place an optional comment at the end of an executable command line. This is useful when using BUILD and DO for command chaining, because it allows you to document the command being executed. For example, a line could say "CREATE TEST/DAT (LRL=4) - Create index file", in order to let the user know what the command was doing (see also the library command CREATE). For further information and some practical examples of using the comment field, consult the library commands BUILD and DO.

Let's take an example and see how the command interpreter will view a command line. Given the command:

```
DIR :0 TO @PR (ALPHA) - Prints alphabetized directory
```

DOSPLUS will scan the command line from left to right. When you scan the command line and interpret what is there, you are said to "parse" the command line. Notice that the syntax for this command is correct. The I/O field is separated from the command by a space. The various parts of the I/O field have spaces between them. The parameter field begins with a left parenthesis. The comment field follows a right parenthesis, indicating a completed parameter field.

DOSPLUS will pick up the command "DIR". That tells it that we will be doing a directory. Since the first characters in the I/O field are not a delimiter word (FROM, TO, or USING), the system will assume that we are using the default sequence and pick up ":0" as the source field. It finds the delimiter "TO" and therefore knows that "@PR" is the destination field. In this case, the destination field was in the default position and the delimiter word TO was not needed. However, by saying "TO @PR", we free ourselves from the default positions. That phrase can occur anywhere in the command line and if the delimiter is present, it will be parsed as the destination field.

Next, DOSPLUS finds a left parenthesis. This tells it that the I/O field is complete and we are beginning the parameter field. To the right of the parenthesis, DOSPLUS finds the parameter "ALPHA", indicating that we desire the directory listed alphabetically. The next item found as DOSPLUS parses the command line is the right parenthesis. This tells the system that the parameter field is through and that anything that follows that parenthesis is a comment and should be ignored.

Definition of terms

The following is a list of DOSPLUS terms and their definitions. It is not meant to be a system glossary, merely to cover some often used technical expressions. Before these terms can be understood, novice users may find it necessary to read the preceding text on files and devices. More experienced users and programmers will find this a good "quick reference section" for terminology.

Term	Definition
Filespec	A reference to a particular disk file. This may not contain any wildcard characters, but can contain an optional drive specifier. A more detailed breakdown is afforded above.
Drivespec	A colon ":" followed by a one or two character drive name. Used to refer to a particular disk drive. May only be used when file I/O is specified. It is NOT a character orientated device.
Devicespec	An at sign (@) followed by a one or two character device name. Used to refer to one of the eight system devices. May only be used when character I/O is specified. It can be specified when an I/O channel is requested.

Definition of terms (cont)

Term	Definition
Channel	A channel is a character orientated I/O path. When a channel is requested, it is indicative of the fact that the data will be moved a byte at a time. File by file I/O is not allowed with channels. A channel may be either a filespec or a devicespec. It may NOT be a drivespec except in cases where a drivespec is only part of a filespec.
Wildmask	A filespec containing wildcard characters. Used to make the effect of a command global to several files. May not be used when a channel is requested. Consists of a filename and extension only. It can be used in conjunction with a channel, but cannot be specifed AS the channel. For further details on the use of wildmasks, see the section above - "Detailed explanation of the command line".
Parameter	An optional control field that can specify additional information on exactly HOW you want the indicated command to function. Can be a switch (On or Off), a string (passwords, etc.), or a value (buffer size, record length, number of lines per page, etc.) If the parameter is a switch, usually the mere mention of the parameter will engage it (i.e. "=Y" will be assumed).
Separator	Used to separate delimiters and channels, parameters, etc. Within the I/O field, separators MUST be a space. Within the parameter field, they must be a comma. If you use commas within the I/O field, DOSPLUS will terminate the I/O field and start looking for parameters. Separators are NOT optional. For the command line to be evaluated properly, you must separate the various portions of the fields.
Delimiter	A field specifier. Will be either FROM (or <), TO (or >), or USING. Indicates direction within the I/O field. These may not be used as filenames (i.e. you can't call a file TO/CMD, because "TO" is a reserved word). Remember, these must be surrounded by separators. You need not actually mention these terms in the command line unless you wish to specify the various portions of the I/O field in something other that the default order (e.g. specify the destination channel before the source, etc.). If the delimiter is present, it will override any default positioning and re-route I/O any way you wish.

Throughout the manual, we will be referring to these terms. Realizing that some of them may be un-familiar to you, we suggest that you review the above section carefully if you run across terms that you do not understand.

In the next section of the Operations manual, we will discuss the subject of Piping and filtering.

PIPES and FILTERS

The DOSPLUS IV piping feature allows the screen output from one program to be used as the keyboard input of another. Piping is a novel and powerful feature of the DOSPLUS system. The term "pipe" is actually a verb used loosely to describe what happens to the data (e.g. it is "piped" from one program to the other). A pipe is signalled by the vertical bar (|) in the command line (You obtain a vertical bar by pressing CTL-;. In other words, hold down the control key and press the semi colon). Normally, you may append two commands together by using the semi colon (;) to separate them. A pipe works in much the same manner, with the exception that it does some automatic I/O routing for you.

Pipes introduce two new devices: @SI and @SO. @SI is the <s>tandard <i>nput device and @SO is the <s>tandard <o>utput device. When a pipe is in effect, all display output from the first program in the "pipeline" (i.e. the command line where piping is in effect) will be sent to the @SO device. When this is finished, the second program in the pipeline will receive all of its keyboard data from the @SI device. Thus, whatever data program one outputs to the screen will become the input data for program two.

Normally, the two new devices are not connected to anything, so a pipe does nothing. You must first do a little set up work. In order for a pipe to be effective, the output data from program one in the pipeline must be stored somewhere until program two is operating and ready for it. The obvious choice is a file. We accomplish this by using the command FORCE (see FORCE) or the command ROUTE (see ROUTE) to re-direct all the output to the @SO device into a disk file. We suggest the filename "%PIPE1/$$$" so that you do not confuse this file with anything else.

That will take care of storing the output. But what do we do with the input for the next program? In that case, we must route the @SI device (which is replacing the keyboard input of the second program) also to a disk file. It would seem obvious that we need to tie it to the same disk file that we sent the output to earlier, but it is not that simple. It is possible for both elements of a pipe to be running at the same time (e.g. as program one is outputting data, program two is inputting it). This will occurr if you set up multiple pipes such that data flows from one program to the next and from there to yet another program. Therefore, special provision must be made. This is part of the DOS' piping function.

ROUTE or FORCE the @SI device to a second file. We recommend that you use the filename "%PIPE2/$$$", again, so that you do not confuse it with anything else. Now, if program one is outputting to @SO and that is routed to %PIPE1/$$$, and program to is inputting from @SI and that file is routed to %PIPE2/$$$, how does the data get from one file to the other? The DOS will do it all automatically. You do not even need to concern yourself with it. The operating system will see that the data gets from the ouput file to the input file when a pipe is in effect.

We mentioned a term earlier and have not yet discussed it. The term is "filter". DOSPLUS already has one form of filter, the translation filter for character I/O devices. These are valid and totally separate from what it is that we are discussing now. The FILTER command in the library (see FILTER) is designed to translate one character values as they move to and from the various character I/O devices (i.e. the printer, the display, the RS232, etc.) in the system.

But when talking in terms of piping, a filter is a program that is designed to get its input from the standard input device (normally @KI, but when a pipe is in effect, @SI) and output to the standard output device (normally @DO, but when a pipe is in effect, @SO). Thus, a filter is a program that receives the data from the pipe and in some way acts upon it (called "filtering" the data, hence the name "filter"), and then outputs this data to either the display or into another pipe unless you override this with specific operators (depending on the filter).

Does that mean that we can have more than one piping operation in the same command line? Most certainly. You may pipe data from one library command into as many filters as you can fit on the command line. Each filter will send its data to the next filter and they will all work together. At the end of the pipeline, the final filter program will send the result to the standard output device for that filter. Let's examine this from the practical standpoint. The mechanics of using pipes are simple. Before using a pipeline, we need two commands. They can even be placed on one line ich as this:

```
FORCE @SO %PIPE1/$$$;FORCE @SI %PIPE2/$$$
```

This sets up the temporary file areas for the pipes to store their data. Then, let us suppose that we desire to send the output of the FREE command (see FREE) to the filter "MORE". MORE is an example of a filter program that has been included with DOSPLUS so that you can see a possible application for this. The MORE filter will read data from the standard input device (@KI or @SI, depending on whether or not a pipeline is established) and output this to the standard output device (@DO or @SO, depending again on whether or not a pipeline is established). When MORE has output what it feels to be 22 lines of data, it will pause and place the prompt "---MORE---" on the screen. You will have to press ENTER to get the output to continue. So, if we desire to send the output of FREE to the filter program MORE, we need only enter the command:

```
FREE :5|MORE
```

This will display a free space map from drive ":5" (for this example, we are assuming a hard disk with a LARGE map). Because the pipeline is established, this output will go to the @SO device which we previously routed to a disk file. Once this is complete, the filter program MORE loads and begins requesting data from the standard input device. But since the DOS has seen to the transfer of data, MORE will actually read what was piped to the @SO device.

It will display the free space map and when the screen is full, it will pause and wait for ENTER to be pressed before continuing. But this is simply one sample filter program to give you an example of what is possible. The total applications for this procedure have no limit. Admittedly, the novice user will not be able to create these filters by themselves, but over a period of time such programs should become available from other sources.

Another example might be listing a file through the MORE filter using a pipeline. The command to list, say, a BASIC program saved in ASCII could be:

```
LIST TEST/BAS|MORE
```

This would display all the data in the file, one screenful at a time. All with the inclusion of only one extra character (the pipe - |). For still another example, using the conditions we established above (e.g. the routing of the @SI and @SO devices to the files), we want to send the output of the free space map to the line printer via the fictional filter program called CASE (that will have the assumed function of converting all lower case and graphics characters to something that the printer will handle correctly). To do that, we would use:

```
FREE :5|CASE TO @PR
```

This re-directs the output of the CASE filter program to the line printer. However, you may also send the output of the CASE filter further into the pipeline. Let's assume that you have a second filter program called ORDER, that places all input in alphabetical order. You might then use the command:

```
FREE :5|CASE|ORDER
```

The output of the FREE command gets piped to the input of the CASE filter program. The output of this program becomes the input of the ORDER filter program. This "pipeline" could go on for the entire command line. As long as the @SI and @SO devices are routed to the disk files, you may continue to implement pipes. The data will automatically flow from one program to the next with no additional user input.

You must be careful not to modify in any way (or kill, either) the temporary pipe files. These must be left for the system use only. They do not contain data in the standard sense. When you wish to remove them and disable the piping, use the commands (usually on the same line):

```
RESET @SI;RESET @SO
```

This will close the temporary files and at that point you may delete them. After you have performed this RESET, do not attempt any further piping.

So, in conclusion, the bulk of the work in the pipeline falls on the filter programs. These programs must be set to accept data from the standard input device and send data to the standard output device. Optionally, they may accept device re-routing in the command line, that is up to each individual filter program. As stated earlier, the only filter program supplied with DOSPLUS is MORE, but with the passing of time, we hope that more and more filter programs become available. Pipes and pipelines are simply conditions in which the operating system performs a certain amount of device re-routing for the user. In order to use the pipes effectively, you must route the @SI and @SO devices to disk files. To allow multiple pipes, we suggest that you use two different disk files as stated above.

Filters are simply programs that will accept data from the standard input device (which is either the keyboard or the pipeline, if the pipeline is in effect), modify or "filter" the data in some way, and output this data to the standard output device (which is whatever the filter was written to use or the pipeline, if yet another pipe is specified).

Piping and filtering provide a powerful addition to DOSPLUS in which the user, wit a minimum of effort, may pass the output data of one program to another program as input data. This allows the creation of special purpose filter programs that in some way manipulate this data flow. It is simply one more example of DOSPLUS' powerful device independence.

Summary

Remember, this is a REFERENCE manual, not a TUTORIAL. It is not written as a book would be from start to finish. Use the index and the table of contents for the various sections to locate the commands that you need.

It is our hope that you find DOSPLUS IV useful and valuable to you. We think that you'll enjoy the system.

DOSPLUS IV Library of commands

The following are the library commands for DOSPLUS IV. To execute a command, enter the name of the command followed by any needed parameters:

Command	Description	Page #
APPEND	(Append two devices or files together)	2-2
ASSIGN	(Install a driver routine)	2-6
ATTRIB	(Alter file's attributes)	2-10
AUTO	(Set auto execute command)	2-15
BOOT	(Execute system "cold-start")	2-19
BREAK	(Disable/Enable BREAK key)	2-20
BUILD	(Create ASCII text file)	2-21
CAT	(Display drive's file catalog)	2-24
CLEAR	(Clear user memory and files)	2-29
CLOCK	(Turn on/off system clock display)	2-32
CLS	(Clear screen)	2-33
CONFIG	(Alter system configuration)	2-34
COPY	(Copy device/file to device/file)	2-48
CREATE	(Create and pre-allocate disk file)	2-55
DATE	(Display or change system date)	2-60
DEBUG	(Activate system memory monitor)	2-61
DIR	(Display detailed file listing)	2-64
DO	(Execute command chain file)	2-73
DUMP	(Save memory to disk file)	2-77
ERROR	(Display detailed error message)	2-80
FILTER	(Filter I/O to/from specified device)	2-81
FORCE	(Re-direct I/O to device/file)	2-85
FORMS	(Alter printer pagination parameters)	2-87
FREE	(Display free space data)	2-92
I	(Initialize disk drive)	2-95
JOIN	(Link two logical devices)	2-97
KILL	(Kill specified device or file)	2-100
LIB	(Display list of library commands)	2-104
LINK	(Link two logical devices)	2-105
LIST	(List file to device)	2-108
LOAD	(Load disk file into memory)	2-110
PAUSE	(Pause execution)	2-112
PROT	(Alter disk's protection status)	2-113
REMOVE	(Kill specified device or file)	2-116
RENAME	(Rename a device or file)	2-120
RESET	(Restore device to default driver)	2-121
ROUTE	(Re-direct I/O to device/file)	2-122
RS232	(Display/alter serial port settings)	2-124
SCREEN	(Send contents of screen to device)	2-128
SYSTEM	(Customize your operating system)	2-129
TIME	(Display time or set system clock)	2-137
VERIFY	(Toggle automatic disk verification)	2-138

APPEND

This command allows you to append one device or file to another device or file.

APPEND [FROM] device1/file1 [TO] device2/file2 (param=switch...)

device1/file1 is the SOURCE device or file. This is the device or file that will be appended.

device2/file2 is the DESTINATION device or file. This is the device or file to which you will be appending.

(param=switch...) is the optional action switch.

The parameters are:

CMD=switch	Appends to destination file in load module format (i.e. a /CMD file).
STRIP=switch	Backspaces one byte from the end of file on the file being appended to.

Abbreviations:

CMD	C
STRIP	S

The APPEND command may be used as a means of easily combining two data files. By using APPEND, you avoid having to open both files, position to the end of the destination file, read from the source, write to the destination, etc., etc.

You MUST append from an input device or a disk file (source) to an output device or disk file (destination). A list of default devices and their names and classes is available in the DOS overview section of this manual. A disk file may function as either input or output. APPEND will never affect the source device.

APPEND can also be used as a sort of dynamic disk merge with BASIC program files. You may append one BASIC program (saved in ASCII) on to the end of another BASIC program (also saved in ASCII) and then load the resulting file. The lines appended will overlay any lines in the original file and the program may then be saved back to the disk under whatever filename you choose in compressed format, if you desire.

Generally, APPEND is used to join two files together, however, it can also function with devices. There are two special cases in disk files that APPEND must compensate for: files with special "end of file" markers and machine language programs. These are the reasons for the special parameters.

STRIP. Some data files may also have an "end-of-file" marker. Most data files will not, they let their end-of-file be maintained by the DOS and the directory points to the end-of-file in those cases. This is the case with both data files created by BASIC and with BASIC programs themselves. However, certain programs create data files that use a one-byte value to signal the end of the file (an example would be data files from some word processors which use a 00 byte to signal the end of file). In those cases, when you append another data file onto the end of the first, the end-of-file marker would inhibit the program from using it. Therefore, to get around this, DOSPLUS' APPEND command has a STRIP parameter. When you specify strip, it will overlay the last byte in the file being appended to with the first byte of the file being appended, thereby stripping the end-of-file marker.

CMD. APPEND also has an optional switch to append to the destination file in load module format (i.e. the CMD switch). Load module format is simply the format used to store a machine language program on the disk so that it can be correctly loaded at exact locations later. Simply stated, a file in load module format contains "block markers" that inform the DOS' program loader what sort of information is here and where it should be loaded in memory. When loading a data file, DOSPLUS does not "look at" the file, it merely loads the data that it finds into the locations specified. However, when loading a machine language program (load module format), DOSPLUS actually scans the file to find out where it wants to load. Because the block markers identify comments and the like, these will be skipped during the program load.

Also, the last four bytes in any machine language program's disk file is called the "transfer address". These bytes tell the DOS where to begin executing the program it has just loaded. When the transfer address is encountered, execution begins immediately. Therefore, you could not effectively append two machine language programs together if the second never got loaded because the first was immediately executed. To avoid these problems, when you append in load module format, the last four bytes of the file being appended to (that file's transfer address) will be overlaid by the first four bytes of the file being appended. When the DOS encounters no transfer address, the file will continue to be loaded, any duplicate addresses will be overlaid with the instructions from the second file, and the transfer address of the appended module will be used.

Unless you specify the CMD option, APPEND will always assume you wish to save the appendage in data file format. Machine language programs MUST be appended with the CMD option.

When you append a device to a disk file, all data coming from the device is sent to the disk file instead of its normal destination. If you append a disk file to a device, all data contained within the file will be sent to the device. If you append two devices together, all data from the one device will be sent to the other.

Appending a device to a file is essentially the same thing as copying that device to the file (see COPY), except that if you append a device to a file it will position to the end of the file after opening it instead of over-writing.

PLEASE use extreme caution when appending devices. As with any system this flexible, it can be mis-used and "hang-up" the system. Think through your logic carefully when appending devices. Always bear in mind that you must append from an input device and to an output device. If you are not certain what a particular device is, use the FORCE or JOIN command's display (see FORCE or JOIN) to distinguish.

Examples:

```
APPEND FROM DATAFIL1 TO DATAFIL2
APPEND DATAFIL1 DATAFIL2
```

This command will take all the data in DATAFIL1 and append it to the end of DATAFIL2.

```
APPEND NEWMOD/BAS TO OLDPROG/BAS
APPEND NEWMOD/BAS OLDPROG/BAS
```

This command would append the file NEWMOD/BAS on to the end of the file OLDPROG/BAS. In the case of two BASIC programs saved in ASCII, when the file OLDPROG/BAS was loaded next, the lines in the appended module would overlay those in the initial module. For example, let's assume that the file OLDPROG/BAS contained the lines:

```
10 CLS : PRINT "This is the old program."
20 FOR I=1 TO 1000
30 NEXT I
```

And the file NEWMOD/BAS contained the line:

```
20 FOR I=1 TO 250
```

After you had saved both of these in ASCII and executed the above APPEND command, the next time that you loaded in the file OLDPROG/BAS, you would get the following:

```
10 CLS : PRINT "This is the old program."
20 FOR I=1 TO 250
30 NEXT I
```

As you can see, the line from NEWMOD/BAS has become part of the program OLDPROG/BAS. However, if you had listed the file from the disk first (see LIST), you would have seen:

```
10 CLS : PRINT "This is the old program."
20 FOR I=1 TO 1000
30 NEXT I
20 FOR I=1 TO 250
```

As you see here, there are TWO lines with the line number 20. The second will always overlay the first. After loading in the new program, you should save it out in its altered form.

```
APPEND PATCH/CMD PROGRAM/CMD (CMD)
APPEND PATCH/CMD PROGRAM/CMD,C
```

This command will take the load module format file PATCH/CMD and append it to the end of the load module format file PROGRAM/CMD. It will keep the appendage in load module format. When the file PROGRAM/CMD is executed from DOS, the instructions in the file PATCH/CMD will merge themselves in with the program and modify it. This is a VERY effective way of patching programs. Simply write the patch module and assemble it to load in at whatever address it needs to to modify the existing code and then append it to the end of the file to be patched.

```
APPEND @KI DOCUFILE/TXT:1
```

This command will append any further data that is input from the keyboard (i.e. any further keystrokes) on to the end of the file DOCUFILE/TXT that is located on drive one. This would allow you to append further instructions onto the end of a build file, for example (please note that BUILD itself has a superior manner of accomplishing this, though).

```
APPEND TO SERIAL/DAT:0 FROM @RS (STRIP)
APPEND TO SERIAL/DAT:0 FROM @RS,S
```

This command will open the file "SERIAL/DAT" on drive zero, position to the end of the file, backspace one byte to strip off any end-of-file marker that your last operation might have put there, and then append any further incoming data from the serial interface to the end of that file (this assumes that you have installed the serial drivers and activated the device).

The important thing to note here is that in this example the order within the I/O field was changed. Under normal circumstances, the I/O field specifies the source first and the the destination. By including the FROM and TO delimiters, however, you may override the default evaluation and route the I/O any way that you want. Remember, you must specify the delimiters FROM and TO if you wish to change the normal order within the I/O field.

ASSIGN

This command will allow you to install an alternate driver for a device or drive.

ASSIGN drivespec/devicespec filespec [param=exp...]
ASSIGN drivespec drivespec

> drivespec/devicespec is the name of the drive or device for which we are installing the driver.
>
> filespec is the name of the file that contains the driver. This filespec will be assumed to have the extension /DVR.
>
> param=exp... is the optional parameters for the driver. Whether or not these even exist is dependent upon the driver itself.

In the DOS overview section of the manual, we explain we devices and drives are and the difference between them. All devices and all drives require a program to handle the data transfer between the device/drive and the CPU. In effect, this program "drives" the device/drive. Hence, these are called "drivers".

The driver is that program that is actually responsible for the various features and items of performance that the device/drive has. For example, the computer cannot simply operate a hard disk without some form of driver for that drive. The computer has the spot to which to attach that drive and the DOS has floppy disk drivers built in to it. But these drivers lack the ability to communicate with the hard disk. So, without special hard disk drivers, you will not be able to use the hardware. Hence, the driver determines what the hardware (be it a device or a drive) can do.

You will also use the ASSIGN command to install special program for such devices as a spooler, a MEMDISK, or the MacroKEY program. These things are not normal functions of the operating system and require an additional driver. The actual command syntax of installing each driver will be covered with the driver itself. We are merely discussing the general function of the ASSIGN command.

ASSIGN does nothing magical of itself. It simply affords you a means of installing a special driver into the system. As time goes on, more of these drivers will become available for DOSPLUS to accomplish special tasks. By having a device structure such that drivers can be installed later (called an "external" device structure), DOSPLUS is an open-ended system. For as many different types of hardware or special applications as exist, there can exist a driver for each. There is no planned obsolesence with DOSPLUS.

The ASSIGN command merely installs the driver. What messages the driver displays as it loads, what functions it performs, and whether or not it may be de-installed once in effect is determined by the actual driver itself.

Note that the ASSIGN command has two forms shown in the command box. The first form is when we are installing the driver from the disk file into memory and activating it for that device. This is the form that will always be used the FIRST time a driver is loaded. The second form is for a special case involving hard disk drives.

Each drive position that will be using a hard disk must have a hard disk driver installed for it. However, each time that you load a hard disk driver into memory, it takes up space. If you are going to divide the hard disk into, say, six volumes; you would load the driver SIX times! We avoid this unwelcome situation by allowing you to have the same driver installed for two different disk drives. Once you install the hard disk driver for one of the positions using the first form of the command, you may assign all other hard disk drivespecs to that first drivespec. For example:

ASSIGN :4 RIGID

This command installs the hard disk driver RIGID/DVR for the drive named ":4". To assign the same driver for the drive named ":5", use the command:

ASSIGN :5 :4

This command installs the same driver currently in effect for drive ":4" on drive ":5". The two drives, in effect, become identical. You would then use the CONFIG command (see CONFIG) to direct each drivespec (e.g. ":4" and ":5") to point at the proper areas of the hard disk.

The important point is that the driver was only actually loaded into memory once! This will apply only to disk drives. You cannot assign a device to another device.

DOSPLUS has drivers for all of the four main system devices. The keyboard (@KI), the display (@DO), the printer (@PR), and the RS232 (@RS). These drivers are present upon powerup. There is no need to assign them. However, should you want use the MacroKEY function with the keyboard driver, you will have to install the MKEY/DVR program (see MKEY) onto the keyboard driver.

Drivers written for DOSPLUS should be "relocatable". Relocatable is a term that means the driver will load into whatever area of memory is free. Also, when a driver is intalled on a device by means of the ASSIGN command, whatever linking or routing is in effect for that device will be reset. However, any character filtration (not to be confused with filter programs in pipelines) will remain in effect.

Any drivers that have been assigned and are in effect when a system configuration file (see SYSTEM) is saved will be saved with the file and re-instated when the file is loaded again.

Parameter passing with the ASSIGN command

Certain special drivers may require a set of parameters to indicate how the driver should react or where it should load. These parameters must be passed at installation time. That is where the optional parameter field comes into play.

These parameters could even be an additional filespec as in the case of the MKEY driver which picks up the name of the MacroKEY text file from the command line when re-defining the keys.

To pass parameters to the driver, simply type a space after the name of the driver program in the command line and begin entering the parameters. Separate the parameters from themselves with a comma (,).

The parameter list must be terminated by either a carraige return (you pressign ENTER at the end of the line), an implied carraige return (a semi colon before the next command), or a right parenthesis ()). It will be up to the individual driver to retrieve the parameters and interpret them. The technical manual will have a description of what you can expect to find when ASSIGN is used in this manner.

Examples:

```
ASSIGN @PR SPOOL CHRS=5000
```

This command installs the driver from the disk file SPOOL/DVR for the printer (@PR) device. The parameter CHRS=5000 will be passed to the SPOOL/DVR program and it may do whatever is indicated by it.

```
ASSIGN :4 RIGID
```

This command will install the driver from the file RIGID/DVR for disk drive ":4". All I/O to or from that drive would be controlled by that driver.

```
ASSIGN :5 :4
```

This will duplicate the driver from drive ":4" for drive ":5". All configuration parameters will be identical.

```
ASSIGN @PR @PR
```

This is an invalid command. You may not assign a device to a device.

ATTRIB

This command allows you to set a file's user definable attributes.

==========

ATTRIB filespec (param=exp...)
ATTRIB [USING] wildmask (param=exp...)

filespec specifies the file that you wish to alter the attributes of.

wildmask is the wildmask that indicates which file or group of files we are operating upon.

(param=exp...) is the attribute we wish to alter and the new value we wish to assign to that attribute.

Your parameters are:

PW="string"	Disk Master Password. Required during wildmask ATTRIBs.
ACC="string"	New access password.
UPD="string"	New update password.
PROT=value	New protection level.
LRL=value	New Logical Record Length.
INV=switch	New invisible status.
KEEP=switch	New non-shrinkable status.
MOD=switch	New mod flag status.
CLOSE=switch	New file closed status.

Abbreviations:

ACC	A
UPD	U
PROT	P
LRL	L
INV	I
KEEP	K
MOD	M
CLOSE	C

==========

The ATTRIB command gives you total control of a disk file's attributes. You may use it to alter the amount of access you allow to a particular file, set or remove certain flags DOSPLUS maintains on a file, or change a file's password.

The ATTRIB command operates in two modes: standard and global. In the standard mode, you specify the filename after the ATTRIB command itself, including any needed extensions, drivespecs, or passwords. Following that is your parameter list of items to change. In the global mode, you specify the wildmask after the ATTRIB command, and follow that with the parameter list. When doing a global ATTRIB, you will have to specify the disk's Disk Master Password using the PW parameter unless the disk has no master password set for it.

If you are using ATTRIB on a file that already has a protection level and passwords defined for it, then you must specify the password in the filespec when invoking the ATTRIB command. For example, if we desire to make the program TEST/BAS invisible and it already has a password of FARKLE, then we must use the command:

```
ATTRIB TEST/BAS.FARKLE,I=Y
```

With the password in the file specification. This is opposed to the global attrib in which you must specify the Disk Master Password with the PW parameter. In that casem let's assume that you wish to globally alter the attributes of files located on drive ":0". The disk in that drive has a password of PASSWORD on it. Assume that we desire to reset the MOD and CLOSE flags for all files on that drive. The command would be:

```
ATTRIB !:0,PW="PASSWORD",MOD=N,CLOSE=N
```

Use of ATTRIB can be divided into two major areas. The first would be controlling a file's protection level (this includes the actual protection level and the passwords). The second area is controlling various flags and conditions regarding a file. We will cover each in turn.

Controlling a file's protection level

Before entering a discussion on altering a file's protection level, it will be wise to understand what a protection level is and how it works. A protection level is useless unless a password has been set for that file. You see, if no password has been set, then in effect "no password" IS the "password". Therefore, when the user omits the password, they have in actuality SPECIFIED the correct password and they are allowed full access to the file.

Each file has two passwords: access and update. The purpose behind this is to allow you to have to different levels of access for the same file. If a user knows the access password, then they have access to the file at whatever level you have set the protection level for. If a user knows the update password, then they have total 100% access to the file.

It is often judged convenient to set a program file with the access password set to nothing, a valid update password, and "execute only" (level 6) protection. This environment allows any user to execute (run) the program, but only those users in possession of the update password to examine or modify it in any way. This practice is not often used with data files, however. They usually bear no password or both passwords, since "execute only" really only applies to programs.

Also very important to know is that under DOSPLUS, the Disk Master Password may be used at any time in place of a file password. This means that knowledge of that password will let you into any file on the disk (excluding protection level 7 that is set by the DOS as "No access!"). Therefore, you should use care, when protecting files, to not only password protect the files but also the disk. You will be afforded the opportunity to set the Disk Master Password when you format each disk. The DOSPLUS Master diskette bears the password of PASSWORD. To alter an existing Disk Master Password, use the library command PROT (see the library command PROT).

So then, once you have decided which (or possibly both) passwords you will set, now you decide on the level of protection desired. You have several protection levels to choose from. You refer to them and set them by their numbers. This list will illustrate those that you have a choice of:

#	Protection level
0	No protection set. Total access.
1	Kill, Rename, Write, Read, Execute.
2	Rename, Write, Read, Execute.
3	Not used at this time.
4	Write, Read, Execute.
5	Read, Execute.
6	Execute only.
7	No access. Not a user option.

Protection level 1, total, allows you complete access to a file. You may kill it, rename it, write to it, read it without executing, or execute it.

Protection level 2, rename, allows you to do everything to a file EXCEPT kill it from the disk.

Protection level 3 is not implemented in this release of DOSPLUS, but ATTRIB will allow you to set this level.

Protection level 4, write, will allow you to write to a file or load it without executing, but you may NOT rename the file or kill it.

Protection level 5, read, will not allow you to write to the file at all, but will allow you to load it without executing or read it without loading. This would enable you to examine the code but not enable you to alter it.

Protection level 6, execute, will only allow you to execute that file. If it is a BASIC program, you may only RUN it. You may not load it or list it or interrupt program execution while it is operating. Machine language programs may be run but not examined or modified.

Protection level 7, no access, is not a user option. This cannot be set via the ATTRIB command and must be done manually at the system level. It is reserved for special cases and is only explained to avoid confusion if it occurrs.

To actually set these levels and passwords, use the parameters PROT, ACC, and UPD. Simply set the ACC and UPD parameters equal to a string that contains the desired password and set the PROT parameter equal to the number that reflects the desired protection level.

Again, remember that these protection levels work in conjunction with the ACCESS password. They need that password to get to the file at all and once they do, THEN the protection level restricts the amount of access. Anyone with the update password has complete freedom to update the file no matter what protection level has been set.

Manipulating file flags and conditions

The other main function of ATTRIB is to allow you to change certain status flags and conditions that DOSPLUS maintains about each file. These include the file's logical record length, whether the file is visible or invisible, whether the file's disk space can be dynamically altered, or whether or not the file has been written to since you last copied it off or backed up the disk.

LRL. Under DOSPLUS IV, the logical record length of a file is merely for your convenience when it comes to displaying that file from the directory. Both the DOS and BASIC allow you to open a file with a logical record length different than the one indicated when the file was opened. However, certain programs may require that the logical record length be correct and it is always convenient when working with variable length data files in BASIC to be able to see the logical record length. This option allows you to alter a file's logical record length.

To use it, specify LRL=value (where value is the desired logical record length). A logical record length can be anywhere between 1 and 256. You may use the wildmask option to alter the logical record lengths of a group of files or even an entire disk.

INV. When a file is invisible, it does not get displayed via a normal directory display. In order to see these files, you must specify the INVIS option from the DIR or CAT command (see the library commands DIR and CAT). This is very useful when a file is a permanent part of your working DOS system and you do not wish to see that filename constantly displayed when you list the disk's directory. This option affects only whether a file is visible. Simply because a file is invisible doesn't mean that it is protected. You must set the protection level independently.

To make a file invisible, specify INV=Y as the parameter for the ATTRIB command. To restore it to visible status, use INV=N. Remember, by utilizing the wildmask capacity of this command, you may make large classes of files visible and invisible.

KEEP. When a file has the KEEP option set, that tells DOSPLUS not to decrease the disk space for that file. Normally, when you create a data file on a disk and then access it later without filling up the file, the un-used space will be de-allocated (freed for other use). This can cause problems when you have pre-allocated space in a data file to prevent another program from using required disk space. This does not inhibit DOSPLUS from expanding the file, it merely prevents it from shrinking.

To set the KEEP option, attrib the file as KEEP=Y. To turn this off, attrib the file as KEEP=N.

MOD. This flag tells you when a file has been updated since you last copied it or backed up the disk that it resides on. Updating a file refers to writing to the file. If you simply read from a file, you have done nothing to alter that file, therefore the mod flag is not set.

By using the mod flag with COPY (see the library command COPY), you may copy off only those files needed when making backup copies of programs. For example, suppose you are developing a program. You wish to copy off all the files you worked on today. You merely copy any that have the mod flag set. The rest of them have not been overwritten since the last time you copied the file off or backed up the disk. The same principle will apply with data files.

This parameter may, from time to time, need to be set or reset manually. ATTRIB allows you to do that. An example might be a program that uses a general "system information" file and then several specific data files. You would not need to backup the general file at the end of each session, just the specific files. The easiest way to do this is to use the MOD flag when copying. However, the general purpose file was also modified each time the program was used, even just to read and write one record from it without changing it. You may then use ATTRIB to manually reset the MOD flag before using COPY. Simply attrib the file as MOD=N. To manually set the MOD flag, attrib the file as MOD=Y.

CLOSE. This flag is maintained in the directory by the operating system to indicate when a file has been opened and not closed properly. It is execellent computing practice to close all files when finished with them, and you should do so.

Machine language programs desiring to open for "read only" will have this option available at the system level.

Occasionally, a file may be left open due to error. This will appear as a question mark (?) in the attribute column of the directory display (see DIR). Although DOSPLUS in this release does not use this flag, future versions may and the ability to reset the flag is important. For now, you may wish to reset the flag for purely cosmetic reasons.

To reset the flag, attrib the file as CLOSE=N. You may also set this flag if you desire, but it is not implemented in this release of DOSPLUS.

Examples:

```
ATTRIB UTILITY/PRG:1 (UPD="PASSWORD",PROT=6,INV)
ATTRIB UTILITY/PRG:1 (U='PASSWORD',P=6,I)
ATTRIB UTILITY/PRG:1,U="PASSWORD",P=6,I
```

In this example, we are addressing the file UTILITY/PRG located on disk drive :1. We are setting the update password to PASSWORD, the protection level to 6 (execute only), and making the file invisible. Note that the access password was NOT set. This will allow you to run the program without knowing a password, but you may not modify examine the code without the update password. This all assumes no previous password.

```
ATTRIB FILE (MOD=N)
ATTRIB FILE (M=N)
ATTRIB FILE,M=N
```

All three of these commands will have the same effect. They will do a global search of all drives for the file named FILE. When they find it, they will reset (turn off) the mod flag. This is an example of manually resetting that flag.

```
ATTRIB PAYDATA:AA (KEEP=Y)
ATTRIB PAYDATA:AA (K)
ATTRIB PAYDATA:AA,K
```

All three of these commands will also have the same effect. In this example, we are operating on the file named PAYDATA currently located on the drive named :AA. We are setting (turning on) the KEEP flag to indicate that we do NOT want any of that file's disk space released to the system even if the file decreases in space actually used.

```
ATTRIB !:5 (LRL=256,PW="MARK")
ATTRIB !:5,LRL=256,PW="MARK"
ATTRIB !:5,L=256,P="MARK"
```

These commands would cause the system to alter the logical record lengths of all visible user files on drive 5 to 256. If the file is protected, the PW parameter will give access via the Disk Master Password.

```
ATTRIB TEST1.TEST:1 (CLOSE=N,INV=Y)
ATTRIB TEST1.TEST:1,CLOSE=N,INV=Y
ATTRIB TEST1.TEST:1,C=N,I
```

These commands would all cause the file TEST1 with the password TEST located on drive ":1" to have the CLOSE flag reset (if it set or not) and would make the file invisible (whether or not it is alredy that way).

```
ATTRIB FILE,I
```

This command will search all drives for the file FILE and make it invisible.

```
ATTRIB FILE/*,I
```

This command will search the system drive for all files with the name FILE, disregarding the extension, and make them invisible.

When you specify a single file ATTRIB without a drivespec, ATTRIB will search all available drives looking for the file. If you specify a wildmask ATTRIB without a drivespec, ATTRIB will only search the system drive, but it will find all the files that match the wildmask.

AUTO

This allows you to set a command to be executed upon boot-up of the system.

AUTO [drivespec] [param][command line]

drivespec is the optional drive specifier that tells DOSPLUS which disk you wish to store this AUTO command on.

param is the optional AUTO parameter. This allows you to effect what type of AUTO will be in effect. Do not separate the command line from the parameter, if a command line is included.

command line is the optional command line that you wish executed upon power-up.

Your parameters are:

!	Invisible AUTO. Do not display command line when executing.
*	Non-breakable AUTO. Do not allow the holding of the ENTER key as the system boots to prevent the AUTO execution.
?	Interrogate AUTO. Display whatever command line is set for the disk.

The AUTO command allows you to set a command line that will be automatically executed whenever the system is booted (unless the ENTER key is held down as the system is booting). This command line may be a library command, a program name, a configuration file, or any command that you might normally have to enter yourself.

There are several different types of AUTO. There is the standard AUTO, in which each command is displayed on the screen before it is executed. This form of AUTO may be defeated by holding down the ENTER key as the system is booted. You also have the invisible AUTO, in which the command is NOT echoed to the screen as you boot up. The final form would be the non-breakable AUTO. This is used when you do not want the user to have the option of escaping the AUTO by holding down the ENTER key as the system is booted.

To implement the alternate forms of AUTO, include the correct character in front of your AUTO command. The characters may appear in either order if you choose to use both of them. For the invisible AUTO, use an exclamation mark (i.e. !) and for the non-breakable AUTO, use the asterisk (i.e. *). Therefore, the command DIR would be:

AUTO DIR

with the invisible option (such that DIR would not display on the screen, but could be aborted by holding down ENTER):

AUTO !DIR

with the non-breakable option (such that DIR would display but could not be avoided):

AUTO *DIR

with both engaged (such that DIR would not be seen and could not be avoided):

AUTO !*DIR - or - AUTO *!DIR

As was explained earlier in the operations section, you may enter multiple commands on the same line as long as you separate these commands with a semi colon ";". This implies a carriage return and enters the command to that point. This means that you may actually have two or more commands imbedded in your AUTO statement as long as the TOTAL length of the command is under 31 characters.

If you use the multiple command feature (i.e. command;command), AUTO will write these commands to the disk exactly as you enter them, length permitting. For example:

AUTO LIB;FORMS

would write:

LIB;FORMS

to disk so that when you booted the system the commands LIB and FORMS would be executed. It will NOT write LIB to the disk and then execute a FORMS command. Remember, the total length of the AUTO command must not exceed 31 characters, anything longer will be truncated.

By using the optional drivespec, you may set an AUTO on a diskette other than the one that is in the system drive. This is useful in preparing program diskettes for use. you may set an AUTO on a disk without having to actually boot from that disk. When you wish to set an AUTO on a floppy disk and your system disk is other than a floppy, this can be an extremely important feature.

Also, you can use this same feature to reset an AUTO on a disk. For example, let's assume that we have a program disk of some sort that is set to directly execute from a non-breakable AUTO. We wish to reset this so that it doesn't AUTO directly into the program. All we have to do is place the disk in another drive other than the system drive and enter:

```
AUTO :ds
```

where ":ds" is the drivespec of the drive containing the disk we wish to operate upon. This will cause the AUTO command field on that disk to be reset. If you enter AUTO without the optional drivespec or command (e.g. AUTO by itself on the command line), it will reset the AUTO field on the current system drive.

If you wish to interrogate an AUTO (e.g. discover what AUTO command line, if any, as been set for a particular disk), use the question mark character (?). For example:

```
AUTO :1 ?
```

This command will display the currently set AUTO command line for the disk in drive ":1". If the above command followed such a command as:

```
AUTO :1 !date
```

That command would have set the library command DATE to execute whenever the disk in drive ":1" was booted. If the interrogation command followed, the system would display something such as this:

```
AUTO :1 ?
!date
```

To let you know the current status.

Perhaps one of the most useful aspects of AUTO is to execute a "system configuration" file. For a explanation of how to create these files, see the command SYSTEM. For right now, suffice it to say that the system configuration file offers an easy method to alter your DOSPLUS configuration to meet any special needs or desires that you might have.

Once you configure your DOSPLUS, you will create these files using SYSTEM. In order to resume the configuration you just set once the machine is reset, you merely execute these files. The AUTO command allows you to execute one of these configuration files without having to enter the filename each time. If you wish, you can even make it invisible so that you don't have to be reminded of the file loading each time.

Simply set the AUTO command to the name of your configuration file. For example, if you had installed hard disk drivers and then saved the configuration in a file called RIGID/CFG, every time you wanted to re-load the hard disk configuration, you would just execute RIGID/CFG. This allows you to boot from the floppy and with a single filename transfer control to the hard disk. You would set that file on an AUTO. Something like this:

```
!RIGID/CFG
```

which would cause that configuration file to be loaded each time the system disk was booted. The filename would NOT be displayed.

Examples:

AUTO SYSCON

This command tells DOSPLUS that upon power-up, it is to load and execute the file SYSCON/CMD (the /CMD extension is assumed). If this were a system configuration file, the system would be automatically configured and all needed drivers loaded every time the machine is reset.

AUTO *SYSCON

This command tells DOSPLUS the same thing except that this time the AUTO command will always be executed, even if the ENTER key is being held down to indicate an abort.

AUTO *!SYSCON

This command also tells DOSPLUS to load and execute the file SYSCON/CMD and also tells it to ignore the abort signal. However, this command also tells DOSPLUS not to display the AUTO command as it is executing. In the above two examples, the word SYSCON would appear on the screen as the file was being executed. In this example, it would not.

AUTO :1 DO START

This command will set the AUTO on the disk in drive one to "DO START". Whenever the system is booted using that disk, DOSPLUS will attempt to execute the DO file START/TXT. Remember, the /TXT is the default extension for the DO command. You may specify differently if you wish.

AUTO :B

This command will reset the AUTO command on the disk currently in the drive named :B.

BOOT

This command will allow you to perform a cold system reset from software.

==========

BOOT

There are no parameters for this command.

==========

This function is the same as pressing the reset button. All drivers and configurations are returned to their default levels. The BOOT command is most useful when you wish to have the system reloaded under program control.

You must have the disk in place in the system drive when executing this command. Failure to do so will result in a boot error.

Because this is in effect a system reset, any AUTO functions or DO files that normally start on power-up will begin after this command also. You may abort them, provided they are not non-breakable, by holding down the ENTER key.

You may be prompted for the date and time when booting up. This is a configurable option that may be disengaged by using the SYSTEM command. You may also disable the opening logo if you wish (see SYSTEM).

You may also activate the system debugger by holding down the D key as DOSPLUS boots up. This enables you to go directly to the system's built-in memory monitor and proceed to examine memory without having to go through any start-up procedures or even going to the system level at all.

However, when booting DOSPLUS, if you have any of the prompts engaged (i.e. time or date), the system will pause at those prompts before engaging whatever options you have indicated during boot up. For example, if you hold down the D key to enter the debugger, it will not jump to DEBUG until after the prompts (if any) have been answered.

The same holds true for aborting an AUTO. If you haven't been holding down the ENTER key as the system is booting, it is too late when you find yourself at the prompt.

Example:

BOOT

This command will cause the DOSPLUS system to reboot.

BREAK

This command allows you to enable or disable the break key.

==========

BREAK [switch]

[switch] is the optional switch.

Your switches are:

ON	Enable break key.
OFF	Disable break key.

==========

The BREAK command allows you to manipulate the break key. In some applications, it may be desirable that the user not be able to use the break key (i.e. some DO files or BASIC programs).

All that is needed is for you to use this parameter to turn the break key off and the system will not respond to the break key. Normally, the break key serves as the "abort" for certain functions. If, for example, a PAUSE (see PAUSE) occurrs during the execution of a DO file and the user responds by pressing the break key, they will be returned to DOS. If the break key has been turned off with this parameter, it will simply be ignored.

For some programs, this is not desirable, so use caution with this command. Once you turn off the break key, you have in effect removed it from the system. Until enabled or the system is rebooted, this key will not be recognized if the program depends on the DOS to inform it when the break key has been pressed. Programs that contain their own keyboard drivers may not be affected by this.

Examples:

```
BREAK OFF
BREAK NO
BREAK N
```

This command will disable the break key.

```
BREAK ON
BREAK YES
BREAK
```

This command will enable the break key.

BUILD

This command offers you the ability to create an ASCII text file on the disk or output ASCII statements to any device.

==========

BUILD devicespec/filespec [param=switch...]

devicespec/filespec is the standard DOSPLUS device or file specification that indicates where the output of the BUILD command should be directed.

[param=switch...] is the optional parameter to modify the command's action.

Your parameters are:

APPEND=switch	Optional switch to indicate that you wish to append the instructions you are about to enter on to the end of an already existing file.
APPEND=value	Optional value for switch to indicate where in the existing file you wish to begin appending statements.

Abbreviation:

APPEND A

==========

This command allows you to output ASCII statements to any device or file in the system. You may use this for a variety of purposes. The most common by far will be creating ASCII text files on the disk for use with the various DOSPLUS library commands.

As stated, the BUILD command allows you to construct an ASCII file on the disk. This makes it one of the most often used commands in the entire DOSPLUS system. By using this command, you may create a file on the disk that allows you to store command lines just as you would have entered them from the DOS command mode and execute these later with the DO command (see DO), allows you to create ASCII files with lists of patches in them for use with the PATCH utility (see PATCH), and create ASCII text files that are interpreted by the FILTER library command (see FILTER) and used to modify data as it moves from driver to device.

You do not have to use the BUILD command for these. Any ASCII file will work. We have merely provided this means of accomplishing the creation of these files. This means that you may also create these files from BASIC or machine language applications programs (such as a word processor). Therefore, your programs could create the needed files based on information supplied by the user and the user would never actually interface with the DOS.

However, the BUILD command offers you the ability to create these files easily from the DOS command mode without having to load some intermediate program to do it. For the most part, with the exception of special cases, you will find that BUILD handles the task adequately and there will not be a need for you to use anything else. The only exception might be the fact that BUILD doesn't offer any editing capacity.

BUILD will assume the extension /TXT unless another is given it. That is also the default extension for the DO command (see DO).

When you enter the BUILD command (i.e. BUILD TEST:0), you will see the following initial prompt:

```
Enter text [79 characters/line]

  1:
```

At that point you are free to type up to 79 characters of text. When you have finished typing a line, press ENTER to store that line. When you are finished, press BREAK at the next blank line and BUILD will return to DOSPLUS.

If you specify the APPEND option, BUILD will display whatever lines are currently in the file and then prompt you for the new data. Any text entered will be added to the end of the existing file. If you specify a line number with the APPEND parameter (i.e. BUILD TEST:0,APPEND=4), BUILD will list the file up to the specified line number and begin adding text from there.

When using BUILD to create text files for DO, if you wish to print a line if instructions or comments on the screen, you may do so. Any line that begins with a period "." will not be executed by DOSPLUS. Therefore, to place non-command lines into your DO file, simply start them off with a period. For example:

```
.Insert the #1 disk
```

is a comment line and:

```
DIR:1
```

is not.

Comment lines may also be used in patch and filter files to identify the patch or filter for future reference. the syntax is the same. Simply start the line off with a period (".") and both PATCH and FILTER will ignore it.

Also, when entering lines into a file, you may press <LEFT ARROW> to delete a character and <SHIFT> <LEFT ARROW> to delete a line. No other editing functions are supported.

Sample use

Let's assume that we are going to use BUILD to create a text file to be used by DO as a startup sequence for a BASIC program. It might look like this:

```
BUILD STARTUP/BLD:0 <ENTER>
Enter text [79 characters/line]

    1:FORMS (W=80) <ENTER>
    2:BASIC MENU/BAS (F=1,M=65000) <ENTER>
    3:<BREAK>
```

This example would build a file called STARTUP/BLD on drive :0. This file would be accessed by the statement:

```
DO STARTUP/BLD
```

Notice that the /BLD extension was used when we called DO because we didn't use the default extension of /TXT. This file, when executed, would set FORMS for 80 column paper (see FORMS) and then enter BASIC with one file buffer allocated and memory protected at 65000. Once in BASIC, DOSPLUS would execute the BASIC program MENU/BAS.

Examples:

```
BUILD TEST:0
```

This command would open the file TEST/TXT on drive :0 and store your text there. If a file by that name is already on that drive, the current information will be overlaid.

```
BUILD TEST:0 (APPEND=Y)
BUILD TEST:0 (A=Y)
BUILD TEST:0,A
```

These three commands will all have the same effect. They also will open a file TEST/TXT on drive :0, but if this file already exists; BUILD will display the contents and then append any new text to the end of the file.

```
BUILD MKEY/TXT:1 (APPEND=6)
BUILD MKEY/TXT:1,APPEND=6
BUILD MKEY:1,A=6
```

These three commands all have the same effect. It will search drive ":1" for the file MKEY/TXT. If it is not found, it will create the file and begin input. If it IS located, the lines 1-5 will be displayed and input will begin with line 6.

CAT

This command will display a disk's file catalog.

==========

CAT (FROM) drivespec (TO) file/device (USING) wildmask (param=exp...)

drivespec is the name of the drive for which you desire the file catalog.

file/device is the optional output file or device.

wildmask is the optional wildmask to restrict CAT to a certain group or class of files.

(param=exp...) is the optional action parameter that indicates what type of file catalog you want to see.

The parameters are:

SYSTEM=switch	Display system files as well as standard entries.
INVIS=switch	Display both visible and invisible user files.
KILL=switch	Display names of any deleted files not yet wiped from the directory or over-written by an active file.
ALPHA=switch	Display names in alphabetical order.
MOD=switch	Display files based upon status of the MOD flag.

Abbreviations:

SYSTEM	S
INVIS	I
KILL	K
ALPHA	A
MOD	M

==========

The CAT command is used to display a disk's file catalog (hence the name "cat"). A disk's file catalog is simply a list of files currently residing on that disk. A file catalog will contain ONLY a filename and extension. If you require more information, then request the disk's file directory (see DIR).

The CAT command has two basic types of function: standard and global. In a standard CAT, you will get a catalog only of the drive you request. In the global form, engaged by using a wildmask, you will receive a file catalog of all mounted disk drives. Used in conjunction with a specific enough wildmask, this can be very useful for ascertaining where in the system a file is currently located.

When you request a file catalog, you may also specify the output file or device. If you do not specify a file or device specification when you issue the CAT command, the file catalog will be displayed on the screen. The display (i.e. @DO) is the default device. This feature allows you to output the file catalog to the printer, a disk file, or wherever it may be required.

The simplest form of CAT is:

```
CAT
```

which will display a file catalog of all visible user files on the system drive. Next simplest would be:

```
CAT :1
```

which has the same effect, but restricts itself to those visible user files located on drive 1.

If the switch is not specified in the parameter list, it defaults to "off". For example, if you do NOT specify the SYSTEM option in the parameter field, it will default to SYSTEM=N (e.g. no system files will be included in the file catalog). On the other hand, because of this, the simple inclusion of the option in the parameter field is sufficient to engage it. For example:

```
CAT :0 (SYSTEM=Y)
```

and:

```
CAT :0 (S)
```

are equivalent commands. This applies to all of the optional parameters on CAT. The simple inclusion of the name of the option is sufficient to engage it and the exclusion of the name will cause the option NOT to be in effect.

The file catalog display

When you request a file catalog, your output should look something like this:

```
Drive: 0  [DOS:IV    07/06/83] - Space: 077/128    42.0k

MEMDISK/DVR    MEDIC/CMD      EPSON/FLT      DVORAK/FLT     TEST
FILE/DVR       TEST/TXT       MKEY/DVR       SPOOL/DVR
```

Note that are 5 filenames across. CAT will continue to display files up to one full screen. At that point, it will pause and wait for you to press ENTER or BREAK. If you press ENTER, it will display the next screenful. Pressing BREAK will abort the command.

You may call CAT from BASIC without problems unless you wish to use the ALPHA option for an alphabetical file catalog. This cannot be used from within a BASIC program inasmuch as when you ask for a sorted file catalog, the memory required to do the sort expands past the limits of BASIC's overlay area for DOS commands and will corrupt the BASIC program itself.

Specifying output files and devices

When using CAT, if you wish to specify an output file or device other than the display and you have NOT specified a source drive, you must use the delimiter TO to indicate data flow. This would occur if you were going to get a printout of the file catalog for the system drive. To type:

```
CAT @PR
```

would produce an error, since @PR is in the source field position and @PR is not a valid drivespec. However:

```
CAT TO @PR
```

would work just fine. This does not apply if you are using a source drivespec, because then the the output device is in its proper location. For example:

```
CAT :1 @PR
```

would work properly. @PR is in its proper position and all output will be directed to the printer.

Specifying wildmasks

The only exception to this rule of order is a wildmask that contains wildcard characters. If the wildmask contains a wildcard character (i.e. ?, *, or !), then the DOS will move that to the wildmask position for you and scan the rest of the line in normal order. For instance:

```
CAT :0 USING */BAS
```

is the same thing as:

```
CAT */BAS :0
```

The system will move the */BAS to the wildmask field and accept :0 as the source drivespec. This does not apply if the wildmask doesn't contain any wildcard characters. A wildmask without wildcard characters (with the source drivespec explicitly given) is regarded by the DOS as a valid output filespec. If the source drivespec is not given (e.g. implied), then the filespec will be moved into the source field and an error will result.

If you wish to specify a wildmask without any wildcard characters such that only files EXACTLY matching the wildmask will be included, then include the USING delimiter. For example:

```
CAT :0 TEST/DAT
```

will output the CAT into the file TEST/DAT, while:

```
CAT TEST/DAT
```

will produce an error, and:

```
CAT USING TEST/DAT
```

will function properly.

Follow these rules of order on CAT and you should never get an "Invalid parameter" error. The best rule of thumb is, if you cannot remember whether or not the delimiter is required, include it. It never hurts to have it in the command line, but sometimes it will cost you to omit it.

One other important area to remember is overwriting files by accident. It can occur if we are not careful. The correct form of the command is:

```
CAT <source> <destination> <wildmask> <parameters>
```

If you wish to only specify the source drivespec and a wildmask (e.g. you wish to let the destination default to the screen), then you must either have a wildcard character in the wildmask or use the USING delimiter. There is no way around this.

A wildmask in the destination field that does not contain any wildcard characters will be regarded as the output filespec and the file catalog will be placed into that file. This can destroy the very file that you were seeking to locate.

Examples:

```
CAT :0 (SYSTEM=Y,INVIS=Y,KILL=Y)
CAT :0 (SYSTEM,INVIS,KILL)
CAT :0 (S,I,K)
CAT :0,S,I,K
```

All four of these command lines will perform the same task. They will display a file catalog of the disk in drive :0. The catalog will include all filespecs, whether system, invisible, active or deleted.

```
CAT USING PER/DAT
```

This will search the directory of all available drives and printout a file catalog for any drive having the file PER/DAT on it. This is an example of the method that would be used to locate all occurrences of the file.

```
CAT */CMD TO @PR
```

This example will scan all drives and printout the filespecs of any files that have the extension /CMD.

```
CAT :1 (INVIS=Y,ALPHA)
CAT :1 (I,A)
CAT :1,I,A
```

These three commands are all equivalent. They will display, in alphabetical order, all the user files, both visible and invisible, located on the disk in drive 1.

CLEAR

This command allows you to fill either a file or user memory with user defined data.

CLEAR (filespec) (param=exp...)

filespec is the optional file specification indicating that you wish to operate on a file and which file is to be affected.

(param=exp...) are your optional parameters.

Your parameters are:

START=value	Starting memory address.
END=value	Ending memory address.
DATA=value	Optional fill data.

Abbreviations:

START	S
END	E
DATA	D

The CLEAR command will allow you to fill either a file or a specified amount of memory with a user-definable one or two byte value. The command has two very distinct forms (e.g. file and memory) and certain of the parameters only function in the proper mode.

The two modes of CLEAR are mutually exclusive. Which is to say that you cannot mix the two. While you are clearing out a file, you cannot be clearing memory and vice versa. To fill a file using CLEAR, simply specify a filespec after the CLEAR command. To fill memory, omit the filespec. It is that easy.

Remember, if you use CLEAR to erase a file's data on the disk, that file is gone! There is no way to recover data that has been CLEARed out. The same is true for data resident in RAM. If you use CLEAR to remove it, there is no way to ever recover it.

START. This parameter allows you to specify the starting address of the area to fill when doing a memory CLEAR. This parameter does not effect the file CLEAR and will be ignored if specified when a filespec is given. If you indicate a memory CLEAR by omitting the filespec, but do not expressly state the START address, 5C00H (23552 decimal) will be used.

END. This parameter allows you to specify the ending address of the area to fill when doing a memory CLEAR. As with START, the END parameter does not affect a file CLEAR and will be ignored if specified with a filespec. If you indicate a memory CLEAR and do not give the END address, the address currently defined as the top of memory will be used.

DATA. This parameter allows you to specify a one or two byte value to be used during the fill operation. This parameter is valid for both file and memory CLEARs. To use it, simply specify:

DATA=value

where "value" is the one or two byte value you wish to use. This value may be given in decimal or hexadecimal format. Remember to append an H to any hexadecimal entries.

When you are specifying the DATA parameter, if you only specify a one byte value, then CLEAR simply duplicates the first byte into the second when filling. Which is to say that CLEAR always fills with a two byte value. It simply allows you to only specify one byte if you want the same value in each.

What this means is that the values 6C00H and 006CH will react very differently. In the first case, CLEAR will fill with a data pattern of "6C006C006C00" where the second will use a pattern of "6C6C6C6C6C6C". A leading zero is ignored, a trailing one is not. When using a decimal value, anything between 0 and 65535 is valid.

It will prove most convenient to be able to clean out memory or a file when writing programs. The ability to clear out a file will allow you to "start over" with fresh data space.

The CLEAR command will not allow you to clear out memory below the value 3000H or above the value currently set as the top of memory. By default, it fills between those two. Therefore, if your goal is to fill all user memory, it would be simpler to just omit the START and END parameters.

Examples:

CLEAR

This example will fill all of user memory (the area between 5C00H and the top of memory) with zeros.

```
CLEAR (START=3000H,END=7000H,DATA=6CH)
CLEAR (S=3000H,E=7000H,D=6CH)
CLEAR,S=3000H,E=7000H,D=6CH
```

These three commands are equivalent. All three of them will fill memory between addresses 3000H (12288 decimal) and 7000H (28672 decimal) inclusive with the value 6CH (108 decimal).

```
CLEAR TESTFILE/TXT
```

This command will instruct the system to fill the file TESTFILE/TXT with zeros.

```
CLEAR DATA:TD (DATA=229)
CLEAR DATA:TD (D=229)
CLEAR DATA:TD,D=229
```

These three commands will all accomplish the same thing. They will search the drive named ":TD" for the file DATA. If the file is located, CLEAR will fill it with the value 229 decimal (E5H).

CLOCK

This command allows you to turn on and off the display of the system clock.

CLOCK switch

switch is the optional switch to inform DOSPLUS whether to turn the clock display on or off.

Your switches are:

ON	Display on.
OFF	Display off.

By using this command, you can display the real time clock in the upper right hand corner of the screen. This can be useful in certain applications to indicate to the operator that the time has not been set (i.e. if they see a time of "00:00:00").

The system powers up with the clock turned off, unless you have the clock turned on when you save a configuration file (see SYSTEM). You may either set the time at that prompt upon powerup, or after powerup by using the TIME command (see TIME). When the clock reaches "23:59:59", it will reset itself to "00:00:00" and increment the date by one day. The clock display will be updated once a second.

If you use the system command to disable the time prompt or skip the prompt by pressing ENTER or BREAK, DOSPLUS will attempt to recover the time last set. If, and only if, the values are out of range for a legal time value, "00:00:00" will be used. Also, please note that the CLOCK command affects only the display of the clock. Turning the clock off does NOT shut off the system clock, merely the display.

If you use the CLOCK command without any switches, ON will be assumed.

Examples:

```
CLOCK ON
CLOCK
```

This command turns on the clock display.

```
CLOCK OFF
```

This command turns off the display.

CLS

This command clears the display and resets the video mode.

==========

CLS

There are no parameters for this command.

==========

This command, when executed, will cause the display to be cleared immediately. It will also cause the video mode to be reset. If you are in a 40 character per line format (e.g. double wide text), you will be restored to the standard 80 character per line format.

This command was designed with two primary uses in mind.

First, under DOSPLUS, the library commands (and most of the utilities) do not automatically clear the screen before execution.

Therefore, this command becomes very useful to you. DOSPLUS allows multiple commands on the same line separated by a semi colon ";". Preceding your command with a CLS command will clear the screen before the command outputs to it. For example:

```
DIR
```

would become:

```
CLS;DIR
```

Second, you may also use this command during a JCL file to enhance display output.

Examples:

```
CLS
```

This command will clear the screen.

```
CLS;FREE :0
```

This command will clear the screen and then display a free space map for the drive named ":0" (see FREE).

CONFIG

This command allows you to configure DOSPLUS' disk drive parameters.

CONFIG [drivespec] (param=exp...)

drivespec is the optional drive specification to indicate which drive you are configuring.

(param=exp...) is the optional parameter.

Your parameters are:

Floppy drives:

WP=switch	Sets software write protect.
MD=switch	Configures for delay on motor on.
HL=switch	Configures for delay on head load.
STEP=value	Sets the drive step rate.
SKIP=switch	Sets double step mode.
SIZE=value	Indicates disk drive's physical size.
SIDES=value	Indicates number of read/write surfaces.
PDRIVE=value	Indicates which physical drive this drivespec will address.

Rigid drives:

SIZE=value	Sets platter size.
SIDES=value	Indicates number of read/write surfaces.
WP=switch	Sets the software write protect.
STEP=value	Sets the drive step rate.
HO=value	Sets the head offset.
CO=value	Sets the cylinder offset.
TS=value	Sets the number of sectors per track.
PDRIVE=value	Indicates which physical drive this drivespec will address.

Abbreviations:

Floppy drives:

WP	W
MD	M
HL	None.
STEP	S
SKIP	SK
SIZE	SIZ
SIDES	SID
PDRIVE	P

Rigid drives:

SIZE	SIZ
SIDES	SID
WP	W
STEP	S
HO	H
CO	C
TS	T
PDRIVE	P

The CONFIG command allows you to configure your DOSPLUS to operate correctly with all manner of disk drives. CONFIG will allow you to set any parameter for any drive. It is up to the driver for that drive to interpret that parameter and act accordingly.

Using the CONFIG command can be as simple or as difficult as you choose to make it. If you are operating standard Radio Shack hardware, then you do not need to use CONFIG unless you wish to change the order in which your drives are scanned or in some other way alter the regular scheme of things.

Since CONFIG has two areas of operation, floppy and rigid disks, we will cover each in turn. Many of the parameters are the same, but since the types of configurations differ so greatly we will cover each separately.

Floppy disk drives

The first step is displaying your current CONFIG settings. To do that, type :

```
CONFIG
```

and press ENTER. You should see something similar to the following :

```
$00 :0  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=0,MD
$01 :1  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=1,MD
$02 :2  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=2,MD
$03 :3  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=3,MD
$04 :4  NIL
$05 :5  NIL
$06 :6  NIL
$07 :7  NIL
$08 :8  NIL
$09 :9  NIL
$10 :10 NIL
$11 :11 NIL
$12 :12 NIL
$13 :13 NIL
$14 :14 NIL
$15 :15 NIL
```

Note: The above settings are the standard default settings for DOSPLUS. Unless you received some variety of special "pre-configured" system, this is the manner in which your DOSPLUS should be set when you receive it.

The first item displayed is the drive device number. As you can see from the numbers 0 through 15, there are 16 drive devices in the DOSPLUS system. You may define these in any manner you wish up to a maximum of four physical floppy drives and four physical rigid drives. You may have more than one drive device referencing the same physical disk drive.

The next item displayed is the drivespec. The drivespec is simply the name by which you reference the disk drive. This has no relation whatsoever to the manner in which the drives are scanned (the drives will always be scanned in the order of device number, starting with 0 and proceeding to 7) or any other area of drive performance. These may be changed via the RENAME command to suit the needs and desires of the user. The only restriction is that you may not have two drives with the same drivespecs (see RENAME and File and Device Specifications).

After those two items, the various parameters for each drive will be displayed. Let's cover now those used for floppy drives:

Floppy disk parameters

Floppy

Floppy media. This parameter indicates that the drive device whose CONFIG line it appears in is currently defined as a floppy disk drive. This is controlled by the driver program and cannot be altered by the user without changing which driver is installed for that device. You would accomplish this via the ASSIGN command if it is so desired. This parameter's only purpose in the display line is to inform you which type of driver is in effect.

Dden or Sden

Media density. This parameter indicates the density of the drive whose CONFIG line it appears in. This is also not a user alterable parameter. DOSPLUS will automatically recognize the density of a disk (e.g. single or double) and will adjust itself accordingly. For your convenience, this information is displayed here. It will either read "Dden" or "Sden", depending on the density of the media.

Dden, of course, is double density while Sden is single density. Having this parameter displayed like this allows you to, at a glance, be informed as to what type of media is mounted in each disk drive.

Size

Physical disk size. This parameter displays and allows you to configure the physical size of the media. We are referring to whether the drive is 5 or 8 inch. This parameter allows you to alter that as required. Be advised that the standard floppy disk I/O drivers no longer support 8 inch disk operation. If you wish to use an 8 inch floppy drive, you will have had to install the proper alternate driver first.

What all this means is, if you wanted to operate an 8 inch disk drive (or any non-standard disk drive, for that matter), the first step is to ASSIGN the proper drivers. After that, you simply use CONFIG to set the drive parameters as needed. Since you are operating an 8 inch drive, you would CONFIG size equal to 8.

```
Example: CONFIG :2 (SIZE=8)
         CONFIG :AA (SIZE=5)
         CONFIG :3,SIZ=8
```

Sides

Number of sides. This parameter allows you to configure DOSPLUS to access double sided disk drives. The actual creation of a double sided disk is handled by the FORMAT utility. When FORMAT is prompting you for the disk information, one of the questions asked you will be "Number of sides?". If you respond with a "2", FORMAT will create a double sided disk. Every time FORMAT initializes a disk's system information it stores a table on that disk that is used by the system to inform DOSPLUS what sort of disk it is. This table is called a DCT (Drive Control Table). One of the items stored in this table is whether a disk is single or double sided. DOSPLUS will automatically adjust for sides each time that it accesses a disk. There may sometimes be a reason, though, in which you wish to force DOSPLUS to double sided recognition (perhaps the disk is double sided, but has not been formatted by DOSPLUS and therefore lacks the information in the DCT) This can be accomplished here.

This parameter allows you to adjust manually what the system is capable of automatically. An automatic function that works only automatically is poor at best. Please be aware of the fact that double sided operation is not a software function. Without the software, the hardware won't operate, but the software is not an end to itself. The standard Radio Shack disk drives are not double sided. If you are not very aware of the fact that you have double sided disk drives, you probably don't.

Example: CONFIG :2 (SIDES=1)
CONFIG :A (SIDES=2)
CONFIG :2,SI=1

Step

Step rate. This parameter displays and allows you to alter what step rate DOSPLUS will use for the various disk drives in the system. The value used here is NOT an exact track to track step rate but rather a relative value that the DOS then interprets. Your values are :

Value	Step rate
0	6 milliseconds
1	12 "
2	20 "
3	30 " (double density)
	40 " (single density)

A drive's step rate determines how much time the system allows for the read head to move between cylinders. Drives with a low track to track access time can step the head faster than those with a high track to track access time.

DOSPLUS sets the drive step rate at two locations. The first, controlled by SYSTEM is the system default step rate. This step rate is what will be assumed for all drives on power up. However, you may have certain disk drives in your system that cannot step as fast as all the others. For those drives, you should use the second method, which is to use CONFIG to individually alter the step rate to whatever is needed.

When you use CONFIG to alter a step rate individually, you must store this as part of a configuration file (see SYSTEM), and execute that file to restore the configuration later. On the other hand, the default system step rate is written to the disk's DCT each time it is altered.

Example: CONFIG :0 (STEP=0)
CONFIG :DS (STEP=2)
CONFIG :3,S=1

PDRIVE

Physical Drive. This parameter displays and allows you to alter what actual, physical drive a particular drive device addresses. You have four possible floppy disk drives (0 through 3). Any drive device may address any one of these. Two drive devices may address the same physical drive, if desired.

This parameter is what is used to reorder the drives. To reorder the drives means that you change the order in which the drives are scanned. In a standard system, DOSPLUS will scan from drive device 0 to drive device 7 in ascending order. This may not always be what you desire. To effect a change, you may use this parameter.

To accomplish this, simply place the various Pdrive parameters into the drive device list as you want them scanned. In our example above, drive 1 would be scanned before drive 2. But if the Pdrive value for drive 1 was 2 and the value for drive 2 was 1, this would be reversed. DOSPLUS, during a global operation, would scan drive 0, then drive 1 (which would be physical drive 2), then drive 2 (which would be physical drive 1), and finally drive 3. If you didn't like having physical drive 2 addressed as logical drive 1, you could use the RENAME command to alter the drivespecs (see RENAME).

Therefore, by changing the order in which the physical drive numbers appear the drive device list, you change the order in which the drives are scanned. You may then alter the drivespecs with RENAME to read any way you like. The most important use of the parameter, though, is simply when you are creating your system configuration, be certain that all of the physical drives present in your system have at least one drivespec assigned to them if you hope to access them later.

Example: CONFIG :2 (PDRIVE=1)
CONFIG :1 (PDRIVE=2)
CONFIG :2,P=1

WP

Software Write Protect. This parameter does not appear in our example, but were it to be set, this is the location that it would occur in the CONFIG display line (e.g. immediately following the physical drive number), so we will cover it here. This parameter allows you to set a software write protect option for any logical drive. This has exactly the same effect as engaging a hardware write protect (e.g. the system will not write to that drive).

The advantage is that this can be set and reset easier than you can engage a hardware write protect and also often the logical drive is simply a portion of a physical drive or is really a file within a drive (see under Driver and Filters, FILE/DVR). You cannot engage a hardware write protect in such instances.

If this option is engaged for that drive, the letters WP will appear in the display line for that drive. If these letters are not present, then the option is not engaged. If you do not specify Y or N when mentioning WP in a CONFIG command line, Y will be assumed.

```
Example: CONFIG :2 (WP=Y)
         CONFIG :++ (WP=N)
         CONFIG :2,W=Y
         CONFIG :2,W
```

MD

Motor on Delay. This parameter allows you to configure DOSPLUS to operate with drives that only switch on the motor when selected or drives that run their motors constantly. This is primarily for use with the 8 inch drives. All standard 5 inch drives will not switch on the motor until a drive is selected. Because of this, the system has to delay slightly while waiting for the drive to come to speed. On the other hand, many of the 8 inch drives run the motor constantly, so having the DOS delay in those cases would be a needless waste of time.

You will notice that the MD parameter was present in all the floppy disk display lines of our CONFIG example. If you do not specify a switch when using this parameter, Y will be assumed.

```
Example: CONFIG :1 (MD=Y)
         CONFIG :2 (MD=N)
         CONFIG :1,M=Y
         CONFIG :1,M
```

HL

Head Load delay. This parameter allows you to DOSPLUS to operate with drives that load the head on motor on and drives that load the head with drive select. When a drive's read head is against the media ready to read or write data, that head is referred to as being loaded. Certain drives keep the read head against the media all the time. Others load and deload the heads between accesses. Of the drives the load the heads, there are two ways they can do it.

The first method is called head load with motor on. This means the drives load the heads whenever the motor on signal is received. Since all drives engage their motors when any one drive is selected, this would mean that all drives would load their heads when any one drive is selected. Because we already delay for the motor on signal, there is no need for an additional head load delay initially and because the heads are all loaded when the first drive was selected, we don't need an additional delay for head load when moving between two drives.

The second method is called head load with drive select. In this method, each drive keeps its read head deloaded until that drive is specifically selected for use. This would mean that we would have the heads constantly loading and deloading as we moved data between two drives. Because of this, an additional delay will be required at each drive selection to allow the heads to load. By setting the head load parameter, we accomplish this.

To summarize, if your drives keep the head loaded against the media all the time, then you do not need this parameter set. If your drives load the head with the motor on signal, then you still don't need this parameter. The only time that you need this parameter set is when you are functioning with drives that load the head with drive select.

Skip

Double step drive. This parameter allows you to instruct a drive to double step, or read every other track. Again, this parameter doesn't appear in our standard example, but if set, this is where in the line it will occur (e.g. after the HL parameter). This is primarily used to read 40 track disks in 80 track drives. 80 track disk drives use a track density of 96 TPI (Tracks Per Inch). 40 track drives use a density of 48 TPI. 80 track drives write data exactly twice as dense. Therefore, if you instruct an 80 track drive to skip, or read only every other track, it will read at half its regular density or 48 TPI. This will allow it to read a 40 track diskette.

Caution! Please do NOT write to a standard 40 track disk in a skipped 80 track drive. Not only do 80 track drives use twice the tracks per inch density, but the actual tracks themselves are somewhat smaller. This causes no problem when you are only reading, but should you write to the disk, the track would not completely overlay the old one. Then, when you moved it back to the 40 track drive, that drive would read a portion of the old track as well as the new when attempting to read this disk. This would, of course, render that track unreadable.

It is vital that you remember this. If you are using an 80 track drive to backup a 40 track disk, you could cause serious problems. BACKUP seeks to clear all mod flags in the directories of both disks after making the backup. To do that, it writes to both the source and the destination directories. When it writes to the source disk, a 40 track disk in a skipped 80 track drive, it will ruin the directory. Therefore, before using a skipped 80 track drive to backup a 40 track disk, either write protect that disk or set the WP parameter on CONFIG for that drive. Failure to do so will make the disk unusable in the 40 track drive.

The simplest way to avoid this is to never write to a 40 track disk using a skipped 80 track drive. And when using a skipped 80 track drive to backup a 40 track disk, either hardware or software write protect that disk. An execellent rule is to always engage software write protect at the same time you engage the skip option for any given drive.

The same warning applies when copying a file from that disk. COPY will also seek to remove the mod flag for the file it just copied. This will cause it to write to the source disk directory. Please be aware of this and prevent it before losing any disks. This parameter is too useful to be removed just because it can cause a problem if misused, so you (the user) are responsible for seeing that it is properly implemented. When using the Skip parameter, if you do not specify a switch with the parameter, Skip=Y will be assumed.

```
Example: CONFIG :2 (SKIP=Y)
         CONFIG :3 (SKIP=N)
         CONFIG :2,SK=Y
         CONFIG :2,SK
```

If we were to set all of the possible floppy disk drive parameters for one of the drives listed in our example above, it would looks something like this :

```
$00 :0  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=0,MD
$01 :1  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=1,MD
$02 :2  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=2,WP,MD,HL,Skip
$03 :3  Floppy,Dden,Size=5,Sides=1,Step=3,Pdrive=3,MD
$04 :4  NIL
$05 :5  NIL
$06 :6  NIL
$07 :7  NIL
$08 :8  NIL
$09 :9  NIL
$10 :10 NIL
$11 :11 NIL
$12 :12 NIL
$13 :13 NIL
$14 :14 NIL
$15 :15 NIL
```

Note that drive 2 now has all available options engaged for it. This is how the CONFIG line would appear in such cases.

Rigid disk drives

This first step in CONFIGuring rigid disk drives is to ASSIGN the drivespec you wish to use with the proper rigid disk driver. You will do this via the ASSIGN command. For more specific information and exact syntaxes for installing the various drivers, please look up that driver in the section Drivers and Filters elsewhere in this manual. For our purposes here, we will assume that you have already installed the driver and will deal simply with changing the parameters.

To display the current CONFIG settings, type :

```
CONFIG
```

and press ENTER. You should receive something similar to the following display :

```
$00 :0  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=0,MD
$01 :1  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=1,MD
$02 :2  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=2,MD
$03 :3  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=3,MD
$04 :4  Hard,Fix,Size=5,Sides=0,Step=6,Pdrive=0,CO=0,HO=0,TS=0
$05 :5  NIL
$06 :6  NIL
$07 :7  NIL
$08 :8  NIL
$09 :9  NIL
$10 :10 NIL
$11 :11 NIL
$12 :12 NIL
$13 :13 NIL
$14 :14 NIL
$15 :15 NIL
```

Note: The above example is of a standard DOSPLUS after installing a rigid disk driver and before any further installations or configurations. Yours may appear slightly differently regarding the setting of the parameters (depending on the individual driver), but the parameters should remain the same.

The first item displayed is the drive device number. As you can see from the numbers 0 through 15, there are 16 drive devices in the DOSPLUS system. You may define these in any manner you wish up to a maximum of four physical floppy drives and four physical rigid drives. You may have more than one drive device referencing the same physical disk drive.

The second item displayed is the drivespec. The drivespec is simply the name by which you reference the disk drive. This has no relation whatsoever to the manner in which the drives are scanned (the drives will always be scanned in the order of device number, starting with 0 and proceeding to 7) or any other area of drive performance. These may be changed via the RENAME command to suit the needs and desires of the user. The only restriction is that you may not have two drives with the same drivespecs (see RENAME and File and Device Specifications).

After those two items, the various parameters for each drive will be displayed. Let us cover now those used for rigid drives :

Rigid disk parameters

Rigid

Rigid media. This parameter indicates that the drive device whose CONFIG line it appears in is currently defined as a rigid disk drive. This is controlled by the driver program and cannot be altered by the user without changing which driver is installed for that device. You would accomplish this via the ASSIGN command if it is so desired. This parameter's only purpose in the display line is to inform you which type of driver is in effect.

Fix or Rem

Fixed or removable platters. This parameter indicates that the driver you have installed for that drive is set to work with either fixed platter (non-removable) or removable platter drives. If you are going to be removing the platters and replacing them with a new set, the system needs to be informed that this might be the case at any given time. This, however, is NOT a user option. It is controlled by the driver. If the driver you have installed will operate with removable platter drives, then the word "Rem" will appear.

Size

Platter size. This parameter allows you to configure DOSPLUS for the actual physical size of the drive's platters. Some drivers may work with several different units from a single manufacturer by simply altering the CONFIG line to reflect the proper settings for that drive. One of the items that may change between the units is the physical platter size for the drive.

This is referring to 5 or 8 inch platters. After installing the drive, you would simply configure the drive for whatever size hardware is correct. At this writing, the majority of the drives being sold used the 5 inch drives, so this should be by far the most common size encountered.

Example: CONFIG :4 (SIZE=5)
CONFIG :5 (SIZE=8)
CONFIG :4,,SIZ=5

Sides

Number of surfaces. This parameter allows you to use CONFIG to inform the rigid disk driver regarding the number of surfaces on the drive that a particular drivespec addresses. This is measured in sides or number or read/write surfaces. This is exactly twice the number of platters, because each platter contains exactly two read/write surfaces.

Therefore, if you have a three platter drive, you have 6 sides. A 2 platter drive has 4 sides, and a single platter drive has 2. The rule is, when configuring your rigid disk, multiply the platter count by two and set sides equal to that. There are some restrictions involved here. You may not have more than 256 sectors on any given cylinder.

A cylinder is defined as being all like-numbered tracks all on all platters. You may not have more than 256 sectors on any one of these. To determine how many sectors you have on a cylinder, multiply the number of surfaces (sides) by the number of sectors on each surface (TS or Track Size, covered later). If, for example, you have a Track Size of 32 (32 sectors per track or surface), then the maximum numbers of platters allowed would be 4. 4 platters are 8 sides and 8 time 32 is 256, the maximum number of sectors per cylinder.

For the most part, this will not even concern you. For each drivespec that you are configuring, set the sides value equal to the number of platters multiplied by two.

Example: CONFIG :4 (SIDES=4)
CONFIG :5 (SIDES=6)
CONFIG :4,SI=4

Step

Drive step rate. This parameter allows you to set the relative step rate for the drive. This will be a value between 0 and 255. It has no relation to the actual rate at which the drive steps. Different drivers will require different values. The same driver may require a different value for two separate drives.

Each driver description should contain what step rates are valid for which drives. Simply configure each drive according to the information included with that driver.

Example: CONFIG :6 (STEP=6)
CONFIG :7 (STEP=128)
CONFIG :4,STEP=0
CONFIG :6,S=6

PDRIVE

Physical Drive. This parameter allows you to control what physical drive a drivespec addresses. You may only have a maximum of 4 physical hard disk units (drives 0 through 3) attached to your machine. However, you may partition those using any or all of the 8 drivespecs available to you.

If, for example, you wanted to split physical hard drive 0 (the first one on the chain) into two volumes and chose to use drivespecs 4 and 5 for it, you would set the Pdrive parameter for both of those to 0. This would have both drivespecs addressing the same physical hard disk. Then, by using the other hard disk parameters, you may tell each drivespec which portion of the hard disk to use. For a detailed explanation of the concepts behind drive partitioning, consult the portion of the technical manual called Rigid Disk Partitioning.

Example: CONFIG :4 (PDRIVE=0)
CONFIG :6 (PDRIVE=1)
CONFIG :5,PDRIVE=0
CONFIG :4,P=0

CO

Cylinder Offset. This parameter is part of what allows you to control what area of the hard drive a drivespec will address. If you are dividing a 230 cylinder hard drive into two equal volumes, you would want each volume to use 115 cylinders. And, since in this case you would not want two drivespecs using the SAME 115 cylinders, you would have to use the CO parameter to tell one of the drivespecs to start at cylinder 115.

Let's assume that you are using drivespecs 4 and 5 again. Allow drive 4 to start at cylinder 0. To give drive 4 115 cylinders would encompass cylinders 0 through 114. This would mean that drive 5 would begin at cylinder 115. You would set the CO parameter for drive 5 to 115. When addressing the drive via DISKZAP (see utilities) or in any other method, that would be referred to as cylinder 0. However, CONFIG would keep track of the fact that cylinder 0 through 114 of logical drive 5 are really cylinders 115 through 229 of physical drive 0. And it would do all this via the Pdrive and CO parameters.

Example: CONFIG :4 (CO=0)
CONFIG :5 (CO=115)
CONFIG :6,CO=112
CONFIG :4,C=0

HO

Head Offset. This parameter allows you to control at which surface a logical drive will begin. This may be used to create a logical drive that only uses certain platters of a hard disk. To use this parameter, you will set HO equal to a value that represents how many surfaces to skip before starting the logical drive. This is used when you wish to partition a drive by platter either in conjunction with or in lieu of partitioning it by cylinder.

For example, to start a logical volume with the second head of a particular drive, you would use a head offset of 1, since skipping one head will cause you to begin with the second head.

This parameter becomes useful when you wish to assign each head as a separate logical volume or when dealing with drives that would otherwise have too many platters for you to address via cylinder partitioning. If you specify this in the command line, you must give an accompanying value.

Example: CONFIG :5 (HO=2)
CONFIG :4 (H=0)
CONFIG :5,H=2

TS

Track size. This parameter is used to inform DOSPLUS as to the number of sectors stored on one track of the hard disk. Do not confuse this with the number of sectors per cylinder. A cylinder may contain more sectors than a track, depending on the number of surfaces. This is not an arbitrary parameter, but rather must be set to what the hardware requires. Your drive owner's manual should have this information. If it does not, contact the drive assembler and ask them.

This parameter may not be set by the driver (depending on the drive). Always check to see that it is set before attempting to use the hard disk formatter or make any other access to the drive. A track size of 0 will usually cause unpredictable results.

Example: CONFIG :4 (TS=32)
CONFIG :6 (T=33)
CONFIG :4,T=32

Important: None of the above described rigid parameters will appear without the installation of the rigid disk driver. If your DOSPLUS does not have this driver, you do not have hard disk capability.

The following is an example of a 230 cylinder, 3 platter, 10 megabyte hard disk configured as three volumes with the CO parameter. There are three floppy disks in the system and two drive devices set to NIL.

```
$00 :4  Hard,Fix,Size=5,Sides=6,Step=6,Pdrive=0,CO=0,HO=0,TS=32
$01 :5  Hard,Fix,Size=5,Sides=6,Step=6,Pdrive=0,CO=115,HO=0,TS=32
$02 :6  Hard,Fix,Size=5,Sides=6,Step=6,Pdrive=0,CO=172,HO=0,TS=32
$03 :2  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=2,MD
$04 :1  Floppy,Sden,Size=5,Sides=1,Step=0,Pdrive=1,MD
$05 :0  Floppy,Dden,Size=5,Sides=1,Step=0,Pdrive=0,MD
$06 :A NIL
$07 :B NIL
$08 :C NIL
$09 :D NIL
$10 :E NIL
$11 :F NIL
$12 :G NIL
$13 :H NIL
$14 :I  NIL
$15 :J NIL
```

Notice that in this example, the hard disk has been installed as the system device. This was accomplished through the ASSIGN command (see ASSIGN). This operation also requires that you have formatted and sysgened the drive.

In addition, the drive scan sequence in this system has been modified such that when DOSPLUS does a global search on all drives, it searches the hard disk volumes first and then comes back to the floppies (as long as they are available).

The entire operation can be outlined as follows :

(1) Use ASSIGN to install the driver and activate that drive device.

(2) Use CONFIG to alter the parameters correctly for each volume. There may be as many volumes as you desire (or need) up to the limit of the system to accept drivespecs.

(3) Use the rigid disk formatter utility supplied with your drivers to format the hard disk. Instructions on its use will be contained in documentation sent with the specific drivers (because each may operate slightly differently).

(4) Use the SYSGEN utility to copy the system files to the hard disk.

(5) Copy all user files to the hard disk.

(6) Use the ASSIGN command to duplicate the driver information from the hard disk volume you just performed all these operations on into the system drive. This would transfer system control to the hard drive. You may then rename the drives to suit you.

Perhaps this model will be of some aid to you.

Examples

```
CONFIG :1 (STEP=1)
CONFIG :1 (ST=1)
CONFIG :1,ST=1
```

This command will cause the system to configure the step rate for drive 1 as "1" or 12 mS.

```
CONFIG :4 (TS=34,CO=0,HO=0)
CONFIG :4 (T=34,C=0,H=0)
CONFIG :4,T=34,C=0,H=0
```

This command will set the drive defined as ":4" for certain rigid disk parameters. This example assumes that you have the rigid disk drivers installed. Otherwise, any attempt to configure these parameters (which are non-existant on floppies) will result in an error.

Finally:

Remember that none of the alterations you make with CONFIG are automatically permanent. If you wish to preserve these, you must use SYSTEM to create a configuration file while these parameters are set the way that you want them. Then, by executing this file, you may resume that configuration.

Please don't be intimidated by CONFIG and the rigid disk drives. With each driver that Micro-Systems Software produces, we will attempt to produce a JCL file that will install the system for you in several different standard configurations. The only time that you really need to become involved in altering the parameters is when you are setting up some non-standard configuration and no JCL exists to aid you.

COPY

This command allows you to copy data from one point in the system to another.

---

1. COPY [FROM] device/file [TO] device/file (param=exp...)
2. COPY [FROM] filespec [TO] filespec/drivespec (param=exp...)
3. COPY [FROM] drivespec [TO] drivespec [USING] wildmask (param=exp..)

Your parameters are:

DPW="string"	Destination disk's Disk Master Password.
ECHO=switch	Display (echo) filenames as they are copied.
INVIS=switch	Copy invisible files, also.
KILL=switch	Delete source file after copying.
MOD=switch	Copy based on MOD flag condition.
OVER=switch	Prompt before overwriting.
PROMPT=switch	Prompt for disks (single drive copy).
QUERY=switch	Prompt before copying.
SPW="string"	Source disk's Disk Master Password.
TINY=switch	Copy with tiny buffer (a sector at a time).
NEW=switch	Copy only files from source disk that DON'T exist on destination.
OLD=switch	Copy only files from source disk that DO exist on destination.

Abbreviations:

DPW	D
ECHO	E
INVIS	I
KILL	K
MOD	M
OVER	O
PROMPT	P
QUERY	Q
SPW	SP
TINY	T
NEW	N
OLD	O

The COPY command is used to copy data from one point in the system to another. Whether you are copying a file, a group of files, or data to/from a device, this is the command you will use.

The average user may use COPY more than almost any other command in DOSPLUS. This command operates in three separate styles or "modes":

(1) One device/file to another copying a byte at a time.

(2) One file to another copying a file at a time.

(3) Several files from one disk drive to another copying a file at a time.

In the command syntax box, we showed syntax examples of each of the three modes of COPY. Let us now take an expanded look at each of those and what they are used for:

Mode 1

In this mode, also known as a "device copy", we are copying a:

* Device to a file.
* Device to a device.
* File to a device.

Therefore, if your copy doesn't involve a device, it is not operating in this mode.

An example of copying a device to a file might be copying the keyboard (@KI) to a disk file (FILENAME/EXT). The format would be:

COPY @KI TEXT:1

Any output from the keyboard (i.e. characters that you type...) would be sent to the disk file TEXT on drive 1.

An example of copying a device to a device would be copying the keyboard (@KI) to the printer (@PR). This would, in effect, give you a typewriter (depending on the type of the printer, of course). Any character typed on the keyboard would be copied directly to the printer without being sent to the screen or executed. The format would be:

COPY @KI @PR

An example of copying a file to a device may be copying a text file (FILENAME/EXT) to the serial communications device (@RS). This would allow you to send data directly from a disk file out the serial communications device. The format would be:

COPY TEXT:1 @RS

This would instruct the DOS to copy the file TEXT located on drive 1 out the serial communications device.

The copy will be terminated during device copies when an ETX (03H) is received. If the keyboard is the source device, this will be a CONTROL-C.

Also bear in mind that a file may function as either an input or an output device. The is not the case with all of the various system devices, though. Some of them function only as input devices (the keyboard is an example), and others only as output devices (the printer, for instance). You must always copy data from an input device to an output device. When using device copies, a filespec may appear in EITHER position. However, you must be careful to assure that the device mentioned in the other position is of the correct type (e.g. input or output). Should you copy to an output device or from an input device (i.e. COPY @PR TESTFILE or COPY TESTFILE @KI), you will create an error.

To summarize the principle involved; COPY is used to move data between two devices of differing types. Since a file can be either input or output, COPY can be used to move data between two files. But when dealing with the system devices, care must be given to the device type. To move data between two system devices of the same type, use either the FORCE or JOIN command, depending on what your application is. These commands also provide a display of the system devices and what type they are (e.g. input or output).

Mode 2

In this mode, also known as a "file copy", we are copying a:

* File to a file, with rename.
* File to a file, without rename.

Therefore, if a copy involves a device or more than one file at a time, it is not operating in this mode.

An example of copying a file to a file would be if you copied the file TEST1 from drive 0 to drive 1. The format would be:

```
COPY TEST1:0 TEST1:1
     or
COPY TEST1:0 :1
```

Notice the two different manners of issuing that command. In the first form, where the second filespec IS specified, you have the option of changing the filespec as you copy it. For example:

```
COPY TEST1:0 TEST2:1
```

would be perfectly legal. When the file TEST2 on drive 1 was examined, you would find that is it the same as the file TEST1 on drive 0.

Using the second form, in which the second filespec is not specified, but rather assumed to be unchanged from the first, you may not rename the file while you are copying it. For example:

```
COPY TEST1:0 :1
```

is going to create a file TEST1 on drive 1. Since copying without changing the filespec is a far more common occurrence than copying with the change, you will be using primarily this form.

Technical note: When you are using either of these forms you are engaging a file by file copy. By default, this uses the "big buffer" (all available memory) for the copy. In other words, it reads in as much of the file as available memory will allow before writing it back to the destination disk. This greatly increases the speed and efficiency of the copy.

You may specify the tiny buffer option (TINY parameter) and force COPY to copy only one sector at a time. This will slow down the copy, but it will force COPY to keep its buffer within the system overlay areas and out of user memory altogether.

If you are going to copy a file into a different area of the same disk, you MUST change the filename (because two files with the same name may not exist on the same disk). Therefore, the second form is only legal when moving files between two disks. You may, however, use the second form within a single drive when copying between two disks in that drive with the PROMPT parameter (e.g. single drive copy).

To copy a file between two disks in a single drive, specify the PROMPT parameter. COPY will then prompt you for the source, destination, and system disks as needed. Please pay close attention to the prompts and only insert the proper disks at the proper times.

Mode 3

In this mode, also known as a "wildmask copy", we are copying between a:

* Drive and a drive, using a wildmask.

Therefore, if your copy involves a device, only a single file, or you wish to rename the file during the COPY, you should be using one of the other modes.

An good example of copying a drive to a drive might be if you wanted to move all the files from the disk in drive 1 to the disk in drive 2. You would accomplish this by instructing DOSPLUS to move all files that match a certain wildmask from one drive to another. You would then simply make the wildmask general enough to incorporate ALL files.

In this area, wildmasks, DOSPLUS is VERY flexible. All these commands would accomplish the same thing:

```
COPY !:0 :1
(copy everything from drive 0 to drive 1)
COPY :0 :1 !
(copy from drive 0 to drive 1 using everything)
COPY FROM :0 TO :1 USING */*
(copy from drive 0 to drive 1 using everything)
COPY USING ! TO :1 FROM :0
(copy using everything to drive 1 from drive 0)
COPY TO :1 !:0
(copy to drive 1 everything from drive 0)
COPY :0 :1
(copy drive 0 to drive 1)
```

The phrase in parenthesis underneath the command example is there to help you get the feel of what each command is telling the system to do. You see, DOSPLUS evaluates each command line and determines what the user wanted to do.

It is during these wildmask copies that most of your parameters come into play. Let's cover them each now and what effect they have.

During one of these wildmask copies, if a file is invisible it will NOT be copied unless the INVIS parameter has been specified. This will become very important when setting up non-standard system disks with SYSGEN and COPY.

The filenames will NOT be displayed during a wildmask copy unless you use the ECHO parameter. Under most circumstances, you will want to see what files COPY is moving, and so use this parameter.

By using the MOD parameter, you may copy on the basis of whether or not the mod flag has been set. The mod flag (short for "modification flag") indicates that a file has been modified (e.g. opened and written to) since it was last copied or the disk was last backed up. This is displayed as a "+" in the directory entry of that file (see DIR). In some instances, you may wish to copy all files that have been modified from a particular disk. To do that, indicate "MOD=Y" in the parameter list during your wildmask copy. Then, when DOSPLUS encounters a file that matches the wildmask, it checks first to see that the mod flag is set before copying (e.g. has the file been modified?). If it is not set, in other words the file has not been modified, then DOSPLUS will skip that file and proceed.

The KILL parameter, when specified, will cause COPY to delete the file being copied from the source disk after it has been moved.

If you use the QUERY parameter, DOSPLUS will ask you if you wish to copy each file BEFORE it actually copies it. This can be useful in screening out one or two files that you don't want copied, but if you have a large number of files being copied it can get tedious to be prompted for each one.

If you use the OVER parameter, DOSPLUS will ask if you wish to overwrite a file (when it finds the same filespec on the destination drive) BEFORE it actually overwrites it. This prevents you from accidentally overwriting a file. It is often a good precaution to use this parameter.

When using either of these "prompting" parameters (i.e. QUERY or OVER), you will receive a prompt on the screen requesting action. In the case of QUERY, DOSPLUS will prompt "Copy ?" each time it finds a file matching the wildmask. In the case of OVER, DOSPLUS will prompt "Overwrite ?" each time it encounters a file on the destination disk with the same filespec as the one it is copying. If you press ENTER by itself at either of these prompts, it will have the same effect as typing "N" and pressing ENTER. The only way that action will be taken in either case is to type a "Y" and press ENTER.

When doing a wildmask copy, often you will have either a source or destination file that is password protected. You will not be able to copy or overwrite this file without a password. But since there may be several such files in any given copy, you cannot specify all the needed passwords. This is where the expanded use of the Disk Master Password in DOSPLUS and the SPW and DPW parameters come in. In DOSPLUS, you may always specify the Disk Master Password in place of a file password. That is why it is so important to assign passwords when formatting. If you will be copying in a situation where you may encounter protected files, COPY allows you to specify the source and destination Disk Master Passwords. Using the form "SPW=password" and "DPW=password", you can supply these in the parameter line. When COPY encounters any protected files, it will then use the supplied password to attempt to access it. You will need to avail yourself of this when using COPY to move the DOSPLUS utilities after a SYSGEN.

The NEW and OLD parameters are also for use during a wildmask copy. They provide a means of copying between two disks based on what files exist on which disk. For example, if you copied from drive 0 to drive 1 using the NEW parameter, all the files that existed on drive 0 (the source drive) that did not exist on drive 1 (the destination drive) would be copied (e.g. all the new files). To do the same thing with the OLD parameter would only copy those files that already existed on the destination drive (e.g. all the old files). NEW can be used to make certain that two disks contain the same files. OLD can be used as a convenient way to update programs from new masters.

Examples:

```
COPY FROM :0 TO :1 USING ! (INVIS,ECHO,OVER,SPW='PASS')
COPY :0 :1 ! (INVIS,ECHO,OVER,SPW='PASS')
COPY :0 :1 ! (I,E,O,SP='PASS')
COPY !:0 :1,I,E,O,SP='PASS'
```

All four of these commands will have the same effect. They will cause all files from drive 0 to be copied to drive 1. Invisible files will also be copied and DOSPLUS will NOT overwrite a file without first asking. The Source Disk Master Password is "PASS", in case any of the files being copied are protected.

```
COPY FROM @KI TO @DO
COPY TO @DO FROM @KI
COPY @KI @DO
```

These three commands all instruct DOSPLUS to copy all characters received from the keyboard to the display. As you would type in characters, they would be echoed to the screen, but would NOT be acted upon. Remember that you MUST copy FROM an input device TO an output device. To do otherwise will cause an error.

```
COPY MYFILE/BAS.PASSLOG:0 YOURFILE/BAS:2
```

This example will copy the file MYFILE/BAS from drive 0 to drive 2. In the process, it will rename it to YOURFILE/BAS. On drive 0, the file is protected and uses the password "PASSLOG", so this is specified in the source filespec.

```
COPY THISFILE/CMD:0 THATFILE/CMD.CHECK:1
```

In this example, we have reversed the protection situation. This time, the destination file is password protected and the password had to be included with it. This example assumes that the destination file is already existing, but if it were not, COPY would create it. Because COPY clones attributes when it creates files, if it had to create the destination file it would bear the same password and protection status as the source file.

```
COPY SHORT:1 :0
```

This command will move the file SHORT from drive 1 to drive 0. The second filespec is assumed to be SHORT as well, because only the drivespec was specified.

Finally:

When using wildmask copies that affect a great number of files, please use the QUERY, OVER, and ECHO parameters if there is any doubt at all as to whether or not your mask is too general. Don't wait until it is too late to discover that you have set a mask that allows too many files to be moved and potentially corrupts valuable data.

When using the PROMPT parameter, you may not be copying with a device or wildmask copy. It must be a file copy. This means that single drive copies must be done one file at a time.

COPY will duplicate the attributes of any file that it copies. That is, the file it creates will have the exact same attributes (i.e. Logical Record Length, Protection levels, etc.) as the file it is copying. This does not apply to device copies.

If you have created any files with the system attribute set on them (please note that this is not a normal user option, you must have done this manually), COPY would normally not copy these files during a wildmask copy. We have included an extra parameter to help with that, though. COPY has a parameter called SYSTEM that works like INVIS. If you specify SYSTEM, COPY will simply consider system files as well as user files during the copy. This may not be used to copy the DOSPLUS system files. It is there to aid some users in very special cases.

Some of the parameters may not apply in all cases. Please use common sense. If you specify OVER and NEW together, it will never prompt you to overwrite. If the file exists, it won't be copied, therefore how could it be overwritten? The same is true for QUERY and ECHO. You will never see the filename twice. If DOSPLUS shows you the filename to ask you whether or not to copy, why should it show you the filename again when you say yes? These are not errors, but rather the command has fairly sophisticated error trapping.

CREATE

This command allows you to create disk files and pre-allocate their space.

CREATE filespec (param=exp...)

filespec is the standard DOSPLUS file specification that informs CREATE of the name of the file you wish to create.

(param=exp...) is the optional parameter indicating what further action you might wish CREATE to take past simply creating a directory entry.

Your parameters are:

DATA=value	Fill data (one or two bytes).
GRANS=value	Number of grans.
KEEP=switch	Sets keep flag (non-shrinkable attribute).
KILO=value	Number of kilobytes.
LRL=value	Logical record length.
SIZE=value	Number of records.
VERIFY=switch	Verify disk space after creation.

Abbreviations:

DATA	D
GRANS	G
KEEP	K
KILO	KI
LRL	L
SIZE	S
VERIFY	V

By using the CREATE command, you may create and pre-allocate (set aside space for) a disk file. This is different than normal operation in which the file has space allocated to it dynamically (as it is needed). Normally, whenever data is written into the file, if more room is needed, the system will assign the file more disk space.

When you create a file, you have the option of setting the KEEP parameter. This affects the allocation/de-allocation of a file still further. Normally, even if you use CREATE to create the file, the space may be re-claimed dynamically under certain circumstances (i.e. if the file is closed after data is written to it sequentially).

Therefore, by using CREATE, you have brought the file into existence. Space is still allocated and de-allocated dynamically unless you use the KEEP parameter. The KEEP parameter tells the system to never DE-ALLOCATE space from that file. The file may still be extended dynamically, but DOSPLUS will never reclaim space from it.

If you attempt to create a file that already exists, CREATE will inform you that the file DOES already exist and abort. If you do not specify the drivespec when giving CREATE the filespec to be created, then CREATE will attempt to create it on the first available drive. If there is insufficient space on a drive to hold the file, then you will receive a error message informing you that the disk space is full and it will allocate as much space as WAS available to that file.

In such a case, the file will have been created and space allocated to it, but the file will contain no records because it was never actually closed. The date in the directory will also not be set.

This is, of course, assuming that you have elected to pre-allocate the disk file in addition to creating it and instructed CREATE to do so. If you do not tell CREATE to pre-allocate disk space, this command will simply create the directory entry. The file will have no space allocated to it and will not take up any space on the disk. The system will allocate space to the file the first time that you write to that file.

Using CREATE to pre-allocate data files can greatly increase the speed of data handling. Because the file already exists and has its space allocated, time will not have to be taken to do it dynamically. Also, depending on the disk, the file will tend to be less segmented. The fewer segments the file is in, the less that the drive has to move the head around when reading in the data. It also gives you the very important option of filling the file with a specified data pattern and then verifying the file's disk area before beginning operations.

Pre-allocation

To determine the size of the file when you wish to pre-allocate, you have three options. You may:

(1) Allocate by the number of records with SIZE.
(2) Allocate by the number of granules with GRANS.
(3) Allocate by the number of total kilobytes with KILO.

When allocating by number of records, then the logical record length has a great bearing on file size. The logical record length of a file in DOSPLUS does little more than affect how the directory entry will look. You can choose any logical record length between 1 and 256, inclusive. The DIR command displays both the number or sectors in a file and the number of logical records (see DIR). Only by having the proper logical record length for each file will the information in the "number of records" column be really useful.

However, DOSPLUS does allow you to access files with a different logical record length than they had when they were opened. You may find this of some use.

You will adjust the logical record length of the files you create with the LRL parameter. For example, LRL=128, would be a logical record length of 128.

When you are pre-allocating a file by records, you specify the number of records you desire in that file by using the SIZE parameter. Simply set SIZE equal to however many records you anticipate. The actual physical size of the disk file will be equal to the number of records specified times the logical record length used. Of course, if the logical record length happened to be 256, then SIZE would equal the number of sectors.

Allocating a file by "granule" assumes that you have at least a passing familiarity with what a granule is. A granule is defined as the smallest unit of disk allocation. A disk is made up of sectors. Each sector is 256 bytes long. These sectors are grouped into tracks. The tracks are concentric circles of data on the disk. Each track has a pre-defined number of sectors on it. As data is written to the disk, space must be made available to the file.

If DOSPLUS were to allocate space to a file one sector at a time, the result would be very slow. The drive would constantly be stepping out to the directory track to ascertain where the next free sector was and assign it to the file to which you are. Therefore, DOSPLUS will allocate space several sectors at a time. This unit of allocation is called a granule. On a standard 5 inch single sided floppy disk, a granule is made up of 5 sectors single density and 6 sectors double density. There are a total of two granules (10 sectors) on each track in single density, or three granules (18 sectors) in double density.

You may also allocate a file by specifying how many granules that you wish the file to contain. You will adjust this value via the GRAN parameter. Granules are normally invisible to the user. The only place in the entire DOSPLUS system that the number of free granules on a disk is displayed is from the DIRCHECK utility (see DIRCHECK). When you specify the number of granules, the system will calculate how many records to allocate and act accordingly. It does not matter what the logical record length is, CREATE will adjust for it. There will ALWAYS be the number of granules in the file that you have specified regardless of whether or not you specify a logical record length of less than 256 (unless, of course, you try to allocate more space than is on the disk).

And finally, pre-allocating by kilobytes. Allocating a file by kilobytes is simple. Just determine how big the file should be and inform the system. This figure is expressed in kilobytes, so be careful not to ask for more than you desire. For example, "KILO=100" is asking for 100,000 bytes, not 100. Again, it will not matter what the logical record length is. CREATE will allocate as many records as it needs to to come up to the specified amount of total disk space.

All of the values that we have just talked about (SIZE, GRANS, and KILO) MUST be entered in the command line as positive integers.

Verification of data

To use CREATE to verify a file's disk space, you have two parameters available. These are DATA and VERIFY. Both of these parameters are invalid unless you have pre-allocated some space to the file in the CREATE statement.

To simply fill a file's disk space with a specified data pattern, you only need to use the DATA parameter. This will cause CREATE to write to every sector of a file's disk space with whatever one or two byte value you specify.

When you are specifying the DATA parameter, if you only specify a one byte value, then CREATE simply duplicates the first byte into the second when filling. Which is to say that CREATE always fills with a two byte value. It simply allows you to only specify one byte if you want the same value in each.

What this means is that the values 6C00H and 006CH will react very differently. In the first case, CREATE will fill with a data pattern of "6C006C006C00" where the second will use a pattern of "6C6C6C6C6C6C". A leading zero is ignored, a trailing one is not. When using a decimal value, anything between 0 and 65535 is valid.

To have CREATE verify the file's disk space after it has filled it, just include the VERIFY parameter in the parameter list when you issue the command. CREATE will create the file and pre-allocate the space, fill the file with the specified data, and then read each record to make certain that the area is readable.

You do not have to fill a file to verify it, just pre-allocate space. If you include the VERIFY parameter without the DATA parameter, the file's area will still be read. However, in most cases, you will probably want to clean out the area to be used or perhaps fill it with a more critical data pattern for an extra measure of verification.

Examples:

```
CREATE NEWDAT:0 (LRL=128,SIZE=100)
CREATE NEWDAT:0 (L=128,S=100)
CREATE NEWDAT:0,L=128,S=100
```

These three commands will all have the same effect. They will create the file named NEWDAT on Drive 0 with a logical record length of 128 and pre-allocate 100 records to it. It will not write any data to the file, nor will it verify the file's disk space.

```
CREATE PAYROLL:B (DATA=229,GRANS=12,VERIFY)
CREATE PAYROLL:B (D=229,G=12,V)
CREATE PAYROLL:B,D=229,G=12,V
```

These three commands will also perform the same function. They will create the file PAYROLL on the drive B with a logical record length of 256. They will pre-allocate 12 granules of disk space to the file and then fill each sector with a data pattern of 229 decimal (E5 hex) and then verify each sector to make certain that the space was readable.

```
CREATE DATAFILE/DAT (DATA=108,KILO=100,KEEP)
CREATE DATAFILE/DAT (D=108,KILO=100,K)
CREATE DATAFILE/DAT,D=108,KILO=100,K
```

These three commands are equivalent. They will each cause the system to attempt to create a file called DATAFILE/DAT on the first available disk drive. It will create this file with a logical record length of 256 (because nothing else was specified) and pre-allocate 100K to it. Then the system will fill each sector with a data pattern of 108 decimal (6C hex). It will NOT verify these. Finally, it will set the KEEP parameter, instructing the system never to de-allocate disk space from that file.

```
CREATE BADFILE (DATA=108)
```

This is an example of an illegal command. You have specified a data pattern without pre-allocating any disk space to the file. DOSPLUS will simply ignore the DATA parameter, create the file, and proceed.

```
CREATE WORSEFIL (VERIFY)
```

This is another example of an incorrect command. You have instructed CREATE to verify a file that you have not pre-allocated any space for.

Finally:

The most important thing to remember when using CREATE is, don't allocate more space than you have. If there is only 90K free on a disk, don't specify "KILO=100" when pre-allocating disk space. If you DO receive an error, don't panic. That is one of the reasons for CREATE, so that you may first test to see if you have the space for a file and then, if you wish, to test every record of the file's disk space.

Also keep in mind that CREATE will NOT work if a file already exists on the disk with the same filespec as the one you are creating. This is for your protection, so that you don't accidentally destroy all data in a file.

DATE

This parameter allows you to display or set the current system date.

```
DATE
DATE mm/dd/yy or mm/dd/yyyy
DATE CAL
DATE CAL mm/yy or mm/yyyy
```

To display the currently set system date, type:

DATE

and press ENTER. The date will be displayed in the following format:

Thu - Jan 27, 1983 - 27

The first item is the day of the week, followed by the date, and the last number is the day of the year (1-365). When using the DATE command to set the system date, the date can be specified in a variety of ways. Allowable separators are any non-numeric characters. This flexibility allows you to specify the date in whatever format is most comfortable to you. Also, DATE will accept either a two or four digit year value when accepting a date. All this allows you to enter the date as you are used to with TRSDOS and later as you become more familiar with DOSPLUS, move to the more convenient formats. DOSPLUS will accept dates from 1900 to 1999. Micro-Systems will issue free patches to owners in the year 2000 to change the base year.

Optionally, you may specify the CAL parameter and receive a calendar display for the month that contains the current date. Still further, you may request a calendar for another month in an entirely different year (between 1979 and 2079) without affecting the current date. The two forms are:

DATE CAL

displays the month for the current date. Also:

DATE CAL 01/2000

displays a calendar for January of the year 2000.

Example:

```
DATE 01/27/83
DATE 01/27/1983
DATE 1.27.83
```

This command will set the system date to January 27th, 1983.

DEBUG

This command will engage or disengage the system debugger (memory monitor).

==========

DEBUG [switch]

[switch] is the optional ON or OFF condition.

==========

DEBUG is a powerful disk based memory monitor. With it you can examine any memory location in RAM or any CPU register. You may also change the content of a RAM location or register.

Unlike the other DOSPLUS commands, when you enable DEBUG you will not see any noticeable change on the screen. This is because DEBUG is transparent to the execution of your program and is only entered when called. There are two ways to call DEBUG when it has been enabled. They are:

(1) Pressing CLEAR-BREAK at any time.

(2) Automatically after a machine language program has been loaded and before the first instruction has been executed.

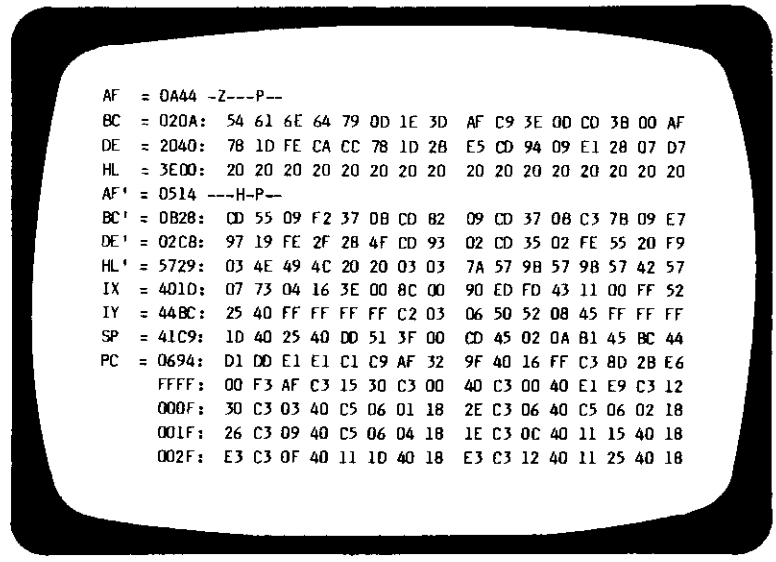
(3) If an abort due to error occurs, DOSPLUS will exit to DEBUG instead of the DOS command mode.

Once DEBUG is called, you have the following commands:

Command	Operation performed
A	Set ASCII/Graphic display mode
C	Instruction/Call step
Daaaa	Set memory display address to aaaa
Gaaaa,bbbb,cccc	Go to address aaaa, with breakpoints optionally set at bbbb and cccc
H	Set hexadecimal display mode
I	Single step next instruction
Maaaa<space bar>	Set memory modification mode starting at address aaaa (optional). ENTER records change and ends, space bar records change and moves to next address.
O,aaaa,bbbb	Exit to DOSPLUS (DEBUG still engaged) with breakpoints optionally set at aaaa and bbbb.
Rpr aaaa	Alter register pair pr to value aaaa. Space between register pair and value is required.
S	Set full screen memory display mode
U	Set dynamic display update mode
X	Set register examine mode
; (Semi colon)	Display next memory page
- (Dash)	Display previous memory page

The following is an example of a DEBUG register examine mode display (X):

Insert DEBUG fig 1



The general format is:

The register pairs are indicated down the left hand side of the screen, with the standard registers listed first, and the prime registers following. Last are listed the special registers (IX, IY, SP, and PC).

The AF register contains the system flags. They are all set in the example above. They are:

S - Sign flag
Z - Zero flag
H - Half-carry flag
P - Parity flag
N - Overflow flag
C - Carry flag

These are indicative of system status after an operation, and of limited usefulness to anyone save the machine language programmer.

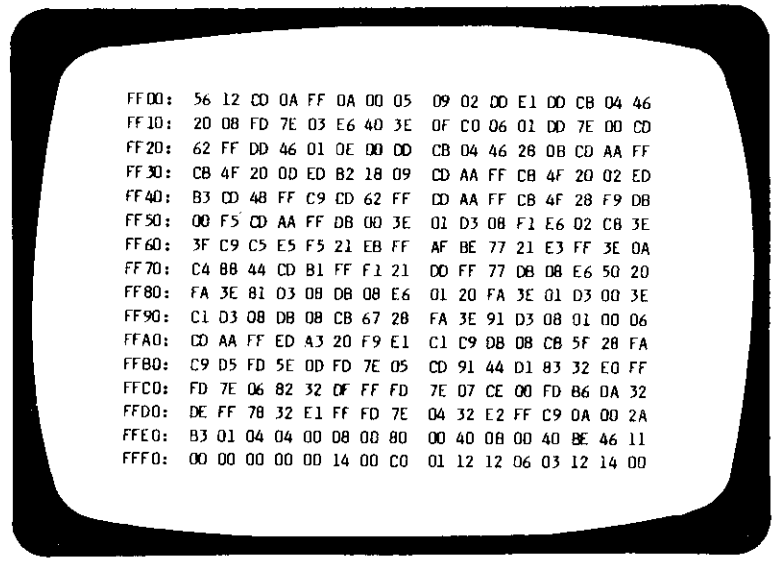
The rest of the registers all display the contents of the register, and then immediately to the right of the register, it displays the sixteen bytes of memory that the contents point to.

In the case of the stack pointer (SP), this will display to you what is on the stack. In the case of the program counter (PC), it will displayed the next instruction to be executed.

The last four lines are simply displaying memory. You can alter these to display any desired address.

The following is an example of a full screen memory display mode (S):

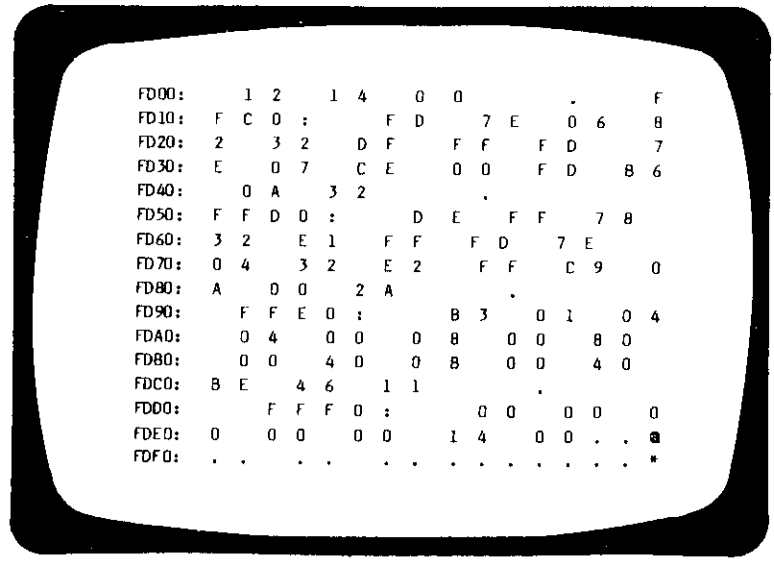
Insert DEBUG fig 2



The left hand column contains the hexadecimal memory address currently being displayed. The memory is displayed in sixteen byte rows for one 256 byte "page".

The following is an example of the ASCII/graphics display mode (A):

Insert DEBUG fig 3



The left hand column contains the address being displayed. To the right is the ASCII translation of the memory contents. Unprintable characters are represented as periods (".").

DIR

This command will display a disk's file directory.

==========

DIR [FROM] drivespec [TO] device/file [USING] wildmask (param=exp...)

drivespec is the optional drivespec indicating which disk's file directory to display.

device/file is the optional output device or file.

wildmask is the optional wildmask to restrict DIR to a certain group or class of files.

(param=exp...) is the optional action parameter that indicates what type of directory you want to see.

Your parameters are:

SYSTEM=switch	Display system files as well as standard entries.
INVIS=switch	Display both visible and invisible user files.
KILL=switch	Display any deleted files not yet wiped from the directory or overwritten by an active file.
ALPHA=switch	Display files in alphabetical order.
MOD=switch	Display files based on status of MOD flag.

Abbreviations:

SYSTEM	S
INVIS	I
KILL	K
ALPHA	A
MOD	M

==========

The DIR command is used to display a disk's file directory (hence the name "dir"). A disk's file directory is a list of files residing on any given disk with a host of accompanying information for each file. This command will display all available information regarding a disk file. It will detail the filename and extension, it will indicate how large the file is, whether the file is segmented or not, whether or not the file has password protection, and what level this protection is set at. It will give you a file's logical record length, it will tell you if the file has been modified since last copied or backed up, and it will tell you the date of the file's last update.

The DIR command has two basic types of function: standard and global. In the standard DIR, you receive only the file directory for the drive that you request. In the global form, engaged by using a wildmask, you will receive a file directory of all mounted disk drives. Used in conjunction with a specific enough wildmask, this can be very useful for ascertaining where in the system a file is currently located.

When you request a file directory, you may also specify the output file or device. If you do not specify a file or device when you issue the DIR command, the file directory will be displayed on the screen. The display (i.e. @DO) is the default device. This allows you to output the file directory to the printer, a disk file, or wherever it may be required.

The simplest form of DIR is:

```
DIR
```

This will display a directory of all visible user files on the system drive The next simplest form would be:

```
DIR :1
```

This accomplishes the same effect, but restricts itself to those visible user files on drive 1.

The various parameters control which files are displayed and in what order. They are:

SYSTEM. If this parameter is specified, system as well as user files will be displayed. This has nothing to do with the /SYS extension, but rather the internal system attribute being set or reset.

INVIS. This has essentially the same effect for invisible files. These files will not display in a standard directory. To include them, include this parameter. As with system files, invisible files will be flagged as such in the display (see below).

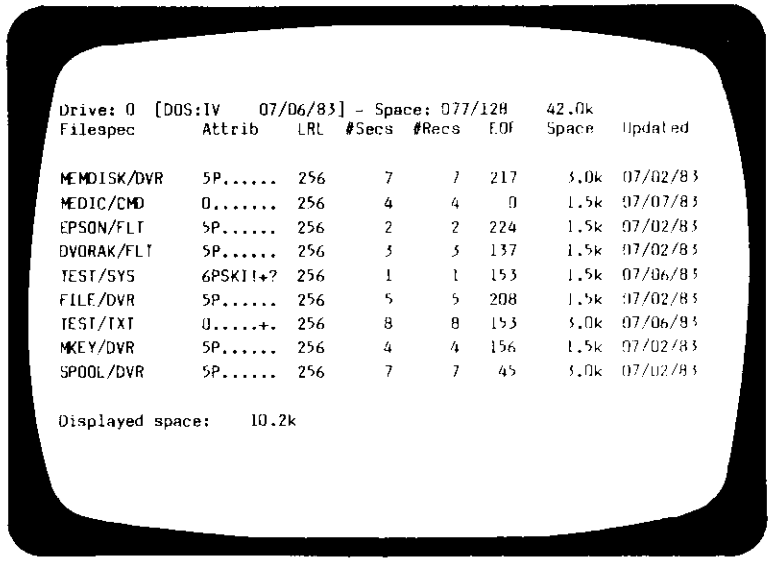
KILL. If included, this parameter will include KILLed files that are still in the directory. They will be flagged in the display with a "K". This will allow you, especially in conjunction with a wildmask, to determine whether or not a file that has been killed is there for the RESTORE utility (see RESTORE) to attempt to bring back.

ALPHA. This parameter allows you to have the output of the directory in alphabetical order.

MOD. This allows the directory display to be based upon the mod flag. You can see a directory of all the modified, or by specifying MOD=N, all the unmodified files.

The file directory display

When you request a file directory, your output should look something like this:



```
Drive: 0  [DOS:IV    07/06/83] - Space: 077/128    42.0k
Filespec       Attrib    LRL  #Secs  #Recs  EOF   Space   Updated

MEMDISK/DVR    SP.....   256      7      7   217    3.0k  07/02/83
MEDIC/CMD      O.....    256      4      4     0    1.5k  07/07/83
EPSON/FLT      SP.....   256      2      2   224    1.5k  07/02/83
DVORAK/FLT     SP.....   256      3      3   137    1.5k  07/02/83
TEST/SYS       6PSKII+?  256      1      1   153    1.5k  07/06/83
FILE/DVR       SP.....   256      5      5   208    1.5k  07/02/83
TEST/TXT       O.....+.  256      8      8   153    3.0k  07/06/83
MKEY/DVR       SP.....   256      4      4   156    1.5k  07/02/83
SPOOL/DVR      SP.....   256      7      7    45    3.0k  07/02/83

Displayed space:    10.2k
```

The display will continue for one full screenfull and then pause waiting for you to press ENTER, SPACE, or BREAK. Pressing ENTER will display the next screenfull. Pressing SPACE will display the next single file entry. Pressing BREAK will abort the command. You may also press SPACE to pause during the output and BREAK to abort.

At the bottom of the directory display is the "Displayed space:" indicator. This will tell you how much space all of the files displayed in the current DIR occupy. This can be useful when combined with wildmasks (i.e. DIR */CMD:0 to ascertain the amount of disk space used by all of the visible files with a /CMD extension).

The directory display will pause only when there is no output device specified. At that time, DOSPLUS will make the assumption that you are send the data to the display and will implement the pause. Should you execute a "DIR :0 @PR", though, there will be no pause at screenfull because of the "@PR" output device. To obtain DIR to the video without an automatic pause, use "DIR :0 @DO". An output device is specified, so there is no pause. It just so happens that the output device is the video again. This may be used with piping or some other application in which pausing the DIR display is not the wisest course of action. As always, SPACE will pause the DIR output.

The simplest way to explain the directory output is to divide it up into three lines. These three lines will be present for all disks that are directoried.

The first line of display will be the free space summary for that drive. This line will give you, reading from left to right, the following information:

* The drive name.
* The disk name.
* The creation date.
* The status of directory entries. (free/total).
* The amount of free space in kilobytes.

The drive name (in our example, this is drive 1). This is the current drive specification for the drive being directoried. The drivespec is, of course, the two character designation (preceded by a colon ":") by which you address the disk drive.

The disk name (in our example, "DOS IV"). At time of format, both the FORMAT and RFORMAT utilities will ask you what you wish to name the disk. This name is stored on the directory and is displayed whenever you do a CAT, DIR, FREE, MAP, or DIRCHECK. Usually, you will use this name to reflect the contents of the disk (i.e. "Profile" or "Gen Led"). This can be up to eight characters in length.

The creation date (in our example, "07/04/83"). At the time of format or backup, the date is assigned to the disk. This is displayed here.

The directory entry status (in our example this was "080/128"). DOSPLUS only allows a certain number of entries per directory (256 maximum). Once you have filled this up, you may not create any new files on that disk regardless of its free space because there is no room to put it in the directory. This does not necessarily mean that you cannot extend existing files. As long as they don't need to create an extended entry and there IS some free space on the disk, you may extend the existing files.

The directory entry status display tells you how many file entry positions you have free (i.e. how many files remain) and how many possible entries there were for that drive. The second number will never change, but when the first number reaches 0, the directory is full. Therefore, in our example, it is informing us that we have used 80 out of a possible 128 files on this particular disk.

The free space in kilobytes (in our example, 67.5k). This will give you, in brief, the free space remaining on that drive. This figure will be expressed in kilobytes and will be rounded to one decimal point.

The next line of the directory display is the header line. The directory entries themselves are divided into columns of information. This header line titles each column and identifies it. We will list the columns here and explain them one at a time when we cover the directory entries.

* Filespec.
* File attributes (Attrib).
* Logical record length (LRL).
* Number of physical sectors (#Secs).
* Number of records (#Recs).
* End of file (EOF)
* Total size in kilobytes (Space).
* Date last updated.

Following this line are the directory entry lines themselves. If there are no files to display (i.e. no visible files and no invisible parameter, no files matching wildmask, or whatever the reason), there will be none of these. There WILL be as many of these display lines as there are entries to show. Let's address now the display entry line one item at a time:

The filename. This first piece of information will consist of the file's name AND extension. For further detail as to what are legitimate filenames and extensions, consult the operations section under "File and Device Specifications". If you don't see the filename that you expected, perhaps you didn't use the INVIS, SYSTEM, or KILL parameter that was needed. If you wish this display to be alphabetized, remember to use the ALPHA parameter.

The file attributes. This column, underneath the word "Attrib", contains seven items of great importance regarding the disk file. These are called the file's attributes. For more detailed explanation of the attributes and when to use them, see the command ATTRIB. At this time we will only discuss them such as they affect the operation of DIR.

(1) The file's protection level. This first character will never be a period. A file always has a protection level. Even if it is simply "0", it is still a valid protection level. This position will contain a number from 0 to 7 that corresponds with one of the seven protection levels listed in ATTRIB. Note that in the example, the file TRAP/CMD has a 0 in this position. This indicates no protection at all. The other files have either a 5 or 6 in this position indicating that they have a protection level assigned them.

(2) The password flag. If a file has a password set for it, either access or update, this second character will be a "P". Otherwise, a period will be displayed here. The differences between access and update passwords are also covered in ATTRIB. This flag simply lets you know that one or the other or both are set.

(3) The system file flag. If a file carries the system attribute, then an "S" will be displayed in the third character. The files SYS0/SYS and SYS2/SYS in our example illustrate this. This is not an attribute that can be set from DOS. It is reserved for the DOSPLUS system files.

(4) The kill flag. If a file is deleted (non-active), then a "K" will appear in the fourth character. DOSPLUS does not remove the filespec from the directory when it is killed unless you instruct it to do so with PROT. This leaves open the possibility of using the RESTORE utility to bring back a file that was accidentally killed. This flag will appear in the attribute line of all "dead" files during a directory display.

(5) The invisible flag. If a file is invisible, normally it will not be seen in the directory. If you use the INVIS parameter, though, these files will be included. At that point, you would have no way of discerning which files are normally visible and which are normally invisible. That is, no way without this flag. If there is an "I" in the fifth character, you know that file is normally invisible.

(6) The keep flag. If a file has the KEEP flag set for it, the DOS will never de-allocate space from it. It may extend it, but it will never shrink. This parameter is set by and documented under ATTRIB and CREATE. If a file has the keep flag set, and exclamation mark will appear in the sixth character. The file FILE/DVR in our example illustrates this.

(7) The mod flag. This flag indicates that a file has been modified since the last time that file was copied or the disk it resides on was backed up. If set, it will appear as a plus sign (+) is position 7. This serves as a warning to you that a file exists with unique data in it (e.g. if you lose this disk, you have potentially no other copies of this data). This parameter is extremely useful when using a backup by file method of preserving your data inasmuch as COPY allows you to copy based on this mod flag (see COPY).

(8) The close flag. This flag indicates that a file has been opened and not closed. If set, a question mark (?) will appear in position 8. Under this version of DOSPLUS, this does not cause any operational difference, but may in future implementations. You should always close files when you are through with them. To reset this flag, use the ATTRIB command (see ATTRIB).

The logical record length. This figure is essentially for display only. This will be expressed as a numeric value between 1 and 256. If the logical record length is less than 256, the "Number of records" display will be a larger number than the "Number of sectors".

The number of records. This figure will give you the number of records currently in that file. This may or may not the same as the "Number of sectors" display, depending on the logical record length of the file. This figure will be the actual number of records written to the file.

The number of sectors. This figure will give you the number of actual disk sectors currently written to by that file. This will NOT reflect the number of sectors allocated. DOSPLUS allocates by granule (for an explanation of granule allocation, see the library command CREATE or the technical manual under diskette formats). Since a granule is comprised of several sectors, there will usually be more sectors allocated than are currently written to.

The end of file. This figure will indicate how many bytes the file takes up in the last sector allocated to it. This will be one higher than the actual number of the byte, since we start numbering bytes at 00H.

The total size in kilobytes. This will give you a figure to indicate what the total size of the file is. This figure will be given rounded off to the first decimal point (i.e. to the nearest hundred bytes). For example, if a file was 1220 bytes long, the directory entry would indicate 1.2K. If it was 1270 bytes long, it would indicate 1.3K. This figure represents the amount of space ALLOCATED. This is different from the number of records and number of sectors parameters. Those two indicate the number actually written, while this indicates not only the space already used, but also the space that has already been allocated to be used. For that reason, the "number of sectors * 256" formula may not always agree with this figure.

The date last updated. As you know, DOSPLUS maintains a "system date". On power-up, the DOS will prompt you for the date (unless you have disabled this question by using the SYSTEM command). It will preserve this date in memory and any time that it would normally ask you for the date (FORMAT, BACKUP, etc.), it will skip that prompt. One additional feature of having the date set is this display in the directory. Each time that you access a file, the current system date is updated to that file's directory and displayed via the DIR command. If this system date is not set, the date "--/--/--" will be used.

Using DIR

You may call DIR from BASIC without problems unless you wish to use the "Alpha" function for an alphabetical DIR. This cannot be used from within a BASIC program because when you ask for a sorted directory, the memory required to do the sort expands past the limits of BASIC's overlay area for DOS commands and it will corrupt your program.

When using DIR, if you wish to specify an output device/file and you have NOT specified a source device/file, you must use the delimiter "TO" to indicate data flow. This would occur if you were going to get a printout of the file catalogs for all available drives. To type:

```
DIR @PR
```

would produce an error, since "@PR" is in the source field position and "@PR" is not a valid drivespec. However:

```
DIR TO @PR
```

would work just fine. This does not apply if you are using a source drivespec, because then the output device/file is in the proper location. For example:

```
DIR :1 @PR
```

will operate properly. "@PR" is in the proper position for an output device/file and all will work well.

The only exception to this rule is the wildmask. If the wildmask contains a wildcard character (i.e. "?", "*", or "!"), then the DOS will move that to the wildmask position for you and scan the rest of the line in normal order. For instance:

```
DIR :0 USING */BAS
```

is the same thing as:

```
DIR */BAS :0
```

The system will move the "*/BAS" to the wildmask field and then pick up ":0" as the source drivespec. This does NOT apply if the wildmask doesn't contain any wildcard characters. IF you were to specify a wildmask without wildcard characters, then only files EXACTLY matching the wildmask would be displayed. However, with no wildcard characters to signal DOSPLUS that this is indeed a wildmask, it will simply be regarded as an invalid source drivespec. For example:

```
DIR TEST/DAT
```

will produce an error, while:

```
DIR USING TEST/DAT
```

will not. If you follow these rules of order, you should never get an error while using DIR. The best rule of thumb is, if you can't remember whether or not the delimiter is required, include it. It never hurts to have it in the command line, but sometimes it will cost you to omit it.

If the switch is not specified in the parameter list, it defaults to "off". For example, if you do NOT specify the SYSTEM option in the parameter field, it will default to SYSTEM=N (e.g. no system files will be included in the file directory). On the other hand, because of this, the simple inclusion of the option in the parameter field is sufficient to engage it. For example:

```
DIR :0 (SYSTEM=Y)
```

and :

```
DIR :0 (S)
```

are equivalent commands. This applies to all of the optional parameters on DIR. The simple inclusion of the name of the option is sufficient to engage it and the exclusion of the name will cause the option NOT to be in effect.

One other important area to keep in mind is the accidental overwriting of files. The correct form of the command is:

```
DIR <source> <destination> <wildmask> <parameters>
```

If you wish to only specify the source drivespec and a wildmask (e.g. you wish to let the destination default to the screen), then you must either have a wildcard character in the wildmask or use the USING delimiter. There is no way around this.

A wildmask in the destination field that does not contain any wildcard characters will be regarded as the output filespec and the file directory will be placed into that file. This can destroy the very file that you were seeking to locate.

Examples:

```
DIR :0 (SYSTEM=Y,INVIS=Y,KILL=Y)
DIR :0 (SYSTEM,INVIS,KILL)
DIR :0 (S,I,K)
DIR :0,S,I,K
```

All four of these command lines will perform the same task. They will display a file directory of the disk in drive :0. The catalog will include all filespecs, whether system, invisible, active or deleted.

```
DIR USING PER/DAT
```

This will search the directory of all available drives and printout a file directory for any drive having the file PER/DAT on it. This is an example of the method that would be used to locate all occurrences of the file.

```
DIR */CMD TO @PR
```

This example will scan all drives and printout the filespecs of any files that have the extension /CMD.

```
DIR :1 (INVIS=Y,ALPHA)
DIR :1 (I,A)
DIR :1,I,A
```

These three commands are all equivalent. They will display, in alphabetical order, all the user files, both visible and invisible, located on the disk in drive 1.

```
DIR :1 (INVIS=Y,MOD=N)
DIR :1 (I,M=N)
DIR :1,I,M=N
```

These three commands will display the file directory for the disk in drive ":1". It will include invisible files and will exclude any files that have the mod flag set.

DO

This command allows you to begin execution of a command chaining file (sometimes called "DO files").

==========

DO filespec (param=exp...)

filespec is the standard DOSPLUS file specification that indicates what file the commands to be executed are stored in.

(param=exp...) is the optional action parameter that modifies the operation of the command.

Your parameters are:

BREAK=switch	Break key enable/disable.
HIGH$=value	Set high memory before beginning DO execution.

Abbreviations:

BREAK	B
HIGH$	H

==========

The DO command allows you to execute, from a file on the disk, sequences of commands that you wish the system to accept exactly as if typed from the keyboard. This can be useful in the case of command sequences that will be repeated often or the case of startup procedures for "turn-key" programs.

These commands may be DOSPLUS library commands, the name of an applications program you wish to execute, or anything that you might normally enter from the DOS command mode.

When DO reaches the end of a list of commands, it will return control to DOSPLUS. The "DOS PLUS" prompt will be redisplayed along with the cursor. Commands may be entered as soon as the control is returned to the keyboard.

BREAK. This parameter allows you to set up whether the break key may be used to abort the DO file at a PAUSE statement (see PAUSE). Normally, you may abort a DO and return control to whatever program (DOS or BASIC) happens to have it whenever DO has stopped at a PAUSE statement and is requesting you to press a key. Sometimes, this is not desirable. In those cases, use this parameter to turn the break key "off" (i.e. BREAK=N). This only affects the break key within DO. It will function normally when DO has finished.

If you wish to create a "non-breakable" DO file, use the non-breakable AUTO option (see AUTO) and then when calling your DO file, turn the BREAK key off. The user will have to execute all the way through the file before gaining control of the DOS.

HIGH$. This parameter allows you to set high memory before starting a DO file and thus control where in memory DO will reside. DO uses 288 bytes of high memory (256 bytes for an I/O buffer, 32 bytes for a DCB). This area will be located starting at the top of memory and proceeding downward. By adjusting the top of memory pointer, you may control where DO resides. If you have applications programs in high memory that do not protect themselves by adjusting this pointer, DO may overwrite them unless you make provision for it. Please note that none of the DOSPLUS drivers exhibit this problem.

Also, it was stated earlier that DO used 288 bytes of high memory. This is true. But if DO has been used once already, it will reuse the same 288 bytes.

You may adjust the top of memory pointer in two ways. You may use the HIGH parameter on the SYSTEM command (see SYSTEM) and then call DO, or you may use the HIGH$ parameter when actually calling DO. The function is the same. When using the HIGH$ parameter, set it equal to an address, either in decimal or hex, that represents the address in memory that you do NOT want DO to use above. This should be one byte lower than the starting address of the block of memory you are seeking to protect (i.e. HIGH$=7FFFH or HIGH$=32767 would serve to protect from 8000H up). The pointer value will be adjusted before DO begins and will not be altered when DO is complete.

Applications of DO

There are many areas that DO used in, but perhaps the most common are:

(1) Startup for applications programs.

(2) Routine sets of often used instructions.

(3) Installing patches to the system.

(4) Automatic operation of programs.

In the first, startup for applications programs, DO is perhaps most useful. Many of your applications programs, especially those written in BASIC, will require that you set certain library commands or in some other way interface to the system BEFORE running the desired application program. DO will allow you to enter all needed commands and chain into your program without operator intervention.

Because DO allows you to do this automatically without forcing the novice user or non-technical operator to remember DOS syntax, your programs and computer system can seem more "user friendly". Simply set up a file with BUILD (see BUILD) that contains the needed sequence of commands. Remember, enter these EXACTLY as you would if you were entering them in the DOS command mode. Then you would have the last statement in your DO file call your program (i.e. BASIC MENU/BAS).

The most convenient method of starting this file executing is to set the DO command on an AUTO statement. For example, if we created a file to adjust FORMS and then load our BASIC file, we might place the following statements in the file STARTUP/TXT:

```
FORMS (PAGE=66,LINES=60)
BASIC PAYROLL(F=5)
```

Then we would enter the statement:

```
AUTO DO STARTUP
```

Whenever we re-booted the system, the statement "DO STARTUP" would appear and the statements we had stored there would be executed. We would see them as they were being executed. For more specific information on setting AUTO commands, see the library command AUTO.

For the second application, routine execution of sets of often used instructions, perhaps the best example would be in backing up your data disk after using some application program. You would set up a DO file with all the needed statements to call in BACKUP and copy the disk.

By using comment lines for instructions (see BUILD) and the PAUSE command to stop the DO file whenever it is necessary to swap diskettes (see PAUSE), you can make the entire procedure automatic. The advantages of this are two-fold. First, it makes it easier for you to backup the disks yourself because you're not typing in the instructions each time you do it. Second, it makes it easier on an operator if all they have to remember when backing up the disk is type "DO BACKUP" and follow the directions that appear on the video, answering all questions as they are asked.

The third application, installing patches to the system, is a method that we will be using to keep your DOSPLUS up to date and supply you with patches to other software to make it run with DOSPLUS. The method is simple. The PATCH program is able to accept input from a disk file containing an ASCII list of the patches. You would simply have to have the patch file present and you could instruct PATCH from the DO file to "patch this file using that set of patches".

Therefore, it is often easier to create a file (again, using the BUILD command), that contains these patches and then allow DO to instruct PATCH to install them. The reasons for this are clear. First, it allows you to review the patches before they are actually installed. Second, it allows you to easily move the DO file to another disk and install the same patches there. Third, it allows you an easy method of distributing these patches (in the case of software houses, to customers) to others.

When using DO from BASIC, you must provide for this buffer. You have basically two methods of doing this. One is to protect memory when calling BASIC with BASIC's own syntax for this (see BASIC) and the other is to call BASIC itself from within a DO file. Since DO protected its buffer with HIGH$ before calling BASIC, BASIC will not expect to be able to use that area of memory. And since DO will reuse it, there will be no conflicts. Choose whatever method pleases you most, but you MUST protect the memory.

The fourth and final application, automatic execution of programs, is the one that the average user will find the least useful. However, software manufacturers wishing to "demo" their programs have a powerful tool at their disposal, and the function should be described.

The method is simple. Create a DO file with all the needed information to begin operating your program. Then, include the statements needed in order to answer any prompts that the program might ask. Whenever your program request keyboard entry, DO will send it the next statement from the file.

Examples:

```
DO STARTUP (BREAK=Y)
DO STARTUP (B=Y)
DO STARTUP,B=Y
```

All three of these commands will execute the statements located in the file STARTUP/TXT. Pressing the BREAK key will abort the operation, because the BREAK key has been enabled.

```
DO FREE/DO:2
```

This command will execute the statements located in the file FREE/DO on drive 2. Notice that the extension "/TXT" was not used because another was specified.

```
DO TEST:3 (BREAK=N,HIGH=BFFFH)
DO TEST:3 (B=N,H=BFFFH)
DO TEST:3,B=N,H=BFFFH
```

These commands, all equivalent, will cause DO to set the HIGH$ value to BFFFH and then execute the instruction set in the file TEST/TXT located on drive 3. You will <u>not</u> be allowed to abort execution by pressing the BREAK key.

DUMP

This command allows you to take a specified area memory and transfer it to disk as a file.

==========

DUMP filespec (param=exp...)

filespec is the standard DOSPLUS file specification indicating which disk file you would like the information to be stored in.

(param=exp...) is the optional action parameter that modifies the action of the command.

Your parameters are:

DATA=switch	Indicates that the file being created is a non-program file. DUMP will not create a file in load module format in such cases.
END=value	Ending memory address.
RELO=value	Relocation address.
START=value	Starting memory address.
TRA=value	Transfer address.

Abbreviations:

DATA	D
END	E
RELO	R
START	S
TRA	T

==========

DUMP is used any time that you wish to transfer an area of memory to disk and store it as a file. It applies to both machine language programs and data files. Any area of memory (between the low memory and high memory pointers) may be dumped to disk.

Once the file is on the disk, in the case of machine language programs, you may either execute the file directly by typing in the filename from the DOS command mode or you may load the file via the LOAD command and execute it via DEBUG or by specifying the RUN parameter on LOAD (see the library commands LOAD and DEBUG).

By using the DUMP command, you may enter machine language programs into memory via the modification mode of the DEBUG command and then dump them into a disk file to be executed later.

The DUMP command in DOSPLUS is unique in the manner in which it allows you to completely control all items of information about the file as you are saving it to disk. You may alter the load address of the program or change the transfer address, whichever you choose.

When transferring memory to a disk file, if you wish the system to store it as data and not a machine language file, then you must remember to specify the "Data" parameter. Machine language programs and data files are stored on the disk in two completely different manners.

When a machine language program is stored on the disk, there are two pieces of information that the CPU needs to know in order to load and execute it properly. First, where the program loads or its "load address". Second, where in memory to pass the program control to after the program has been loaded or its "transfer address".

Under certain circumstances, you may want to alter either or both of these. You may wish, for example, to dump memory between 7000 hex and D000 hex to disk, but to have the system load it back later from 6000 hex to C000 hex. When you dumped that file, you would use the RELO parameter. You would set that parameter to "6000H" and from then on, when the system loaded that file, it would start at 6000 hex.

Some programs also use a transfer address that is different from their load address'. In other words, the program does not actually begin executing at the exact same location in memory as it loads. This is commonly handled by the assembler creating the object file, but in the case of a file being dumped to disk, that obviously would not apply. Therefore, DOSPLUS' DUMP command allows you to set a transfer address for the file.

If you specify neither the RELO or the TRA parameter, DUMP will assume that:

(1) The program loads back into memory at the same area that it came from.

(2) The program is NOT to be executed, but instead, you wish to return to DOS after loading the file.

Examples:

```
DUMP TESTFILE (START=7000H,END=D000H)
DUMP TESTFILE (START=28672,END=53248)
DUMP TESTFILE (S=7000H,E=D000H)
DUMP TESTFILE,S=7000H,E=D000H
```

All four of these commands will have the same effect. They will attempt to write the area of memory between 7000 hex and D000 hex to the first available disk drive under the filename "TESTFILE/CMD". The load address and the transfer address for the file will both be left set to 7000 hex. Note that the decimal input was used interchangeably with the hexidecimal.

```
DUMP DATAFILE/DAT:1 (START=3000H,DATA)
DUMP DATAFILE/DAT:1 (S=3000H,D)
DUMP DATAFILE/DAT:1,S=3000H,D
```

These three commands will all accomplish the same effect. They will move the area of memory from 3000 hex to whatever HIMEM is currently set to a disk file named "DATAFILE/DAT" located on drive ":1". It will store the information on the disk in data file format (as opposed to load file format).

ERROR

This command allows you to get a get a detailed message printout of any error number or a display of the last error displayed, depending on the syntax used.

==========

ERROR [value]

value is the optional error number that you wish to obtain a message for.

==========

DOSPLUS does provide you with detailed error messages instead of numbers, but for reference's sake, this command will still translate error numbers into error messages. A complete list of the error messages will be published in the technical section of the manual.

ERROR also has the unique feature of recalling the last error displayed. Simply enter the word "ERROR" without a number and the resulting message will be the last error the system displayed.

As stated before, DOSPLUS itself always prints out a detailed error message. However, some applications software, in keeping with TRSDOS tradition, may give you simply an error number. This command allows you to quickly see what the error message for that number is.

Other programs may use the ERROR command in TRSDOS within the actual program. For that reason, we have this command in DOSPLUS.

A very useful application of the ERROR command is the error "replay". If an error occurs and the message is scrolled off the screen, or for some other reason you are unable to read it, this will allow you to pick up the error message later.

Examples:

ERROR

This command will print to the screen the message corresponding to the last error displayed.

ERROR 32

This command will print the error message pointed to by Error 32. Note that only decimal values are accepted by this command.

This capacity (expressing values as literals) makes it easy for even the novice user to create these files. The next step is to include the equals sign ("=") to indicate that the translated value follows. Then you may express the new value for this character. Let's assume you wish to translate that capital "A" into a lower case "a". The statements could look something like these:

```
41=61
"A"="a"
```

you may also mix and match the types:

```
41="a"
"A"=61
```

Hexadecimal values are assumed and quoted literals are obvious to the system. Once you have completed the first translation, you may separate it from the next with a comma and proceed for as many of these as you need. You may also put each statement on a line by itself, if you wish.

An actual example of a filter file might be to filter out certain codes that would cause your printer to go into special print modes. Let's assume those codes are 0E (14 decimal) and 0F (15 decimal). Our filter file would be short (only two translations), and would look something like this:

```
0E=00,0F=00
```

Let's also assume that we store this on the disk with a filename of PRT/FLT. We would then say:

```
FILTER @PR PRT
```

to install the filter. You can create these filter files using the BUILD command (see the library command BUILD).

When using quoted literals in a filter file, you may use either single or double quotes.

Examples:

```
FILTER FROM @DO TO DISPLAY/FLT
FILTER @DO DISPLAY
```

These two commands are equivalent. Notice that the FROM and TO words are optional and that the extension "/FLT" is assumed. These will install the filter DISPLAY/FLT on the video device.

```
FILTER @DO (MAP)
```

This command will display any filter that is currently in effect for the video device. If one is NOT present, it will inform you that "No function exists". If one IS present, it will list that filter to the display. It will print the character, unless it is unprintable in which case it will print a period, followed by the value for that character in parenthesis. The equals sign and the new value (in the same format) will follow.

```
FILTER @DO (NO)
FILTER @DO (N)
FILTER @DO,N
```

These three commands will all dis-engage the filter currently on the video device. They will NOT deinstall the filter. Memory will still be reserved and the filter is still there if they wish to re-engage it.

```
FILTER @DO (YES)
FILTER @DO (Y)
FILTER @DO,Y
```

These three commands will re-engage any filter installed on the video device.

FORCE

This command allows you to route the I/O of one DOSPLUS system device to another.

==========

FORCE
FORCE [FROM] devicespec [TO] device/file

devicespec is the primary device which is to be routed.

device/file is a device or a file to which the primary device or file is to be routed.

==========

The FORCE command provides the DOSPLUS user with the ability to redirect the I/O paths of the system's devices. This provides unparalleled operational flexibility with a minimum of effort. With this command, lineprinter output may be sent to the display, or display output sent to a disk file. This avoids the need to rewrite programs in the event that, for example, a lineprinter should fail; all lineprinter output could merely be rerouted to the display, or to a disk file for later printing.

Unlike the JOIN command, which links two I/O devices together so that data goes to the two devices simultaneously, FORCE actually routes data intended for one device to another device or to a disk file.

Neither "devicespec" nor "device/file" default to anything. If a device or file is specified, then a file must also be specified. If a devicespec is specified, then a device or file must also be specified. If FORCE is entered without any device specification or channels, then the current device settings will be displayed. It will appear something like this:

```
$00 @KI <-  06CBH
$01 @DO <-> 08C5H
$02 @PR  -> 0BD9H
$03 @RS <-> 0CEAH
$04 @SI <-  NIL
$05 @SO  -> NIL
$06 @U1  -  NIL
$07 @U2  -  NIL
```

The I/O directions of the devices involved in the FORCE must be the same, that is. input devices may only be linked to other input devices, and output devices to other output devices. Routing an input device to an output device or vice versa, is illegal. For moving data between two devices of dissimilar natures, use the COPY command (see COPY).

If a device is routed to itself (i.e. FORCE @PR TO @PR), then that device is reset that device, that is, any previous routing established will be removed. You may also use the RESET command to remove any routing (see RESET).

Restrictions

(1) Only devices 0-7 can be the primary device. These are the system devices @KI, @DO, @PR, @RS, @SI, and @SO plus the two user-definable devices @U1 and @U2 (remember that these devices may be renamed; if they are, then the current name of the device is the one which should be used). Drives may NOT be specified, either as the primary device or the destination device/file. Drivespecs are valid only with filenames.

(2) Input devices should only be routed to other input devices (or devices capable of simultaneous input and output) and output devices may only be routed to other output devices or disk files. Routing an input device to an output device, or vice versa, is possible but the results are not always predictable.

(3) The order in which the devices are routed is important. Remember that you are FORCEing the primary device to the destination device or file. For example, if @DO is routed to @PR, the printer output will be sent to the display. Order is important.

(4) When routing an output device to a file, remember that the file will remain open until the device is reset and the FORCE removed. If the the computer is rebooted without resetting the device, the file may not be readable.

Examples:

FORCE @DO TO @PR

This command will send all output normally going to the display to the lineprinter instead. Once this command is given, no new data will appear on the display screen.

FORCE @PR PRINTFIL/TXT:3

All output to the lineprinter will be sent to a disk file called PRINTFIL/TXT on drive :3. If PRINTFIL/TXT previously exists, then any data going to the routed @PR device will OVERWRITE the contents of PRINTFIL/TXT. If PRINTFIL/TXT does not previously exist, it will be created on drive :3. The disk file will remain open until the routing is cancelled by means of the RESET command.

FORCE @PR TO @PR

This will cancel any active routing or linking for the @PR device, in effect performing a RESET @PR.

FORMS

This command will allow you to define certain parameters concerning the printer.

FORMS
FORMS (param=exp...)

Your parameters are:

PAGE=value	Number of physical lines on a page.
LINES=value	Number of lines per page to be actually used. The ROM driver will not implement this.
WIDTH=value	Number of characters that will be allowed on a line.
TOP	Sends an immediate top-of-form to the printer.
CODE=value	Sends the specified one or two byte value immediately to the printer.
INDENT=value	Number of characters to indent each printed line.
HTAB	Display current horizontal tabs.
HTAB=value	Set a horizontal tab.
RESET	Initializes character and line counters.
EMPTY	Flushes spooler (if active).
CR+LF	Activates automatic line feed after carraige return feature.
XLATE=switch	Engages/disengages translation of form feeds/tabs.

Abbreviations:

PAGE	P
LINES	L
WIDTH	W
TOP	T
CODE	C
INDENT	I
HTAB	H
RESET	R
EMPTY	E
CR+LF	CR
XLATE	X

The FORMS command gives you control over the items that will instruct the printer driver regarding the dimensions of the paper on which you will be printing. If you enter the word FORMS without any accompanying parameters, the current settings will be displayed.

PAGE. This parameter allows you to set how many physical lines there are on each page. Most printers will print 6 lines in every inch. Standard paper is 11 inches long. This means that standard 11 inch paper in a standard 6 line per inch printer mode will hold 66 lines per page. To use the parameter, simply set PAGE equal to the number of actual physical lines there are on the paper.

This parameter is useful when working with paper of varying lengths. When working with, for example, 14 inch long paper, you will have 84 lines on each page. When you execute a top-of-form, if the PAGE parameter had not been adjusted to indicate longer than normal paper, the system would not top-of-form correctly. It would not advance the paper far enough. The same holds true in the opposite direction with paper shorter than 11 inches. If you only have 42 physical lines on a page and execute a top-of-form with the PAGE parameter set for 66 lines, the system will advance the paper too far.

This can also be used to work with printers that print more than the standard 6 lines per inch. Any application that increases or decreased the actual physical number of lines that can be printed on a page will be adjusted for using this parameter.

LINES. This parameter is used to adjust the number of lines that the driver will use on each page before executing an automatic top-of-form to the next. This allows you to automatically skip over perforations on fan fold paper. Without having the LINES parameter set correctly, pagination would not be possible. To use it, set LINES equal to the desired number of lines.

When using this parameter, simply set LINES to execute the top-of-form at the proper time to provide the desired margins. This option can be very nice when the operation being performed has no internal provision for paginated output.

WIDTH. This parameter will allow you to instruct the printer driver how many characters to print on a line before it terminates that line. When the number of characters printed on that line attempts to exceed the value defined for WIDTH, the printer driver terminates that line, outputting a carriage return at the point you tell it to with WIDTH and then printing the rest of the characters on the next line. To use it, set WIDTH equal to the number of characters that can be printed on a single line.

This parameter comes in most useful when using 8.5 inch wide paper in printers designed to accommodate 13 inch paper as well. Many printers that are designed for the narrower paper will automatically "wrap around" any lines that exceed the maximum paper width. However, a printer that can handle either size paper would have no real way of knowing how wide the particular paper loaded at the moment would be. This means that a wide carriage printer printing upon a narrow sheet of paper could run over the edge and print on the platen.

The WIDTH parameter will prevent this. If you set it for the maximum number of characters your printer can print on any particular size paper, it will wrap any lines exceeding the maximum length down to the next line.

One item to watch out for is special graphics modes on various lineprinters that cause you to output great numbers of bytes in order to configure the printer or engage the various graphics modes. Sometimes this can cause to to attempt to output more than the limit of characters. If the WIDTH parameter is set to 0, then DOSPLUS will allow an infinite number of characters on a line and it is not a problem, but if you have engaged the WIDTH to do wrap around of text, your character counter will be inaccurate due to the codes sent. Use the RESET parameter to reset this counter to 0.

TOP. This parameter sends an immediate top-of-form code to the printer. This provides a simple method for advancing the paper without having to touch the printer. That is especially nice for the various styles of printers that have no simple switch to top-of-form.

There are no parameters for TOP, simply specify it in a command line and the system will send the printer an ASCII 0CH (form feed). If your printer does not perform a top-of-form when receiving that code, you may use the CODE parameter to send whatever code is required. If you wish, engage the XLATE parameter and then use TOP. XLATE will simulate the top-of-form with linefeeds and should work with any printer regardless.

CODE. This parameter allows you to immediately transmit a one or two byte decimal or hex value to the printer. This can be used for a wide variety of purposes. Some of them include:

(1) Sending a top-of-form code to a printer that does not recognize 0CH as such.

(2) Sending a line feed to the printer.

(3) Setting some special mode (underline, boldface, etc.) from the DOS command level.

(4) Using it within a DO file to output codes to the printer to configure it for a particular applications program about to be executed.

These are just some examples of how you can use the CODE parameter. Because you can send any codes you desire, this command is totally flexible. You will more than likely develop many more uses for it than just those listed.

To use this parameter, simply specify the desired one or two byte value by setting CODE equal to that value in the command line. Decimal or hex input is accepted. Be certain to append a trailing "H" to any hexadecimal input. You may send multiple values by using the multiple command feature of DOSPLUS.

When sending a two byte hexadecimal value with CODE, remember to send it in LSB-MSB order. For example, to send the codes 27 (1BH) and 15 (0FH) as one two byte hexadecimal value, it would be done with the statement:

```
FORMS (CODE=0F1BH)
```

The last byte given is sent first. Notice the "H" following the value. This is mandatory for hexadecimal input.

INDENT. This parameter allows you to set a pre-defined number of characters that the printer will space over at the beginning of each line. Please note that is for each line printed, not just the lines that are wrapped around. This may be a value between 1 and 255. Simply specify INDENT=10, for example, to indent 10 characters.

This can be useful when you are running printouts that will later be punched for insertion into a notebook. It allows you to provide for a left margin.

HTAB. This is a dual purpose parameter. If simply included in the command line by itself, it will cause FORMS to display whatever horizontal tabs are currently defined by the user. DOSPLUS supports horizontal tabs to the line printer and allows the user to define them if they wish. What this means is that every time you output a tab character to the printer (from BASIC LPRINT CHR$(9)), DOSPLUS will advance the printer to the next tab zone.

If you have not set any tabs, DOSPLUS has default tabs that occur every 8 characters. Once you begin specifying tabs, these default tabs are no longer in effect. As we said above, this is a dual purpose command. To set a tab, we also use the same HTAB parameter. Simply set HTAB equal to the tab location (i.e. FORMS,HTAB=10 sets a tab at position 10). To reset all tabs, use HTAB=0.

RESET. This parameter allows the user to initialize the character and line counters. DOSPLUS maintains two pieces of information that tell the printer driver where on the page it was and where in a particular line of that page it was. This can sometimes cause a problem if a printout was interrupted and the paper advanced manually before a top-of-form was issued.

In those cases, when printer output is resumed, DOSPLUS will only print until it believes that it has filled up the page or line that it was previously printing. This can cause early line wrap and page advance. To "start fresh" without a top-of-form, include the RESET parameter in the list and the counter will be set to 0.

EMPTY. This parameter, is specified, will cause the DOSPLUS spooler to suspend output and empty its buffer. This, of course, assumes that you have installed the spooler (see Drivers and Filters: SPOOL).

Remember that in addition to suspending output, it resets the buffer. This is not a "printer pause" feature. When you empty the spooler, whatever text was in it is simply discarded.

CR+LF. This parameter engages and disengages an option that sends a line feed after every carraige return. If your line printer does not give you a line feed automatically with carraige return, then engage this parameter. Any output that goes through the standard printer driver after that will have a line feed sent after each carraige return.

XLATE. This parameter allows you to compensate for line printers that do not understand such things as form feed or tab codes. It will simulate these with line feeds and spaces. Essentially, if your line printer won't respond to the actual codes for these functions, engage this parameter and let the DOS simulate these actions instead.

Examples:

```
FORMS
```

This would cause the forms parameters to be displayed on the screen.

```
FORMS (PAGE=66,LINE=60,WIDTH=80)
FORMS (P=66,L=60,W=80)
FORMS,P=66,L=60,W=80
```

This will set the page length to 66 lines, the number of lines to be used to 60, and the maximum line width to 80 characters.

```
FORMS (PAGE=66,TOP)
FORMS (P=66,T)
FORMS,P=66,T
```

This command will set the page length to 66 lines and transmit a top-of-form (ASCII 0CH) to the line printer.

```
FORMS (CODE=10)
FORMS (C=10)
FORMS,C=10
```

This command will send a decimal 10 (hexadecimal 0A) to the printer. Normally, this will produce a single linefeed. The value could have been given as "0AH".

```
FORMS (INDENT=10,HTAB=12,XLATE)
FORMS (I=10,H=12,X)
FORMS,I=10,H=12,X
```

This command will set the left margin at 10 characters, set a horizontal tab at position 12, and engage the translation feature for forms feeds and tabs.

```
FORMS (RESET,EMPTY,HTAB)
FORMS (R,E,H)
FORMS,R,E,H
```

This command will initialize the the character and line counters, empty the spool buffer, and display all currently set horizontal tabs.

FREE

This command will display the free storage space and remaining directory space on all mounted disks.

==========

FREE
FREE [FROM] drivespec [TO] device/file

> drivespec is the optional drive specification. If given, a free space map of that drive will be displayed. If omitted, the free space summary for all mounted disks will be shown.
>
> device/file is the optional output device or file.

This command has no parameters.

==========

The FREE command, when given without any drivespecs, will read the directory of each mounted disk and determine the amount of space available on that disk. "Mounted disks" includes logical drives (such as those on a hard drive which has been split up into one or more volumes). The free space remaining on each disk will then be displayed, along with the number of available directory slots for new files. Free storage space will be given in kilobytes.

It is quite possible that a disk may have free disk space remaining, but has a full directory. In this case, even though there is storage space remaining on the disk, no new files may be placed on it because there is no more room in the directory to hold information about that new file. Conversely, there may be available directory slots, but no free storage space remaining on the disk. DOSPLUS will create the file but will not allocate any space to it in this case.

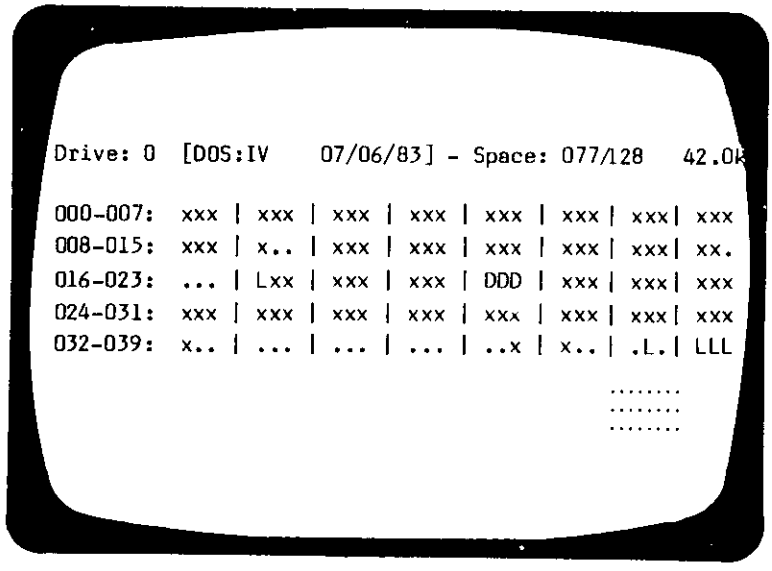
If FREE is given with a drivespec, then DOSPLUS will read only that drive, and then display a map of the disk. The map will show all the formatted sectors on the disk and indicate which ones are in use, which ones are free, and which ones are unavailable (locked out). Granules allocated to a file will be displayed with an "x", free granules with a "." (period). The directory track will have its granules displayed with a "D". This display will be by cylinder, with the cylinder numbers for each line given in the farthest left hand column.

The information generated by the FREE command is normally sent to the video display, but may be sent to any valid output device or a disk file simply by specifying that you desire this. For example, FREE TO @PR, will send the free space information to the line printer.
4 07/04/83] - Space: 254/256 3440.0k
Drive: you should see something similar to this:

```
Drive: 0 [DOS:IV    07/06/83] - Space: 077/128    42.0k
Drive: 1 [DATA      06/27/83] - Space: 060/064    81.2k
```

As covered above, the first item displayed is the drivespec, disk name, and the disk date. Second is the amount of free directory space remaining. The first number is the amount of directory entries free and the second reflects the total available on that drive. By subtracting the first number from the second, you may arrive at the number of directory entries used for that drive. Following that is the free diskette space expressed in kilobytes. This value will be rounded to one decimal place.

If you request FREE with a drivespec, you will get the same line of information, but only for the drive you request. Following that will be the free space "map". This map is also described above. Some maps, such as those for a hard disk volume, may be large enough to scroll off the screen. In these cases, you may press the SPACE BAR to pause the output. By pressing it again, you will re-start it. This allows you to step through the free space map. A free space map will look something like this:



When using the optional output device or file on FREE, be certain to include the delimiter "TO" if you have not given a source drivespec. If you do not, FREE will attempt to interpret the output device as the drive desired and return an error.

For example:

FREE @PR

will cause FREE to evaluate "@PR" as the source drivespec. This is invalid, and an error will result. If you include the word "TO", such as:

FREE TO @PR

all will be fine.

Examples:

```
FREE
```

This command will display free space information for all mounted drives on the video display.

```
FREE TO @PR
```

This command will send the free space information for all mounted drives to the lineprinter.

```
FREE FROM :1 TO @PR
FREE :1 TO @PR
FREE :1 @PR
```

This command will send a map of the disk in drive 1 to the lineprinter.

```
FREE TO FREEINF/TXT:A
```

This command will send the free space information for all mounted disks to a file called FREEINF/TXT on drive :A.

I

This command will instruct the system to log in a disk.

==========

I [drivespec] (param=switch)

drivespec is the specifier of the name of any valid drive in the system.

(param=switch) is the optional action parameter.

Your parameter is:

MOUNT=switch Logs in the disk immediately.

Abbreviation:

MOUNT M

==========

The I command allows you to instruct DOSPLUS to log in a specific disk or all disks. When DOSPLUS "logs in" a disk, it reads the information stored on the disk that describes the disk to the system and it also reads the directory. Therefore, if a disk will log in properly, you are assured of two items:

(1) DOSPLUS now knows exactly how that disk is formatted.

(2) That disk is readable to the system (at least in a general sense, obviously specific read errors could exist elsewhere.)

When the I command is issued, the system will be flagged to log in a mounted disk at the next disk access. If no drivespec is given, then the system will flag for all mounted disks as necessary and read the information from each disk at the first access of that disk following the I command. If a drivespec is specified, then the system will flag just for the disk in that drive.

MOUNT. This parameter will cause the system to immediately read the information from the specified drive into memory without waiting for the next disk access.

Put simply, all the I command does is "initialize" the disk from the DOS' point of view. When you use the I command, DOSPLUS "forgets" all that it knows about the disk and will re-establish all information on the next disk access. By using the MOUNT parameter, you can force it to do that immediately and not wait for the next disk access.

This command can also be useful in determining that a disk is readable before continuing with some procedure. If you can mount the disk, DOSPLUS can read it. If the format is incompatible or the disk is blank, it will not mount properly.

DOSPLUS will not require the I command to log disks, inasmuch as it will even switch between densities, track counts, and surface counts automatically without error. There are, however, occassions such as just before saving a system configuration file (see SYSTEM) that you may wish for the drives to be logged in so that the CONFIG display reflects them as best possible.

Also, there is one other great advantage to using the I command with the MOUNT parameter to log in the disks. Once the DOS logs in all the disks, the information in all of the DCTs (Drive Control Tables) is up to date and correct for all of the available disk drives you have in the system at the time that you used the "I,M" command. Now DISKZAP, that used to require you to "set" each disk with regards to track count, surface count, density, etc., will require this no more. When using DOSPLUS with hard disks and wishing to implement DISKZAP, this is even more convenient.

Examples:

```
I
```

This command will cause DOSPLUS to flag the DCT information in all drives.

```
I :0 (MOUNT)
```

This form of the I command will cause the system to read drive ":0" for its drive control information.

JOIN

This command will link together two devices within the DOSPLUS system. This command is identical to the LINK command.

JOIN [FROM] devicespec [TO] device/file

devicespec is the primary device which is to be linked.

device/file is a device or file with which the primary device is to be linked.

The JOIN command allows simultaneous I/O from two devices in the system. If, for example, you wanted a hard copy of everything that appeared on your video display, you could JOIN the @DO device to the @PR device. After the JOIN is established, then everything going to the display will also be sent to the printer.

It is also possible to JOIN an output device to a file, so that everything sent to that device will simultaneously be sent into a disk file. For example, linking @PR to a file will duplicate all printer output into a disk file.

Neither "devicespec" nor "device/file" default to anything. If a device or file is specified, then a devicespec must also be specified. If a devicespec is specified, then a device or file must also be specified. If neither are specified, then the JOIN settings for all devices, if any such settings exist, will be displayed on the video screen. It will appear something like this:

```
$00 @KI <-  06CBH
$01 @DO <-> 08C5H
$02 @PR  -> 0BD9H
$03 @RS <-> 0CEAH
$04 @SI <-  NIL
$05 @SO  -> NIL
$06 @U1  -  NIL
$07 @U2  -  NIL
```

The I/O direction of the linked devices must be the same, that is, input devices can only be linked to other input devices, and output devices can only be linked to other output devices. Linking an input device to an output device, and vice versa, is illegal. For moving data between two devices of dis-similar natures, use the COPY command (see COPY).

Linking a device to itself (e.g., JOIN @PR TO @PR) will reset that device; that is, any previous linking established will be removed. You may also use the RESET command to remove any linking (see RESET).

Restrictions

(1) Only devices 0-7 can be the primary device. These are the system devices @KI, @DO, @PR, @RS, @SI, and @SO plus the two user-definable devices @U1 and @U2 (remember that these devices may be renamed; if they are, then the current name of the device is the one which should be used). Drives may NOT be specified, either as the primary device or the linked device/file. Drivespecs are valid only with filenames.

(2) Input devices should only be linked to other input devices (or devices capable of simultaneous input and output) and output devices may only be linked to other devices capable of output or disk files. Linking an input device to an output device, or vice versa, is possible but the results are not always predictable.

(3) The order in which devices are linked together is important, since the JOIN is essentially in one direction only (e.g. from the device being linked to the device or file with which the link is established). For example, if @DO was linked to @PR any output sent to @DO would also appear on @PR, but any output sent to @PR would not appear on @DO. JOIN does not establish a two way link.

(4) When linking an output device to a file, remember that the file will remain open until the device is reset and the JOIN removed. If the computer is rebooted without resetting the device, the file may not be readable.

Examples:

```
JOIN @PR TO @DO
```

This command will send all printer output to the video display simultaneously. However, display output will not be sent to the printer.

```
JOIN @DO FILE1/TXT
```

This command will duplicate all data sent to the screen in a file called FILE1/TXT. This would, in effect, give you a "record" of what occurred on the screen in a given period of time (e.g. during the effect of the JOIN). If FILE1/TXT exists, the old data will be overwritten. If it does not, DOSPLUS will create the file.

```
JOIN @RS @KI
JOIN @DO @RS
```

These two commands will link the serial communications device to the keyboard device and the video display to the serial communications device. Since @RS is capable of both input and output, these two links are valid. After these two commands are given, any input that comes over the @RS device will be treated as keyboard input, and any output going to the display will also be sent out the serial communications device. If the serial communications device was set up correctly previously, these two commands will allow your computer to be controlled from a remote terminal. However, the local keyboard remains active, so that any commands typed in at the keyboard will also be handled normally. This, in effect, creates a "host".

```
JOIN @PR TO @PR
```

This command will RESET the @PR device. Any FORCEing or JOINing (see below) which may have been active will be removed.

```
JOIN
```

This command, with nothing given in the I/O field, will simply display a list of devices with their current JOIN settings if any.

Here are some examples of illegal JOINs:

```
JOIN FOO/BAR:00 TO @KI
```

Only devices may be linked, not files. Files may serve as the destination device in a link, but not the primary device.

```
JOIN @DO :1
```

You cannot simply link to a drivespec. You have not correctly specified an output file or device.

KILL

This command will delete a file or group of files from a disk. It will also disable any active devices or drives. This is indentical to the REMOVE command.

==========

KILL filespec (param=exp)
KILL [FROM] drivespec [USING] wildmask (param=exp)
KILL devicespec
KILL drivespec

filespec is the name of the file you wish to delete.

wildmask and drivespec are the wildmask and optional drivespec that indicate the group of files that you wish deleted. If the drivespec is omitted, the system drive will be used.

(param=exp...) is an optional parameter affecting the action of the KILL command. You only have valid parameters when killing files.

devicespec is the name of a device you wish to deactivate.

drivespec is the name of a drive you wish to remove from the system.

The valid parameters for this command are:

INV=switch	Specifies whether or not invisible files are to be included when a wildcard delete is done.
ECHO=switch	When doing a wildmask delete, this will display the name of each file as it is killed.
SYS=switch	Specifies whether or not system files are to be included in a wildcard delete.
QUERY=switch	This will cause the system to display the filespec and prompt you for a reply before deleting the file.
PW="string"	This parameter declares the DISK master password to the system, which will be used in place of file passwords when wildmasks are specified.

Abbreviations:

INV	I
ECHO	E
SYS	S
QUERY	Q
PW	P

==========

The KILL command is used to delete items from the system that are no longer needed or not desired. This may include files, devices, or drives. When deleting files with KILL, there are basically two modes of operation: standard and global.

The standard method is invoked by entering the KILL command followed by a filespec. This will delete that file from the disk. The global method is invoked by entering the KILL command followed by a wildmask. This causes KILL to delete all files matching that mask. When using either of these methods, you have available the above listed parameters to modify the manner in which KILL works.

When KILL is typed with a filespec, but without a drivespec, the system will perform a global search of all mounted disks until it finds the first occurrence of the file, which it will then delete. If a drivespec is supplied, then only that drive will be searched.

When KILL is typed with a wildmask, and no drivespec is supplied, the current system drive will be searched. The system will search the directory of the specified drive for the first filename that fits the wildmask, and kill it. It will then continue to search for other files which will fit the mask, killing each one that it finds.

When using a wildmask, the KILL command requires that the disk's master password be given with the PW parameter. In this case, the disk master password will be used in place of the file passwords when a password protected file is encountered. This does not apply if the Disk Master Password is not set for that drive. Because the password is set to null (no password), by omitting the password you are in effect specifying the correct password (since "no password" is the password). The fact that anyone knowing the Disk Master Password can delete any file on the disk (system files included), should be sufficient to illustrate the importance of setting the Disk Master Password in DOSPLUS. Our increased use of that password means that for a disk to be at all protected, a password must be set.

INV and SYS. These parameters are used when doing a multiple-file kill using a wildmask. Normally, this type of multiple-file kill includes only visible user files. However these two parameters allow you to include invisible and system files. The INV parameter will include invisible files, and the SYS parameter will include system files in the wildmask search. Remember that system files are also invisible, so to include the system files, you must specify both parameters.

ECHO. This parameter may be used when doing a wildmask search, to display the names of the files as they are deleted. You will find that for the most part, this is a desirable option. It is not often that you want to kill multiple files from a disk and not be told which files are being killed. If you see a filename that you did not mean to be included, you have the option of using the RESTORE utility to recover it. As a rule of thumb, always turn ECHO on during a wildmask kill.

QUERY. This parameter will force the system to display the filename before killing it, and prompt the user for a yes or no reply. The file will be killed only if you specifically reply "Y" when prompted. Pressing ENTER alone will not kill the file. This is most useful when the wildmask you have specified is so general that files will be included that you don't want deleted. In most instances, you would rather spend the extra time answering a prompt than recovering a file killed by accident.

The QUERY parameter will override the ECHO parameter. This means that if the system prompts you as it kills the files, it will not re-display the filename if you say "Y".

PW. This parameter declares the disk's master password to the system. This password will be used instead of the file access and update passwords when a wildmask search encounters a protected file. The disk's master password must be enclosed in either single quotes or double quotes and may be in upper or lower case, or both.

When doing a wildmask KILL, this parameter is REQUIRED. The disk master password MUST be specified even though the files are not password protected as long as wildmasks are used.

KILL may also be used to disable devices. If a devicespec is given, then a bit will be set in that device's DCB indicating that it is set to NIL. It will become unavailable until it is re-entered in the table by means of the RESET command (see RESET). For example, KILL @PR will remove the lineprinter device from the system.

Similarly, disk drives may be disabled with KILL. When disk drives are KILLed, they are also set to NIL. Simply because a drive is set to NIL does not mean that it was once active and is now removed. If a driver was never installed for a device, then obviously it cannot be recovered.

When I/O is performed to any KILLed output device, no error message will be returned; however the data will go nowhere.

When a disk drive is KILLed, any attempt to access that drive will return the error message, "Drive not available."

Care should be taken when using KILL to disable devices. The only way out of injudicious use of this command may be to reboot (for example, KILL @KI will disable the keyboard; the only possible recovery from this case would be to reset the entire system). However, if a device is linked to another, the other device will continue to be active even if the first device is KILLed.

KILLed devices and disk drives may be restored to an active state by setting the device back to itself, for example, ASSIGN @PR @PR (See ASSIGN).

Examples:

```
KILL FIRST/CMD
```

This command will cause the system to search the directories of all mounted disks for the first occurrence of FIRST/CMD, which will then be killed.

```
KILL FIRST/CMD:1
```

This command will cause the system to search the directory on drive :1 for FIRST/CMD. If it finds the file, the file will be killed. If the file does not exist on that directory, the system will return a "File not found" error.

```
KILL */OLD:4 (QUERY=Y,PW="SUPER")
KILL */OLD:4 (QUERY,PW="SUPER")
KILL */OLD:4 (Q,P="SUPER")
KILL */OLD:4,Q,P="SUPER"
```

The system will search the directory of the disk on drive :4 for every file with the extension /OLD. It will then display the filename that it finds which fits the wildmask and ask the user whether that file is to be killed or not. If the user replies "Y", then the file will be killed. Otherwise the file will be left alone, and the search will continue for other files with the /OLD extension.

```
KILL PROG?/BAS:A (PW="PASSWORD")
KILL PROG?/BAS:A (P="PASSWORD")
KILL PROG?/BAS:A,P="PASSWORD"
```

This command will search the directory on drive A for any filename that fits the wildmask PROG?/BAS and kill them. Files with names such as PROG1/BAS, PROGA/BAS, PROG$/BAS, etc. would be killed. Note that the disk's master password must be supplied. Due to the lack of QUERY and ECHO parameters, you will not be prompted before the file is killed, nor will you see the filename as it is deleted.

```
KILL MYDATA/DAT.SECRET:0A
```

This command will remove all traces of the file called MYDATA/DAT.SECRET from the disk in drive :0A.

```
KILL :02
```

The disk drive designated as :02 will be disabled. It will be disabled and any attempts to read or write drive :02 will produce an error.

```
KILL */*:X1 (QUERY=Y,PW="CIA")
KILL */*:X1 (QUERY,PW="CIA")
KILL */*:X1 (Q,P="CIA")
KILL */*:X1,Q,P="CIA"
```

This form of the KILL command is a global KILL. Any filename will fit the */* wildmask form, so the use of this command will result in every file on drive :X1 being killed. In this case, QUERY is switched on and the user will be prompted before each file is killed.

```
KILL !:0 (PW="MYFILE",QUERY=N,ECHO=Y)
KILL !:0 (PW="MYFILE",ECHO)
KILL !:0 (P="MYFILE",E)
KILL !:0,P="MYFILE",E
```

The ! is a special wildcard character which is the same as the wildcard combination */*. This command would result in every file on drive :00 being killed. The disk password "MYFILE" will be used to access any file encountered which is password protected. The name of each file will be displayed as it is KILLed, and the user will not be prompted before a file is KILLed. Note that since the parameters default to "Y" if specified and "N" if omitted, the QUERY parameter can be dropped since it is not desired and the expression "=Y" can be dropped from the ECHO parameter.

LIB

The LIB command will send a list of the DOSPLUS library commands to the specified output device.

==========

LIB [TO] device/file

> device/file is any valid output device in the system. If specified, it may not contain any wildmasks.

There are no parameters with this command.

==========

The LIB command will display a list of the DOSPLUS library commands, or send the list to a user-specified output device or a disk file, which may be any output device (for example, @PR) or filespec (for example, LIBRARY/COM:3). If not specified, it will default to @DO, the video display.

DOSPLUS distinguishes between library commands and programs. Library commands are routines which are within the operating system itself. These are the commands displayed with LIB. Library commands are given priority over programs; that is, if a program's filespec is the same as one of the library commands, the system will execute the library command rather than the program. The system is extended by adding programs which perform functions not covered by the library commands. These programs are not part of the operating system itself, and the system is not affected when they are killed. Conversely, library command routines cannot be easily removed from the operating system.

When specifying an output device or file, wildmasks should not be used, and will be rejected. For example, the command LIB TO */LST would not be valid. When sending the list of commands to a file, drivespecs may or may not be specified. If omitted, the system will use the first available drive.

Examples:

```
LIB
```

This command will display a list of the library commands on the video screen. It is identical to LIB @DO.

```
LIB TO LIBLIST:4
```

This command will send the listing of library commands into a file called LIBLIST on drive 4.

```
LIB @PR
```

This command will output the library command listing to the lineprinter.

LINK

This command will link together two devices within the DOSPLUS system. This command is identical to the JOIN command.

LINK [FROM] devicespec [TO] device/file

devicespec is the primary device which is to be linked.

device/file is a device or file with which the primary device is to be linked.

The LINK command allows simultaneous I/O from two devices in the system. If, for example, you wanted a hard copy of everything that appeared on your video display, you could LINK the @DO device to the @PR device. After the LINK is established, then everything going to the display will also be sent to the printer.

It is also possible to LINK an output device to a file, so that everything sent to that device will simultaneously be sent into a disk file. For example, linking @PR to a file will duplicate all printer output into a disk file.

Neither "devicespec" nor "device/file" default to anything. If a device or file is specified, then a devicespec must also be specified. If a devicespec is specified, then a device or file must also be specified. If neither are specified, then the LINK settings for all devices, if any such settings exist, will be displayed on the video screen. It will appear something like this:

```
$00 @KI <-  06CBH
$01 @DO <-> 08C5H
$02 @PR  -> 0BD9H
$03 @RS <-> 0CEAH
$04 @SI <-  NIL
$05 @SO  -> NIL
$06 @U1  -  NIL
$07 @U2  -  NIL
```

The I/O direction of the linked devices must be the same, that is, input devices can only be linked to other input devices, and output devices can only be linked to other output devices. Linking an input device to an output device, and vice versa, is illegal. For moving data between two devices of dis-similar natures, use the COPY command (see COPY).

Linking a device to itself (e.g., LINK @PR TO @PR) will reset that device; that is, any previous linking established will be removed. You may also use the RESET command to remove any linking (see RESET).

Restrictions

(1) Only devices 0-7 can be the primary device. These are the system devices @KI, @DO, @PR, @RS, @SI, and @SO plus the two user-definable devices @U1 and @U2 (remember that these devices may be renamed; if they are, then the current name of the device is the one which should be used). Drives may NOT be specified, either as the primary device or the linked device/file. Drivespecs are valid only with filenames.

(2) Input devices should only be linked to other input devices (or devices capable of simultaneous input and output) and output devices may only be linked to other devices capable of output or disk files. Linking an input device to an output device, or vice versa, is possible but the results are not always predictable.

(3) The order in which devices are linked together is important, since the LINK is essentially in one direction only (e.g. from the device being linked to the device or file with which the link is established). For example, if @DO was linked to @PR any output sent to @DO would also appear on @PR, but any output sent to @PR would not appear on @DO. LINK does not establish a two way link.

(4) When linking an output device to a file, remember that the file will remain open until the device is reset and the LINK removed. If the computer is rebooted without resetting the device, the file may not be readable.

Examples:

```
LINK @PR TO @DO
```

This command will send all printer output to the video display simultaneously. However, display output will not be sent to the printer.

```
LINK @DO FILE1/TXT
```

This command will duplicate all data sent to the screen in a file called FILE1/TXT. This would, in effect, give you a "record" of what occurred on the screen in a given period of time (e.g. during the effect of the LINK). If FILE1/TXT exists, the old data will be overwritten. If it does not, DOSPLUS will create the file.

```
LINK @RS @KI
LINK @DO @RS
```

These two commands will link the serial communications device to the keyboard device and the video display to the serial communications device. Since @RS is capable of both input and output, these two links are valid. After these two commands are given, any input that comes over the @RS device will be treated as keyboard input, and any output going to the display will also be sent out the serial communications device. If the serial communications device was set up correctly previously, these two commands will allow your computer to be controlled from a remote terminal. However, the local keyboard remains active, so that any commands typed in at the keyboard will also be handled normally. This, in effect, creates a "host".

```
LINK @PR TO @PR
```

This command will RESET the @PR device. Any FORCEing or LINKing (see below) which may have been active will be removed.

```
LINK
```

This command, with nothing given in the I/O field, will simply display a list of devices with their current LINK settings if any.

Here are some examples of illegal LINKs:

```
LINK FOO/BAR:00 TO @KI
```

Only devices may be linked, not files. Files may serve as the destination device in a link, but not the primary device.

```
LINK @DO :1
```

You cannot simply link to a drivespec. You have not correctly specified an output file or device.

LIST

This command will list data from a device or disk file to a specified output device or file.

==========

LIST [FROM] device1/file1 [TO] device2/file2 (param=exp...)

device1/file1 is the source device or file.

device2/file2 is the optional destination device or file.

(param=exp) is the optional action parameter.

Your parameter is:

CTL=switch	Determines whether or not control codes (ASCII 00H - 1FH) will be output unchanged or whether they will be displayed as periods (".).

Abbreviation:

CTL C

==========

This command allows you to list the contents of a disk file to the video (or any other output device such as a lineprinter). It can also list data from other input devices (i.e. the keyboard, RS232, etc.).

When the LIST command is outputting data, all control codes (those codes in the ASCII range that are not printable characters and are used for various "control" functions) will normally be sent as periods (.). The exceptions are carraige returns and linefeeds which are always displayed.

Normally, when outputting to the video, you will want to leave this in effect, less unwanted control codes set reverse video or some other undesired condition. However, if you have data in a disk file that was meant for a line printer (you perhaps FORCEed the data there earlier), it will be required that you send the control characters untranslated so that the printer will respond to the codes as normal.

While the LIST command is outputting data, you may press the SPACE BAR to pause the output or the BREAK key to abort.

Examples:

```
LIST ABC/BAS
```

This command will list the disk file called ABC/BAS to the video display (default output device). Control codes will be displayed as periods (".").

```
LIST ABC/BAS (CTL=Y)
LIST ABC/BAS (CTL)
LIST ABC/BAS,CTL
LIST FOO/BAS,C
```

This command is identical to the first one except that now control codes (00H to 1FH) are output unchanged. All other characters will be listed as is. Depending on the control codes present in the file called ABC/BAS the display may react unpredictably.

```
LIST ABC/BAS:1 TO @PR
```

The file called ABC/BAS on drive 1 will be listed to the lineprinter.

```
LIST FROM @KI TO @PR (CTL=Y)
LIST @KI @PR (CTL)
LIST @KI @PR (C)
LIST @KI @PR,C
```

This command will echo keyboard input to the lineprinter device, in a fashion similar to the COPY command. Keyboard input will not be passed to the DOSPLUS system for interpretation as commands. Control codes will be output unchanged (however, the lineprinter may act on certain codes, for example a form feed).

All characters typed in at the keyboard will continue to be sent to the lineprinter until BREAK is pressed.

LOAD

The LOAD command will load a disk file into memory without executing it.

LOAD [FROM] filespec (param=exp...)

filespec is the name of the file to be loaded.

(param=exp...) is an optional parameter which may be specified with the command.

The parameters are:

PROMPT=switch	Determines whether the system will prompt the user for disk mounts or not.
RUN=switch	Determines whether the file is to be executed upon completion of loading.
START=address	Determines the starting point in memory for loading a core image file.
TRA=address	This parameter will determine what address control is to be transferred to if the file is to be executed.

Abbreviations:

PROMPT	P
RUN	R
START	S
TRA	T

The LOAD command will take a file from disk and load it into memory. If the file is a program file (e.g. it is in executable format), then the address at which it is to be loaded will be taken from the file itself. If the file is not in executable format (e.g. it is a "core-image" file), the START parameter must be specified.

A "core-image" file is any file that does not contain loader codes. Executable program files contain special codes which tell the system where in memory it is to load, and what its starting address is. A file which does not contain these codes is considered to be a "core-image" file. Such a file may consist of binary program instructions or ASCII text. It is generally given the extension /CIM. If a file is specified without an extension, LOAD will assume the extension /CIM if START is specified, or the extension /CMD if not.

PROMPT. This parameter will allow you to load programs from other than a system diskette using the system drive. You will be prompted to mount the proper diskette. Pressing ENTER will proceed with the load. If the program is to be executed, you will then be prompted again to re-mount the system disk before the program is executed.

RUN. This parameter tells the system that you want the file to be executed after loading. Using this parameter along with the PROMPT parameter allows you to execute machine language programs directly from data diskettes in a single drive system.

START. This parameter informs the system where to start loading a core-image file. These files do not contain loader codes which tell the system where in memory they are to load, so the load address must be supplied by the user.

TRA. This parameter tells the system where the entry point of a core-image file is, and is generally given with the RUN parameter. After the file has loaded into memory, control will be transferred to the address supplied with the TRA parameter. This parameter can also be used to override the normal entry point address of a /CMD file. Since programs may also be saved as core-image files without loader codes, the LOAD command will also allow you to specify a transfer address if you wish to run such a file after loading. This address is specified by the TRA parameter.

Examples:

```
LOAD TEST/CMD:1
```

This command will load the program file TEST/CMD from disk drive 1 into memory. The locations into which it loads will be determined from the special loader codes within the file itself. Control is passed back to DOSPLUS after the file is loaded.

```
LOAD TEST/CMD:1 (RUN)
LOAD TEST/CMD:1,R
```

The file TEST/CMD is loaded into memory from drive 1. As soon as the file is loaded, control is passed to it and it will begin executing. This is the same as typing "TEST:1" from the DOS command level.

```
LOAD FOOBAR/CMD:0 (PROMPT=Y,RUN)
LOAD FOOBAR/CMD:0 (PROMPT,RUN)
LOAD FOOBAR/CMD:0,P,R
```

The program file FOOBAR/CMD is to be loaded from disk drive :0. The system will prompt the user to mount the correct disk containing FOOBAR/CMD in drive :0 before it begins the load. The user should insert the disk in drive :0 and press <ENTER>. As soon as the file is loaded, you will be prompted to reinsert the system disk. Then FOOBAR/CMD will execute.

```
LOAD MEMTEST (START=7C00H)
LOAD MEMTEST (S=7C00H)
LOAD MEMTEST,S=7C00H
```

MEMTEST/CIM will be loaded into memory starting at address 7C00H (31744 decimal). Control will return to DOSPLUS upon completion of the load. Note that a default extension of /CIM is assumed by the LOAD command because the START parameter indicates a core image file.

PAUSE

This command will pause execution until a key is pressed.

PAUSE [message]

message is an optional message string.

The PAUSE command provides a convenient way to temporarily halt execution of DOSPLUS to give the operator a chance to perform some necessary task. It is generally used inside a DO file. The command may optionally be followed by any string of characters which the user wants displayed at the PAUSE. When the command is executed, the word "PAUSE" will be displayed followed by the string. Execution will then be suspended until the user presses any key on the keyboard. If the BREAK key is pressed, the the DO processing will terminate.

Note that if PAUSE is inside a DO file which is executed with the BREAK key disabled, pressing the BREAK key in response to PAUSE will have no effect. Also note that your command must fit onto a single command line.

Examples:

Suppose a DO file contains the following commands:

```
CLOCK ON
PAUSE Please insert diskette MGPDATA.
LOAD MGP/CIM (S=5500H)
MGP
```

When this DO file is executed, the real-time clock display will first be turned on. Then the PAUSE command will be executed, displaying the line:

```
PAUSE Please insert diskette MGPDATA.
```

At this point execution will be suspended. The user should then insert the proper diskette in a drive and press any key. As soon as he presses any key execution will continue with the next command.

If this DO file was executed with BREAK turned off then all keys EXCEPT the BREAK key could be used to cancel the PAUSE condition. Pressing the BREAK key, however, would not cause execution to proceed.

PROT

This command allows you to change diskette information.

PROT drivespec (param=exp...)

drivespec is the name of the drive that contains the disk we wish to operate with.

(param=exp...) is the optional parameter to be altered.

The parameters are:

PW="string"	Supplies the current disk master password to the system.
MPW="string"	Supplies the new password to the system, if the password is to be changed.
NAME="string"	Specifies the new name for the diskette.
DATE="string"	Specifies the new date for the diskette.
LOCK=switch	Determines whether the disk master password is to be assigned to, or removed from, affected files.
ACC=switch	If LOCK=Y, this will cause the disk master password to be assigned to the ACCESS password of all files. If LOCK=N, this will cause the ACCESS password of all files which have them to be removed.
UPD=switch	If LOCK=Y, this will cause the disk master password to be assigned to the UPDATE password of all files. If LOCK=N, the UPDATE password of all files which have them will be removed.
CLEAN=switch	Specifies whether unused slots in the directory are to be zeroed.

Abbreviations:

PW	P
MPW	M
NAME	N
DATE	D
LOCK	L
ACC	A
UPD	U
CLEAN	C

The PROT command allows you to change diskette attributes which were assigned at FORMAT or BACKUP time. These attributes include the diskette name, date, and master password. In addition, you can use the PROT command to assign the diskette master password to all the files in the diskette directory, or, conversely, remove all passwords from user files (system files will not be affected).

PW. This parameter supplies the diskette's current master password to the system. The password is a string of valid characters enclosed in single or double quotes. Any alphabetic characters in the strings are evaluated in a case-independent fashion, that is, upper and lower case letters are treated equally. To use the PROT command, the diskette's master password must be specified using this parameter unless it is null or nonexistent.

MPW. This parameter assigns a new diskette master password. The password must consist of a string of valid characters enclosed in quotes. Either single or double quotes may be used.

NAME. This parameter assigns a new name to the diskette. The name must be a string of up to eight valid characters enclosed in quotes.

DATE. This parameter allows you to change the diskette date. Normally this date is assigned at FORMAT or BACKUP time, but you may change it using the PROT command. The date may actually be any string up to 8 characters in length which the user wishes to place in this field.

LOCK. This parameter affects the protection status of the files on the diskette. LOCK=Y assigns the disk's master password to the Access and Update passwords of all user files on the diskette (unless used with the ACC and UPD parameters, see below). Conversely, LOCK=N removes all Access and Update passwords from all user files on the diskette. System files (that is, files with a file protection level of 6) are not affected.

ACC and UPD. These parameters are used in conjunction with LOCK to control the assignment or removal of file passwords. For example, if ACC=NO was specified in conjunction with LOCK=Y, then only the UPDATE password of each user file would have the diskette's master password assigned to it. The Access passwords would be left untouched. Similarly, if UPD=NO was specified together with LOCK=N, then only ACCESS passwords would be removed from user files.

CLEAN. This parameter will determine whether unused slots in the diskette directory will be zeroed out or not. Some unused directory slots may contain information pertaining to KILLed files. If the directory slots are zeroed out, then no trace of any killed files would remain, and consequently it would be impossible to attempt the recovery of any killed files.

Examples:

```
PROT :AA (PW="secret",MPW="CIA")
```

This command will change the master password of the diskette in drive :AA from "secret" to "CIA". Note that the case-independent evaluation of alphabetic characters would have allowed you to specify PW="SECRET" or MPW="cia" and still obtain the same results.

```
PROT :5 (LOCK=N,UPD=N)
```

This command will result in the access passwords of all user files being removed. Update passwords, however, would not be touched.

```
PROT :X1 (N="Fiscyr83",D="01.01.83")
```

The name of the diskette in drive :X1 would be changed from whatever it was originally to "FISCYR83", and the diskette date changed to 01.01.83.

```
PROT :K2 (N="New$disk",CLEAN)
```

The diskette in drive :K2 would be renamed to "New$disk" and all unused slots in its directory would be zeroed out.

```
PROT :XX (DATE="KEEPOUT")
```

The DATE field may contain any string, not just the date.

REMOVE

This command will delete a file or group of files from a disk. It will also disable any active devices or drives. This is identical to the KILL command.

==========

REMOVE filespec (param=exp)
REMOVE [FROM] drivespec [USING] wildmask (param=exp)
REMOVE devicespec
REMOVE drivespec

filespec is the name of the file you wish to delete.

wildmask and drivespec are the wildmask and optional drivespec that indicate the group of files that you wish deleted. If the drivespec is omitted, the system drive will be used.

(param=exp...) is an optional parameter affecting the action of the REMOVE command. These apply to killing file only, no devices.

devicespec is the name of a device you wish to deactivate.

drivespec is the name of a drive you wish to remove from the system.

The valid parameters for this command are:

INV=switch	Specifies whether or not invisible files are to be included when a wildcard delete is done.
ECHO=switch	When doing a wildmask delete, this will display the name of each file as it is killed.
SYS=switch	Specifies whether or not system files are to be included in a wildcard delete.
QUERY=switch	This will cause the system to display the filespec and prompt you for a reply before deleting the file.
PW="string"	This parameter declares the DISK master password to the system, which will be used in place of file passwords when wildmasks are specified.

Abbreviations:

INV	I
ECHO	E
SYS	S
QUERY	Q
PW	P

==========

The REMOVE command is used to delete items from the system that are no longer needed or not desired. This may include files, devices, or drives. When deleting files with REMOVE, there are basically two modes of operation: standard and global.

The standard method is invoked by entering the REMOVE command followed by a filespec. This will delete that file from the disk. The global method is invoked by entering the REMOVE command followed by a wildmask. This causes REMOVE to delete all files matching that mask. When using either of these methods, you have available the above listed parameters to modify the manner in which REMOVE works.

When REMOVE is typed with a filespec, but without a drivespec, the system will perform a global search of all mounted disks until it finds the first occurrence of the file, which it will then delete. If a drivespec is supplied, then only that drive will be searched.

When REMOVE is typed with a wildmask, and no drivespec is supplied, the current system drive will be searched. The system will search the directory of the specified drive for the first filename that fits the wildmask, and kill it. It will then continue to search for other files which will fit the mask, killing each one that it finds.

When using a wildmask, the REMOVE command requires that the disk's master password be given with the PW parameter. In this case, the disk master password will be used in place of the file passwords when a password protected file is encountered. This does not apply if the Disk Master Password is not set for that drive. Because the password is set to null (no password), by omitting the password you are in effect specifying the correct password (since "no password" is the password). The fact that anyone knowing the Disk Master Password can delete any file on the disk (system files included), should be sufficient to illustrate the importance of setting the Disk Master Password in DOSPLUS. Our increased use of that password means that for a disk to be at all protected, a password must be set.

INV and SYS. These parameters are used when doing a multiple-file kill using a wildmask. Normally, this type of multiple-file kill includes only visible user files. However these two parameters allow you to include invisible and system files. The INV parameter will include invisible files, and the SYS parameter will include system files in the wildmask search. Remember that system files are also invisible, so to include the system files, you must specify both parameters.

ECHO. This parameter may be used when doing a wildmask search, to display the names of the files as they are deleted. You will find that for the most part, this is a desirable option. It is not often that you want to kill multiple files from a disk and not be told which files are being killed. If you see a filename that you did not mean to be included, you have the option of using the RESTORE utility to recover it. As a rule of thumb, always turn ECHO on during a wildmask kill.

QUERY. This parameter will force the system to display the filename before killing it, and prompt the user for a yes or no reply. The file will be killed only if you specifically reply "Y" when prompted. Pressing ENTER alone will not kill the file. This is most useful when the wildmask you have specified is so general that files will be included that you don't want deleted. In most instances, you would rather spend the extra time answering a prompt than recovering a file killed by accident.

The QUERY parameter will override the ECHO parameter. This means that if the system prompts you as it kills the files, it will not re-display the filename if you say "Y".

PW. This parameter declares the disk's master password to the system. This password will be used instead of the file access and update passwords when a wildmask search encounters a protected file. The disk's master password must be enclosed in either single quotes or double quotes and may be in upper or lower case, or both.

When doing a wildmask REMOVE, this parameter is REQUIRED. The disk master password MUST be specified even though the files are not password protected as long as wildmasks are used.

REMOVE may also be used to disable devices. If a devicespec is given, the a bit will be set in that device's DCB indicating that it is set to NIL. It will become unavailable until it is re-entered in the table by means of the RESET command (see RESET). For example, REMOVE @PR will remove the lineprinter device from the system.

Similarly, disk drives may be disabled with REMOVE. When disk drives are REMOVEed, they are also set to NIL. Simply because a drive is set to NIL does not mean that it was once active and is now removed. If a driver was never installed for a device, then obviously it cannot be recovered.

When I/O is performed to any REMOVEed output device, no error message will be returned; however the data will go nowhere.

When a disk drive is REMOVEed, any attempt to access that drive will return the error message, "Drive not available."

Care should be taken when using REMOVE to disable devices. The only way out of injudicious use of this command may be to reboot (for example, REMOVE @KI will disable the keyboard; the only possible recovery from this case would be to reset the entire system). However, if a device is linked to another, the other device will continue to be active even if the first device is REMOVEed.

REMOVEed devices and disk drives may be restored to an active state by setting the device back to itself, for example, ASSIGN @PR @PR (See ASSIGN).

Examples:

```
REMOVE FIRST/CMD
```

This command will cause the system to search the directories of all mounted disks for the first occurrence of FIRST/CMD, which will then be killed.

```
REMOVE FIRST/CMD:1
```

This command will cause the system to search the directory on drive :1 for FIRST/CMD. If it finds the file, the file will be killed. If the file does not exist on that directory, the system will return a "File not found" error.

```
REMOVE */OLD:4 (QUERY=Y,PW="SUPER")
REMOVE */OLD:4 (QUERY,PW="SUPER")
REMOVE */OLD:4 (Q,P="SUPER")
REMOVE */OLD:4,Q,P="SUPER"
```

The system will search the directory of the disk on drive :4 for every file with the extension /OLD. It will then display the filename that it finds which fits the wildmask and ask the user whether that file is to be killed or not. If the user replies "Y", then the file will be killed. Otherwise the file will be left alone, and the search will continue for other files with the /OLD extension.

```
REMOVE PROG?/BAS:A (PW="PASSWORD")
REMOVE PROG?/BAS:A (P="PASSWORD")
REMOVE PROG?/BAS:A,P="PASSWORD"
```

This command will search the directory on drive A for any filename that fits the wildmask PROG?/BAS and kill them. Files with names such as PROGI/BAS, PROGA/BAS, PROG$/BAS, etc. would be killed. Note that the disk's master password must be supplied. Due to the lack of QUERY and ECHO parameters, you will not be prompted before the file is killed, nor will you see the filename as it is deleted.

```
REMOVE MYDATA/DAT.SECRET:0A
```

This command will remove all traces of the file called MYDATA/DAT.SECRET from the disk in drive :0A.

```
REMOVE :02
```

The disk drive designated as :02 will be disabled. It will be disabled and any attempts to read or write drive :02 will produce an error.

```
REMOVE */*:X1 (QUERY=Y,PW="CIA")
REMOVE */*:X1 (QUERY,PW="CIA")
REMOVE */*:X1 (Q,P="CIA")
REMOVE */*:X1,Q,P="CIA"
```

This form of the REMOVE command is a global REMOVE. Any filename will fit the */* wildmask form, so the use of this command will result in every file on drive :X1 being killed. In this case, QUERY is switched on and the user will be prompted before each file is killed.

```
REMOVE !:0 (PW="MYFILE",QUERY=N,ECHO=Y)
REMOVE !:0 (PW="MYFILE",ECHO)
REMOVE !:0 (P="MYFILE",E)
REMOVE !:0,P="MYFILE",E
```

The ! is a special wildcard character which is the same as the wildcard combination */*. This command would result in every file on drive :00 being killed. The disk password "MYFILE" will be used to access any file encountered which is password protected. The name of each file will be displayed as it is REMOVEed, and the user will not be prompted before a file is REMOVEed. Note that since the parameters default to "Y" if specified and "N" if omitted, the QUERY parameter can be dropped since it is not desired and the expression "=Y" can be dropped from ECHO.

RENAME

This command will permit you to rename devices, disk drives and disk files.

RENAME [FROM] device1/file1 [TO] device2/file2

device1/file1 is the name of the device or disk file being renamed.

device2/file2 is the new name.

The user may rename any device, file or disk drive under the DOSPLUS system. Names must conform to the conventions described in the Operations section of this manual. Briefly, a device name consists of an @-character followed by one or two valid characters; a disk drive name consists of a : (colon) followed by one or two valid characters. A filespec consists of a one to eight character file name, and a one to three character extension preceded by a slash ("/"). A file's password cannot be changed by the RENAME command. However, if a file to be renamed has a password, it still must be entered in order for the RENAME to execute properly.

Duplicate device or file names are not allowed. Wildmask specifications may NOT be used with the RENAME command.

The logical device and/or disk drive names are entered into the system's device table. Thereafter that particular device should be referred to by the new logical name until it is again changed by the RENAME command. New filenames replace the old ones in the diskette directory. If the same filename exists on more than one diskette directory, only the first one is changed if no drivespec is specified.

Examples:

```
RENAME @KI TO @KB
```

This command renames the @KI device to @KB.

```
RENAME :0 :ME
```

This command renames disk drive :0 to :ME.

```
RENAME FOO/BAS TO FOOBAR/BAS
```

This command renames the file called FOO/BAS to FOOBAR/BAS.

RESET

The RESET command will restore a device to its default driver.

==========

RESET
RESET [FROM] devicespec

devicespec is the current logical name of the device to be RESET.

There are no parameters for this command.

==========

The RESET command is used to dissolve any FORCEs or JOINs that happen to be in effect for a device and restore it to its default value. A device's default value could be either the powerup value or an assigned value. RESET will always restore the last value that was in effect.

RESET without any device specification will perform a GLOBAL reset of all devices. If a devicespec is included on the command line, then only that device will be reset. Any linking or routing of the device will be cancelled, and the device will be restored to its normal power-up setting. However, any active translation (that is, the device is FILTERed) will not be affected. Also, the current logical name of the device will NOT be changed.

If a device was linked or routed to a disk file, RESET will close the disk file when the link or route is cancelled.

Examples:

RESET

This command will perform a global reset of all devices. Any devices which were linked or routed will be restored to their powerup condition. Disk files which were the target of linking or routing will be closed.

RESET @PR

This command will restore the @PR device to its normal condition if it had been linked or routed to another device. If no linking or routing had been done, this command would not have any effect.

ROUTE

This command allows you to route the I/O of one DOSPLUS system device to another.

==========

ROUTE
ROUTE [FROM] devicespec [TO] device/file

devicespec is the primary device which is to be routed.

device/file is a device or a file to which the primary device or file is to be routed.

==========

The ROUTE command provides the DOSPLUS user with the ability to redirect the I/O paths of the system's devices. This provides unparalleled operational flexibility with a minimum of effort. With this command, lineprinter output may be sent to the display, or display output sent to a disk file. This avoids the need to rewrite programs in the event that, for example, a lineprinter should fail; all lineprinter output could merely be rerouted to the display, or to a disk file for later printing.

Unlike the JOIN command, which links two I/O devices together so that data goes to the two devices simultaneously, ROUTE actually routes data intended for one device to another device or to a disk file.

Neither "devicespec" nor "device/file" default to anything. If a device or file is specified, then a file must also be specified. If a devicespec is specified, then a device or file must also be specified. If ROUTE is entered without any device specification or channels, then the current device settings will be displayed. It will appear something like this:

```
$00 @KI <-  06CBH
$01 @DO <-> 08C5H
$02 @PR  -> 0BD9H
$03 @RS <-> 0CEAH
$04 @SI <-  NIL
$05 @SO  -> NIL
$06 @U1  -  NIL
$07 @U2  -  NIL
```

The I/O directions of the devices involved in the ROUTE must be the same, that is. input devices may only be linked to other input devices, and output devices to other output devices. Routing an input device to an output device or vice versa, is illegal. For moving data between two devices of dissimilar natures, use the COPY command (see COPY). If a device is routed to itself (i.e. ROUTE @PR TO @PR), then that device is reset that device, that is, any previous routing established will be removed. You may also use the RESET command to remove any routing (see RESET).

Restrictions

(1) Only devices 0-7 can be the primary device. These are the system devices @KI, @DO, @PR, @RS, @SI, and @SO plus the two user-definable devices @U1 and @U2 (remember that these devices may be renamed; if they are, then the current name of the device is the one which should be used). Drives may NOT be specified, either as the primary device or the destination device/file. Drivespecs are valid only with filenames.

(2) Input devices should only be routed to other input devices (or devices capable of simultaneous input and output) and output devices may only be routed to other output devices or disk files. Routing an input device to an output device, or vice versa, is possible but the results are not always predictable.

(3) The order in which the devices are routed is important. Remember that you are ROUTEing the primary device to the destination device or file. For example, if @DO is routed to @PR, the printer output will be sent to the display. Order is important.

(4) When routing an output device to a file, remember that the file will remain open until the device is reset and the ROUTE removed. If the the computer is rebooted without resetting the device, the file may not be readable.

Examples:

```
ROUTE @DO TO @PR
```

This command will send all output normally going to the display to the lineprinter instead. Once this command is given, no new data will appear on the display screen.

```
ROUTE @PR PRINTFIL/TXT:3
```

All output to the lineprinter will be sent to a disk file called PRINTFIL/TXT on drive :3. If PRINTFIL/TXT previously exists, then any data going to the routed @PR device will OVERWRITE the contents of PRINTFIL/TXT. If PRINTFIL/TXT does not previously exist, it will be created on drive :3. The disk file will remain open until the routing is cancelled by means of the RESET command.

```
ROUTE @PR TO @PR
```

This will cancel any active routing or linking for the @PR device, in effect performing a RESET @PR.

RS232

This command allows you to display the current settings of or alter the settings for the serial communications device (RS232).

==========

RS232
RS232 (param=exp...)

(param=exp...) is the optional configuration parameters. If omitted, the current settings will be displayed.

Your parameters are:

BAUD=value	Sets baud rate.
WORD=value	Sets word length.
STOPS=value	Sets number of stop bits.
PARITY=switch	Engages/disengages parity error checking.
EVEN=switch	Configures for even parity.
ODD=switch	Configures for odd parity.
DTR=switch	Sets/resets DTR line.
RTS=switch	Sets/resets RTS line.
BREAK=switch	Sets RS232 break status.

Abbreviations:

BAUD	B
WORD	W
STOPS	S
PARITY	P
EVEN	E
ODD	O
DTR	D
RTS	R
BREAK	BR

==========

The RS232 command is used to control the TRS-80's serial communications device (RS232C). This device is generally used in communications with remote computers or for driving a serial printer. This command will allow you to display and optionally alter any of the settings that are used to control this device.

To display the current settings, type :

RS232

and press ENTER. The current settings for the serial interface will be displayed.

BAUD. This parameter allows you to configure the baud rate for the RS232 to use. This controls the speed of transmission. BAUD is a term used to express speed of data transmission in "bits per second". For example, 300 baud is 300 bits per second. To alter the baud rate, simply set BAUD equal to the desired speed.

Allowable baud rates are : 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, and 19200. Most services are either 300 or 1200 baud.

WORD. This parameter allows you to set the word length to be used. This is controlling the number of bits that make up a data word. During serial communications, many pieces of information are sent, one right after another. In order for the serial drivers to work properly, they must know how many of the bits received are used to make up the actual data word. Other bits received will then be used elsewhere.

Allowable word lengths are : 5, 6, 7, and 8. Seven or 8 bit lengths are usually used for communications since they allow the entire ASCII character set to be transmitted. A word length of 8 would allow any one byte value (0 through 255) to be sent.

STOPS. This parameter determines the number of stop bits that will be used. In asynchronous serial communications, each data word is framed with start and stop bits. These are used to synchronize the start of the data elements. The start bit is normally a single bit set to 0. Following the word are normally 1 or 2 stop bits set to 1. As the data is received, the transition from stop bit to start (1 to 0) signals the beginning of the next data word.

Allowable stop bit values are 1 or 2. All you must know is what the requirements of the service used or peripheral device communicated with are and adjust this accordingly.

PARITY. This parameter allows you to enable or disable parity error checking. The parity bit is an extra bit sent with each data word that indicates how many bits in that word should have been set (1). Parity takes two forms, odd and even. These will be covered with their respective parameters. This parameter simply allows you to configure the system to recognize or ignore that bit as desired.

Parity is either set ON or OFF. Adjust yours to comply with whatever your application demands.

EVEN. This parameter allows you to set the serial device for even parity. Even parity means that if the number of bits set in the data word is odd, then the parity bit will be set so that the total number of bits set in the data word plus the parity bit will be an even number.

This parameter can be turned ON or OFF. EVEN=N is the same thing as ODD=Y. You may set this parameter even if parity checking is not turned on. This simply controls the type of parity check that will be done if it is used.

ODD. This parameter allows you to set the serial device for odd parity. Odd parity means that if the number of bits set in the data word is even, then the parity bit will be set so that the total number of bits set in the data word plus the parity bit will be an odd number.

This parameter can be turned ON or OFF. ODD=N is the same thing as EVEN=Y. You may set this parameter even if parity checking is not turned on. This simply controls the type of parity check that will be done if it is used.

DTR. This parameter allows you to enable or disable the DTR signal. DTR stands for Data Terminal Ready. This is used by many devices such as modems to indicate that you (the terminal) are ready to communicate. This is a logic signal, not transmitted data.

You may set DTR as either ON or OFF. For the most part, this parameter should be left on as many devices will require it so before operating. Simply put, this signal generally indicates that you are ready to send data.

RTS. This parameter allows you to enable or disable the RTS signal. RTS stands for Request to send. This is used by remote devices as an indication that the terminal (you) is ready to receive data. This also is a logic signal rather than transmitted data.

You may set RTS as either ON or OFF. For the most part, you may leave this parameter on as many devices will require it before sending you data. Simply put, this signal generally indicates that you are ready to receive data.

BREAK. This parameter allows you to set the RS232 break condition. The break condition will interrupt all serial communications. It is a special condition that transmits a continuous space as opposed to spaces and marks (pieces of information).

You may set the BREAK parameter to ON or OFF. For the most part, you should leave this off, because while break is engaged the RS232 will not receive or transmit any characters.

Any parameters not specified when you use this command will remain unchanged from their previous value. For example, if you set the baud rate to 1200 and then set the word length to 8 in another statement, the baud rate will still be 1200. It will not revert to its default.

If you wish to alter the default settings, change the settings as desired and save them as part of a configuration file (see SYSTEM).

Examples

```
RS232 (BAUD=300,WORD=7,STOPS=1,PARITY,EVEN,DTR,RTS,BREAK=N)
RS232 (B=300,W=7,S=1,P,E,D,R,BR=N)
RS232,B=300,W=7,S=1,P,E,D,R,BR=N
```

This command will set the serial interface for a baud rate of 300, a word length of 7, one stop bit, even parity, set DTR, set RTS, and turn off break.

```
RS232 (BAUD=1200)
RS232 (B=1200)
RS232,B=1200
```

This command will alter the baud rate to 1200. All other parameters will remain unchanged.

```
RS232
```

This command will display the current RS232 settings.

SCREEN

This command is used to output video data to another device or file.

==========

SCREEN [TO] device/file

device/file is the optional output device or disk file.

There are no parameters for this command.

==========

The SCREEN command will take whatever is on the video display at the time it is issued and send it to a specified output device or a disk file. Normally, the default output device is the lineprinter (@PR). The user may specify other output devices, for example the serial communications device. While the SCREEN command is processing the video output, any other program operations will be suspended.

This command provides you with a convenient way of maintaining copies of screen displays. For example, the SCREEN command can be embedded in BASIC programs, or executed from machine language programs to keep track of user input to particular prompts. It can also be placed at strategic points in user programs to maintain a log of the program's I/O operations.

Examples:

SCREEN

Everything that is on the video display will be sent to the @PR device. Any running program will be temporarily suspended until the operation is completed (or until all the screen data has been loaded into the @PR spool buffer if it is engaged, but NOT until the lineprinter has finished printing).

SCREEN TO SCRNFILE/DAT:0

All characters currently on the video display will be sent to the file called SCRNFILE/DAT on drive 0. If SCRNFILE/DAT does not previously exist, it will be created. If the file already exists, then the screen data will overwrite the previous contents of the file.

SCREEN @RS

Characters on the video display will be output to the serial communications device.

SYSTEM

This command allows you to configure certain aspects of the DOSPLUS system.

SYSTEM
SYSTEM (param=exp...)
SYSTEM [filespec]

(param=exp...) is the optional parameter to be changed.

filespec is the name of the configuration file you wish to create.

The parameters for the SYSTEM command are:

TIME=switch	Enable/disable time prompt.
DATE=switch	Enable/disable date prompt.
LOGO=switch	Enable/disable logo.
BLINK=switch	Enable/disable blinking cursor.
CAPS=switch	Toggle caps mode (upper/lower).
CURSOR=value	Define cursor character.
HIGH=value	Set top of memory pointer.
STEP=value	Set system default drive step rate.
SAVE=switch	Effect permanent change of certain parameters.
PORT=value	Address of port to be output to upon powerup.
MODE=value	One byte value to be output to this port.
EBEEP=switch	Determines whether system will audibly flag errors.
CLICK=switch	Controls keyclick.

Abbreviations:

TIME	T	STEP	S
DATE	D	SAVE	SA
LOGO	L	PORT	P
BLINK	B	MODE	M
CAPS	C	EBEEP	E
CURSOR	CU	CLICK	CL
HIGH	H		

The SYSTEM command is used to set certain parameters regarding your DOSPLUS according to your personal taste. This command allows you to set such items as whether or not you wish to be prompted for the time and date and whether or not you wish to see the logo on powerup.

But perhaps the most important use of this command is the creation of "configuration files". These files are actual programs that can be executed from DOS command level, JCL, or anywhere that you would normally execute a machine language file. When executed, they will restore the configuration of the system to exactly what it was when the file was first made with SYSTEM. It is by using these files that you make permanent alterations to the way your DOSPLUS is configured.

There are three distinct forms of the SYSTEM command. First, you may enter SYSTEM all by itself and receive a display of the currently set memory pointers. Second, you may enter SYSTEM followed by a list of parameters to alter. Third, you may enter SYSTEM followed by a filespec. This filespec will be used for the configuration file.

Mode 1

Simply type :

SYSTEM

and press ENTER. DOSPLUS will display three values.

LOW$. This is the address currently defined as the bottom of free memory. User programs should not load in at an address lower than this value.

HIGH$. This is the address currently defined as the top of available memory. User programs should never use memory above this address. All DOSPLUS utilities and library commands will honor this address. You may alter this with the HIGH parameter.

TOP$. This is the address that indicates the top of actual memory. This value will never change. By subtracting HIGH$ from TOP$, you may calculate exactly how many bytes of high memory are being used at any one time.

LLOW$. This is the address that indicates the bottom of what is called "resident memory". This is an area of low memory between the operating system and the start of user memory that may be used either by the DOS or user programs.

LHIGH$. This is the address of the top of the region of memory called "resident memory".

BANKS. This parameter indicates the status of the additional 64K (if present). If you have a 64K machine, this parameter will not appear. If you have a 128K machine, you will have this parameter. BANK will be followed by two numbers: "1" and "2". Each of these will be followed by a character indicating the status of that 32K bank. An asterisk (*) means that it is available and a plus sign (+) indicates that it is in use.

For example, BANK 1* 2* indicates that both are free. BANK 1+ 2* means that the first bank is used.

All of these values (with the exception of BANK) will be displayed in hexadecimal format.

Mode 2

In this mode, you will use SYSTEM to set certain custom parameters regarding your DOSPLUS. Certain of these parameters will take effect at once, and some will require that you reset the machine before they take effect. Certain of them will automatically permanently configure the disk and others of them will require that you specify an additional parameter if you want to make the change permanent.

TIME. This parameter allows you to turn the time prompt ON or OFF. When DOSPLUS boots up, one of the items you will be prompted for is the time of day. If you do not wish to see this prompt, use TIME=N as a parameter. This parameter will take effect at once and is a permanent change. You may turn it back on at any time you desire. Do not write protect the drive before using this or any other parameter that needs to write to the disk.

DATE. This parameter allows you to do the same thing with the date prompt. Normally, DOSPLUS will ask you for the date each time it boots up. This parameter allows you to turn that off if you so desire. If you turn off the date prompt, DOSPLUS will automatically attempt to preserve the date when the system is rebooted. This parameter takes effect at once and does not require the use of the SAVE parameter.

LOGO. This parameter allows you to turn ON and OFF the DOSPLUS logo that is displayed on powerup. This also takes effect at once and does not require you to specify SAVE to make it permanent.

BLINK. This parameter allows you to engage or disengage the cursor blink function on DOSPLUS. The Model III supports this both with the standard keyboard driver and with the alternate keyboard driver supplied from us. The Model I requires that you load the alternate keyboard driver before the cursor will blink regardless of the status of this parameter. This will take effect at once but does not become a permanent change unless you specify the SAVE parameter. This allows you to turn blink ON and OFF without permanently configuring your system.

CAPS. This parameter allows you to toggle between upper and lower case. This has the same immediate effect as pressing the <SHIFT> and <0> keys simultaneously (on the Model I, this is only true if the alternate keyboard driver is installed). This parameter allows you to toggle back and forth under software control. By using the SAVE parameter, the current CAPS status will become the default powerup condition. This applies even if you have not specified the CAPS parameter. Any time that you specify the SAVE parameter on SYSTEM, the current CAPS status (as of that moment) becomes the default powerup condition.

CURSOR. This parameter allows you to change the cursor character to any one byte value. Simply specify CURSOR=value, where "value" is the byte either in decimal or hex that represents the character you wish to use for your cursor. The change will take effect at once, but is not permanent unless you specify the SAVE parameter. This parameter will only function on the Model I if you have installed the alternate keyboard driver.

HIGH. This parameter allows you to alter the address that DOSPLUS regards as the top of available memory. DOSPLUS will use high memory for some of its alternate drivers and filters. This area of memory should not be corrupted by the user for any reason. You may, however, have some programs that also load into high memory but do not adjust the high memory pointer to reflect their location. DOSPLUS will then use this area of memory if it needs it, thereby corrupting your program. By using this parameter, you may adjust the top of memory pointer downward to protect your programs and DOSPLUS will not use that area of memory.

All of the DOSPLUS drivers and filters are self relocating and will honor this value. Ideally, all programs should automatically adjust the top of memory pointer, but for those that don't you may use this parameter. This parameter is not saved to the disk permanently either automatically or with the SAVE parameter. If you wish to consistently change this value, save it as part of a configuration file.

STEP. This parameter sets the system default step rates. DOSPLUS is supplied stepping the disk drives at the lowest possible rate so that it will function with any brand of aftermarket drive. However the standard Radio Shack drives as well as many of the aftermarket units are capable of stepping faster than this. To avoid having to force you to use a configuration file just to alter your drive step rates, we have included this parameter.

This parameter only affects the overall default step rate. To change the step rate of any individual drive (e.g. you have one drive that needs to step slower than the rest), you must still use CONFIG and store it in a configuration file. To use this parameter, set it equal to one of the following values :

Value	Step rate
0	6 mS
1	12 mS
2	20 mS
3	30 mS (double density)
	40 mS (single density)

The STEP parameter will be saved at once to the disk, no need to use the SAVE parameter. However, the new step rate will not be in effect until you reboot. For standard Radio Shack disk drives on the Model III, you should be able to use a step rate of 0 (6 mS). For Model I drives, you may use 2 (20 mS). Some Model I drives will go faster, but all will work with a value of 2.

Technical note: For any eight inch disk drives, the step rate will be one half that listed in the table. Also, please confine yourself to these values listed. Using other values may cause the floppy disk controller to operate incorrectly. Remember, these are relative values, not the actual step rate.

SAVE. This parameter allows you to make permanent certain of the SYSTEM parameters that would not otherwise be so. Specifically, the parameters BLINK, CAPS, and CURSOR require that you use this parameter to make them permanent. To use it, simply include the parameter in the command line. The status of the three above mentioned parameters will be saved as they are at that time.

PORT. This parameter allows you to set the port that DOSPLUS will output to on powerup. Many clock speed modification kits and other products (i.e. LNW80 microcomputers) will require a value to be output to a port to engage certain functions. You may have DOSPLUS do this automatically if you choose. This parameter allows you to set which port receives this output. This parameter will be automatically altered on the disk without the use of the SAVE parameter.

MODE. This parameter allows you to set what value will be output to the port on powerup. Any one byte value may be used. The value may be expressed in decimal or hex format. When you set MODE, it will be saved on the disk automatically. To disengage MODE, set it to 0.

EBEEP. This parameter will instruct DOSPLUS to emit an audible tone whenever a DOS error occurs. This parameter will require the SAVE parameter to become permanent.

CLICK. This parameter cause DOSPLUS to emit an audible "click" when each key is pressed. If you wish to make this a permanent option, use the SAVE parameter.

Mode 3

This form of the SYSTEM command allows you to create configuration files. These files are used to permanently store your custom configurations. Please note that if you have adjusted any of the above parameters, even permanently, a configuration file will override them. In other words, if you set the cursor to a graphics block, and when a configuration file is loaded that had an underline cursor when it was saved, an underline cursor will be in effect.

It is very important that before you understand the method of creating configuration files with SYSTEM, you understand what these files are and why you use them. Throughout the DOSPLUS system, there are commands such as CONFIG, FORMS, and RS232 that allow you to alter parameters affecting system operation.

In addition, many applications require that alternate drivers be loaded or filters be installed. Before using the Job Control Language, it must also be loaded into memory. For all of these applications, when you reset the system these areas return to their default powerup conditions.

This is the reason that we have provided you the ability to create these configuration files. They are used to preserve these special configurations. Let's take an example.

Assume that we have assigned the alternate display and keyboard drivers for our Model I so that we may have lower case and the advanced keyboard features. In addition, we have configured drive 2 to step at 6 mS and altered the default RS232 parameters. In short, we have customized our DOSPLUS in the manner that best suits operation on our particular machine.

Now we wish to preserve this. To accomplish that end, we might use the statement :

SYSTEM MOD1:0

This command would create the file MOD1/CFG on drive 0. This file would contain all of the drivers we had assigned and a record of the current system configurations in all user definable areas. Note the use of the default extension "/CFG". This is to identify the configuration files. You may use anything you wish.

Now, after rebooting the system, to load the drivers we had assigned and instantly return all the items we had configured to the values we set them to, all we have to do is execute the file MOD1/CFG.

This can be done in more than one manner. The configuration files is an executable program. You may enter the filename at the DOS command level, set it on an AUTO, or use any other method appropriate to executing a machine language program.

Important: There is one restriction with this. When executing a configuration file in a multiple command line or with AUTO, the name of the configuration file must be the first item on the line. To do otherwise will produce some rather annoying results.

Let's take another example. Assume that we have attached a five megabyte hard disk to our computer. We have used ASSIGN to install the driver, CONFIG to adjust the parameters, formatted the drive, and performed all desired operations upon it. In addition, we have transferred the system files to the hard disk with SYSGEN and used CONFIG to move control to the hard disk. Then, also using CONFIG, we re-order our drives such that the various volumes of the hard disk are searched first and the floppies second.

Once the system is set up exactly the way that we want it, all drivers installed and all parameters configured, we might use the statement :

SYSTEM RIGID/CMD

This command would create the file RIGID/CMD on the first available disk drive. This file, when executed, would load the drivers needed and send the system control back to the hard disk.

Note: If the first available drive was the first volume of the hard disk, as it would have been in our example case, the file will be on the hard disk. Please copy it to a floppy disk for the purposes of booting up. Without the file to re-configure for the hard disk, we will have to re-do all the work we just did and will defeat the purpose of the configuration files.

Upon rebooting the system, we would come to the DOS command level and execute the file RIGID/CMD. This file would instantly reload all needed drivers and move the system control to the hard disk as it was when the file was created.

Applications of configuration files

These are far too numerous to go into great detail, but we will address just a few:

(1) The fact that you may create as many of these files as you wish and store them all on the same disk means that several people can use the same copy of DOSPLUS (or the same rigid disk) and configure the system to suit them just by loading in the proper file.

(2) The same fact also makes it possible for the user to have more than one machine with differing numbers of drives and various kinds of peripherals for each. After creating a configuration file for each machine, the user may boot the same system disk in all machines and assume the needed configurations by simply executing the file.

(3) It is also convenient to use the configuration files to remove drivers or filters that are assigned as temporary measures. When one of these files is executed, a true "warm-start" is performed. The system is reset to exactly the same conditions as existed when the file was created. If you have loaded any other drivers or altered parameters in the meantime, these will also be removed from the system or reset to their whatever values are stored for them. This is especially useful in removing programs such as JCL or filters no longer needed.

Examples:

```
SYSTEM
```

This command will display the currently defined memory pointer addresses.

```
SYSTEM (TIME=N,LOGO=N)
SYSTEM (T=N,L=N)
SYSTEM,T=N,L=N
```

This command will turn off the time prompt and the DOSPLUS logo. These items will no longer appear on powerup or reboot.

```
SYSTEM (BLINK=N,CURSOR=140)
SYSTEM (B=N,CU=140)
SYSTEM,B=N,CU=140
```

This command will set a steady block cursor. If we wanted to make this a permanent change, we would have included the SAVE parameter (i.e. SYSTEM,B=N,CU=140,SA). As it is, this command will only be temporary in its effect.

```
SYSTEM MYFILE
```

This command will create the configuration file MYFILE/CFG on the first available disk drive, saving all system parameters to the currently set values along with any drivers or filters that are loaded. To resume this configuration scheme, all that is needed is to execute the file MYFILE/CFG.

```
SYSTEM (EBEEP=Y,CLICK=Y,SAVE)
SYSTEM (E,CL,SA)
SYSTEM,E,CL,SA
```

This command will engage the audible error posting and keyclick and make these a permanent option.

TIME

This command will allow you to set or display the time in the system's real-time clock.

==========

TIME
TIME hh:mm:ss

==========

This command is used to display the time currently in the system's real time clock or to set this time. The time, if displayed, will be in the "HH:MM:SS" format. You may set the time in free form format. To display the current time, specify "TIME" without any accompanying time and press ENTER.

When setting the clock with the TIME command, the time can be specified in a variety of ways. Allowable separators are any non-numeric character. This flexibility allows you to specify the time in whatever format is most comfortable to you.

The time is maintained by the system in 24-hour format. That is, the hours go from 0 to 23. Midnight is 00:00:00, and one p.m. is 13:00:00.

Examples:

```
TIME 3:5:30
TIME 03:05:30
TIME 3-05-30
TIME 03 5.30
TIME 3.05/30
```

All of the above are equivalent and set the system's clock to 03:05:30.

```
TIME 9.00
TIME 9
```

The system clock is set to nine o'clock. If minutes and seconds are not specified, they default to 00.

```
TIME
```

DOSPLUS will print the current time on the video screen.

VERIFY

The VERIFY command causes DOSPLUS to read back whatever is written onto the disk in order to verify that it was written correctly.

==========

VERIFY [param]

param is the optional status switch.

Your switches are:

ON	Engage verify.
OFF	Disengage verify.

==========

This command will enable automatic read-after-write on all disk I/O. It will ensure that data written to the disk can be read back without error. This will slow disk I/O down slightly but might be desirable when writing critical data to the disk.

When VERIFY is engaged, even utilities such as CONVERT and library commands such as COPY will verify what they write. Any user software that uses standard DOS file I/O calls to write to the disk will also be forced to verify what it writes.

Examples:

```
VERIFY
VERIFY YES
VERIFY Y
VERIFY ON
```

These forms of the VERIFY command are all equivalent and enable the read-after-write function. If VERIFY is already on, then these commands will have no effect.

```
VERIFY OFF
VERIFY N
VERIFY NO
```

These forms of the VERIFY command will turn off the read-after-write function. If the function was already disabled, then these commands would have no effect.

DOSPLUS IV Utility programs

The following is the list of DOSPLUS IV Utility programs:

Utility	Description	Page #
BACKUP	(Duplicates floppy diskettes)	3-1
CONVERT	(Move files from Model III TRSDOS to DOSPLUS IV)	3-6
DIRCHECK	(Verifies and/or repairs diskettes directories)	3-11
DISKDUMP	(File oriented sector display/modify utility)	3-13
DISKZAP	(Track oriented sector display/modify utility)	3-16
FORMAT	(Initialize floppy diskettes)	3-28
HELP	(Display abbreviated help listing)	3-33
MAP	(Locate disk files on disk)	3-34
PATCH	(Install patches to machine language programs)	3-36
RESTORE	(Restore KILLed or RESTOREed)	3-39
SYSGEN	(Create DOSPLUS system diskettes from data disks)	3-40
TRAP	(Intercept certain disk I/O errors)	3-42
MEDIC	(Menu driven, user friendly DOS interface)	3-43

BACKUP-

This program is used to copy all data from one floppy disk to another. It will make exact or "mirror-image" copies only. To use a "copy by file" method to backup your disk, use the library command COPY. You will also use COPY to backup the hard disk.

==========

BACKUP [FROM] :sd [TO] :dd (param=exp...)

:sd is the drive that you will be copying FROM. If this information is not provided in the command line, BACKUP will prompt you for it later.

:dd is the drive that you will be copying TO. As above, if this is not specified in the command line, it will be prompted for.

(param=exp...) is the optional parameter that modifies what action the parameter takes.

Your parameters are:

DATE="string"	Allows you to set the date directly from the command line when you are aware that the system date is NOT set and you do not wish to be prompted.
USE="string"	Allows you to indicate that you wish to over-write any existing format on the destination disk WITHOUT being prompted during the backup. Answer with "Y" to procede with the BACKUP, or "N" to abort. Use "F" to reformat the diskette.
VERIFY=switch	Allows you to indicate from the command line that you do not wish to verify the data on the destination disk.

Abbreviations:

DATE	D
USE	U
VERIFY	V

==========

This utility enables you to backup (e.g. duplicate) your floppy diskettes. It is recommended as a good computing practice to use this utility to make frequent copies of your important data diskettes.

With DOSPLUS's BACKUP utility, it is not necessary to pre-format your destination diskettes. If the diskette is blank, DOSPLUS will format it automatically. Even if the diskette was previously formatted, DOSPLUS will offer you the chance to format it again before using it in the backup.

BACKUP allows you to optionally specify the source and destination drive from the command line using the syntax shown above. You, of course, do not need the FROM and TO delimiters unless you are specifying the destination drive first or only. BACKUP will assume that the first drivespec encountered is the source drive unless it finds a TO delimiter. It will likewise assume the second drivespec encountered to be the destination drive unless it encounters a FROM delimiter.

If you do not specify the source and destination drives at the command line, BACKUP will prompt you for them. If you specify one without the other, BACKUP will prompt for the one that is missing.

Also, in addition to specifying the source and destination drives from the command line, you have the option of specifying the date, whether or not you wish to use the disk if it contains data, and whether or not to verify the data.

DATE. To set the date, all you must do is use the statement "DATE=string", where "string" is a quoted string up to eight characters in length. When inputting the backup date, either with this line or in response to the prompt, you are not limited to numeric input. If you do NOT specify the date from the command line, DOSPLUS will use the current system date (if it is a valid date). If the current system date is not valid, DOSPLUS will prompt for the date.

USE. To implement the "Use" parameter, simply type "USE='Y'", "USE='N'", or "USE='F'" in the parameter list (depending on the desired action). This will inform BACKUP whether or not you wish to be prompted before over-writing a disk that already contains data. If you have specified "F", not only will DOSPLUS not prompt you, but it will re-format the destination disk before proceeding.

VERIFY. This parameter allows you to signal BACKUP not to verify the data on the destination diskette. Normally, BACKUP cycles through a loop of reading from the source disk, writing to the destination disk, and then verifying the destination disk to make certain that the data was copied correctly. By specifying "VERIFY=N" in the command line, you may defeat this.

Note: This will greatly increase the speed of your backup, but it is NOT a recommended practice except for certain rare and special instances. Micro-Systems Software in no way encourages you to ever make a backup without verification. This parameter has been provided to make it possible to skip the verification, but doing so will in no way guarantee the integrity of the backup copy. Use the parameter ONLY as a last resort. Also note that this will not cause BACKUP to ignore source disk read errors, you must use the TRAP utility (see TRAP) for that.

You may, if you wish, operate BACKUP from within a DO file. This can allow you to use BACKUP as a menu option from a BASIC program and then return to the menu. The procedure is:

(1) Have the first statement of the DO file exit BASIC and return to DOS.

(2) Execute the BACKUP.

(3) Have the DO file re-load BASIC and run your menu.

This is made simpler by virtue of the fact that you can specify all information needed for BACKUP right from a command line. True "hands off" operation. The computer operator doesn't even need to respond to a "Diskette contains data" prompt.

Prompting messages

If you have not answered the source drive, destination drive, and date questions from the command line, BACKUP will prompt for that information. In the following paragraphs we will discuss these. Note that VERIFY must be used from the command line. BACKUP will not prompt for that.

Source drivespec ?

Reply to this question with the drivespec of the drive that contains the disk you wish to backup. Do not include the colon (:). It is only necessary to provide BACKUP with the one or two character drive name.

Destination drivespec ?

Reply to this question with the drivespec of the drive that contains the disk you wish to backup to. As above, you do not need to include the colon. This drivespec may be the same as the source drivespec if you wish to execute a single drive backup.

Backup date (MM/DD/YY) ?

If the system date has not been set and you have not entered it from the command line, BACKUP will prompt you for the date. When it does, you have three options :

(1) Press BREAK and abort the backup.

(2) Press ENTER and default to a date of "00/00/00".

(3) Type in up to any eight ASCII characters you wish for the date and press ENTER. You are not restricted to numeric characters.

If the diskette is not blank and you have not specified the "Use" parameter from the command line, you will receive the prompt :

Diskette contains data, Use or not ?

You may reply in one of three ways to this prompt :

(1) Press BREAK and abort the backup.

(2) Type "Y" or "U" and press ENTER. This will cause BACKUP to attempt to use the existing format.

(3) Type "F" and press ENTER. This will cause BACKUP to re-format the destination disk first.

Once all of these questions have been answered, BACKUP will proceed with the copy of the disk. The destination disk will bear the same name and Disk Master Password as the source disk. The date on the destination disk will be either the current system date or whatever characters you entered when prompted.

Single drive vs. Multiple drive

If the source and destination drivespec are not the same (in other words, you are backing up between two separate disk drives), BACKUP will proceed with the copy after all information has been provided with no further operator intervention.

If, on the other hand, the source and destination drivespecs are identical (in other words, a single drive backup), BACKUP will procced with the copy but will prompt you for the source, destination, and system disks as they are needed.

Pay close attention to these prompts and insert the proper diskette. If you were to accidentally insert the wrong disk at the wrong time, you could corrupt the data.

Note that BACKUP will not backup between two disks of dissimilar format. For example, you can't backup a single sided to disk to a double sided one or vice versa. For those applications, you should use the COPY command to perform a "copy by file" type of backup. BACKUP also will not backup between rigid and floppy disk drives.

As BACKUP is making the backup, it will ONLY copy those cylinders that have allocated data on them and it will ONLY copy as much data from each cylinder as it contains. Do not be alarmed if you see BACKUP skip several cylinders or it BACKUP seems to copy some cylinders faster than others. If you wish to verify, make note of the cylinders at which this occurs. Then use the library command FREE to display a free space "map" of that disk. The cylinders skipped should show no "x"s at all and cylinders that seemed to backup faster than others should have open space (i.e. not solid "x"s). (See the library command FREE)

BACKUP attempts to make "mirror-image" copies of the source disk. If it cannot for any reason do this (a granule allocated on the source disk is locked out on the destination), BACKUP will report an error and abort to the DOS command mode. You may at that time either re-format the destination disk and try again or resort to a "copy by file" backup.

Important note: After the "Diskette contains data" prompt is on the screen, you may NOT switch the source disk. This will cause incorrect information to be written to the destination disk that will later corrupt data. You may switch the destination disk at that time, if you wish.

Obviously, if you are going to invoke BACKUP with all questions answered from the command line, you had better have the disks to be backed up all mounted and ready. This is doubly true if you have specifed the "Use" parameter.

As a rule of thumb, if you are going to backup two disks that are not currently mounted and ready to go it is best to just type "BACKUP" and allow the program to load and ask you all needed questions. Once the program is loaded, you may remove all disks and proceed. It will tell you when it needs a system disk again.

Examples:

```
BACKUP
```

This will execute the backup program and have it prompt for all information.

```
BACKUP FROM :0 TO :1
BACKUP TO :1 FROM :0
BACKUP :0 :1
```

These three examples are all equivalent. They instruct the BACKUP program to backup the disk in Drive 0 to the disk in Drive 1. Note that if you ARE going to use the drive specifiers from the command line, you will need to hae both disks (source and destination) in place before executing BACKUP.

```
BACKUP FROM :0 TO :1 (DATE="Sept 24",USE="Y")
BACKUP :0 :1 (D="Sept 24",U="Y")
BACKUP :0 :1,D='Sept 24',U='Y'
```

All of these commands will acoomplish the same results. They will backup the disk from Drive 0 to the disk in Drive 1. They will set the backup date to "Sept 24" (note the use of non-numeric characters) and instruct BACKUP to use the destination disk even if it contains data.

```
BACKUP :0 :1 (VERIFY=N)
BACKUP :0 :1 (V=N)
BACKUP :0 :1,V=N
```

This command will inform BACKUP that you wish to copy from the disk in Drive 0 to the disk in Drive 1. BACKUP will not verify the copy.

CONVERT

The CONVERT utility has several uses:

(1) To allow DOSPLUS to display a file catalog of Model III TRSDOS diskettes.

(2) To allow DOSPLUS to copy program and data files from Model III TRSDOS systems onto DOSPLUS compatible diskettes.

(3) To make Model I single density disks readable to Model 4 DOSPLUS.

CONVERT [FROM] :sd [TO] :dd [USING] wildmask (param=exp...)
CONVERT [FROM] :dr

Your parameters are:

CAT=switch	Instructs CONVERT to display a file catalog of a Model III TRSDOS diskette.
DIR=switch	Same as CAT, above.
ECHO=switch	Instructs CONVERT to display each filename as the file is copied onto DOSPLUS compatible media.
INVIS=switch	Allows CONVERT to operate on invisible files as well as visible files.
SYSTEM=switch	Allows CONVERT to operate on system files as well as non-system files.
OVER=switch	Forces CONVERT to query the user whether to overwrite a file which already exists.
QUERY=switch	Forces CONVERT to query the user before copying files from TRSDOS to DOSPLUS.
V12	Informs CONVERT that the Model III TRSDOS to be operated upon is a TRSDOS version 1.2 or earlier.
V13	Informs CONVERT that the Model III TRSDOS to be operated upon is a TRSDOS version 1.3.

Abbreviations:

CAT	C	SYSTEM	S
DIR	D	OVER	O
ECHO	E	QUERY	Q
INVIS	I		

The CONVERT utility is only concerned with diskettes formatted by either Model I or Model III TRSDOS. Model 4 TRSDOS disks can be read by DOSPLUS with no conversion or special commands. With all functions of the CONVERT utility, a system disk MUST be present. Since we list three applications for the CONVERT program, we shall cover each of them in turn:

Reading the directory of a Model III TRSDOS diskette

This is accomplished by using either of the first two parameters (i.e. CAT or DIR). The general syntax is:

```
CONVERT :td,C
   or
CONVERT :td,D
```

where ":td" is the drivespec of the disk drive containing the Model III TRSDOS diskette. Both of these commands will have an indentical effect, that is, both of them will display a file catalog (e.g. filename/ext) of the disk.

Copying files from Model III TRSDOS

Using CONVERT to copy files from Model III TRSDOS requires two disk drives; one to hold the double-density TRSDOS, and another to hold a DOSPLUS compatible diskette onto which the files are to be copied. Of course, a DOSPLUS system diskette must be present in drive 0 at all times.

The general syntax for copying files from double-density TRSDOS is as follows:

CONVERT [FROM] :sd [TO] :dd [USING] wildmask (param=exp...)

where ":sd" is the source drive containing the Model III TRSDOS diskette, ":dd" is the destination drive containing DOSPLUS compatible media, and "wildmask" is a valid DOSPLUS wildcard specification. Consider the following examples:

Command	Action
CONVERT FROM :1 TO :0 USING */CMD	Copies all /CMD files on the Model III TRSDOS diskette in drive 1 onto drive 0.
CONVERT :1 :0	Copy all files from the Model III TRSDOS diskette in drive 1 to drive 0.
CONVERT */TXT:3 :1	Copies all /TXT files from drive 3 onto drive 1.

When using this form of CONVERT, you have many parameters available to you.

ECHO. This parameter is used to instruct the CONVERT utility to echo, or display, the name o each file which it copies from TRSDOS to DOSPLUS as the file is copied. This is especially useful when CONVERTing the entire contents of a diskette, or during a CONVERT on a class of files.

```
CONVERT :2 :1,E
```

INVIS. This parameter is used to tell the CONVERT utility to CONVERT files that are invisible on Model III TRSDOS as well as files which are visible.

SYSTEM. This parameter is used to inform CONVERT to copy files which have the system file attribute (regardless of the extension) as well as non-system files.

OVER. This parameter, when specified in the CONVERT command line, will cause the CONVERT utility to query the user whether or not a file should be copied if a file already exists on the destination diskette. For instance, assume that a DOSPLUS system diskette is placed in drive 0, and a TRSDOS system diskette is in drive 1. The following command is executed:

```
CONVERT :1 :0,S,I
```

This command will cause CONVERT to copy all files on the TRSDOS system diskette onto the DOSPLUS diskette in drive 0. Unfortunately, there may be some files on the TRSDOS diskette that have the same name as files on the DOSPLUS diskette - BASIC, for example. If the above command were given, the program BASIC/CMD on the TRSDOS diskette would be copied over the file BASIC/CMD already present on the DOSPLUS diskette, destroying the DOSPLUS BASIC. The OVER parameter can prevent this from happening. Consider the command:

```
CONVERT :1 :0,S,I,O
```

This command line now contains the O, or OVER, switch. Now, when the CONVERT utility encounters a file which already exists on the destination diskette (like BASIC/CMD in the example), CONVERT will query the operator with the question:

```
Overwrite?
```

The operator should answer the question with a "Y" (for yes) or an "N" (for no). If the ENTER key is pressed, CONVERT will assume the reply is "N". If the operator replies in the affirmative, CONVERT will proceed to copy the file onto DOSPLUS. If the operator answers with an "N" (or by pressing ENTER), CONVERT will skip to the next file on the diskette.

QUERY. This parameterr may be used to make CONVERT ask the operator whether each file affected by the CONVERT command should be copied onto the destination diskette. For example, if the command:

```
CONVERT */CMD:1 :2,Q
```

were given, CONVERT would attempt to copy each with the /CMD extension from drive 1 onto drive 2. Before each file is copied, CONVERT will query the operator with the question:

```
filename/ext                Convert?
```

where "filename/ext" is the name of the file to be CONVERTed. If the operator responds with a "Y", the conversion will take place. If the response is an "N" (or if only ENTER is pressed), CONVERT will not copy the file onto the destination diskette and will skip to the next file.

V12 and V13. Two general types of Model III TRSDOS are in existence at the time of this writing: TRSDOS 1.2 and earlier, and TRSDOS 1.3. When using the CONVERT utility, you must inform it of which type of TRSDOS diskette is to be CONVERTed. CONVERT will assume version 1.3 unless otherwise specified. Therefore, the command:

```
CONVERT AR!:3 :0
```

will copy all files beginning withhe letters "AR" on the TRSDOS 1.3 diskette in drive 3 to drive 0. If the diskette in drive 3 were a TRSDOS version 1.2 or 1.1, the following command should be given:

```
CONVERT AR!:3 :0,V12
```

Note that when working with CONVERT, you never actually specify a filename, but rather wildmasks. To convert just a single file from Model III TRSDOS, simply make the wildmask so specific that only that file will match it. For example, to convert just the file AR/DAT, use the command:

```
CONVERT :1 :0 AR/DAT
   or
CONVERT AR/DAT!:1 :0
```

Notice that in the second example, we were forced to append an exclamation mark (!) to the filename. With the wildmask in the space normally reserved for the source drivespec, we must either include the USING delimiter (i.e. CONVERT USING AR/DAT :1 :0), or include something to let CONVERT know that it is wildmask. If we do not, CONVERT will find an illegal source drivespec and abort. The wildcard character (!) alerts CONVERT to the wildmask.

Making Model I diskettes readable to the Model 4

The third major purpose of the CONVERT utility is to render diskettes created on single density Model I's readable on the Model III. Single density Model I diskettes have the directory track recorded in a manner that is unacceptable to the Model III. The CONVERT utility can be used to alter the directory track of such single density diskettes such that they are useable on the Model III. The general syntax for this operation is:

CONVERT :dr

where ":dr" is the drive specification of the disk drive containing the Model I single-density diskette to be CONVERTed. Note that this form of CONVERT requires only one disk drive; if the target drive specified is the system drive, CONVERT will prompt the operator to insert the target and system diskettes as needed.

This form of CONVERT does not copy files from one diskette to another; rather, it alters the target diskette in order to make it readable on the Model 4.

NOTE: After performing this form of CONVERT, certain Model I operating systems (such as Model I TRSDOS 2.3) may not read the target diskette due to the alterations to the diskette directory. Note that Model I DOSPLUS will read the diskette normally, as will most operating systems currently available for the Model I. If it is necessary to subsequently use such a CONVERTed diskette under an operating system which will not read the diskette, it is possible to reverse the alterations to the diskette directory. Assuming that you have access to a double density Model I system equipped with DOSPLUS 3.4 or later, re-CONVERT the diskette using the same syntax as above:

CONVERT :dr

where ":dr" is the drivespec of the disk drive containing the target diskette. The Model I CONVERT program will restore the directory to its original state, an allow any Model I DOS to read the diskette normally.

DIRCHECK

This utility is used to check the integrity of a diskette's file directory, and optionally, to repair certain types of damage to the directory.

==========

DIRCHECK [:dr] [TO] file/@device (param=exp)

":dr" is the drive specification of the disk drive containing the diskette whose directory is to be examined

"file/@device" is the optional output file or device to which all messages concerning the state of the diskette's directory are routed.

Your parameters are:

FILES	Instructs DIRCHECK to repair faulty file directory entries, if any.
GAT	Instructs DIRCHECK to repair a faulty Granule Allocation Table, if necessary.
HIT	Instructs DIRCHECK to repair a faulty Hash Index Table, if necessary.

Abbreviations:

FILES	F
GAT	G
HIT	H

==========

The DIRCHECK utility may be used to automatically examine a diskette directory and report any errors or inconsistencies within the directory. The simplest form of the DIRCHECK command is:

DIRCHECK :dr

where ":dr" is the drive specification of a disk drive containing the diskette whose directory is to be examined. DIRCHECK will read the diskette directory and display a list of any errors found on the video display. After the list of errors, if any, DIRCHECK will print "DIRCHECK complete, xxx total errors", where "xxx" is the number of directory errors found by the DIRCHECK program.

Also note that if you do not specify any options upon entering DIRCHECK, it will prompt you with an asterisk. You may at that time enter all needed options. Pressing BREAK will return you to DOS.

The list of errors may also be directed to any other character-oriented device, or to a file, by specifying an optional output channel. For example, the DIRCHECK command:

```
DIRCHECK :2 TO @PR
```

will output the list of directory errors to the lineprinter.

```
DIRCHECK :0 ERRORS/TXT
```

will output the list of directory errors to the file ERRORS/TXT. Note that the TO was omitted in this example, it is optional.

DIRCHECK is also capable of repairing certain directory errors. The parameters FILES, GAT,and HIT are used to inform DIRCHECK of which portion(s) of the diskette directory should be repaired. If the FILES parameter is specified, DIRCHECK will repair any errors in the file entry table of the diskette directory. Likewise, if the GAT or HIT parameters are provided, DIRCHECK will fix any errors encountered in the respective table. If any of the parameters are omitted, DIRCHECK will report, but will not repair, any errors discovered in the respective area of the directory.

To specify one of the fix options, include the parameter in parenthesis after specifying the drive number or, if it exists, the optional output file/device. Therefore, to instruct DIRCHECK to repair the GAT, you would use the following syntax:

```
DIRCHECK :2 (GAT)
```

if an output field is specified:

```
DIRCHECK :2 TO @PR (GAT)
```

Note that DIRCHECK (or any "directory fixing" program) is incapable of repairing certain directory discrepancies, listed below:

Error	Possible cure
Locked gran assigned to file	Kill offending file
Granule multiply assigned	Kill offending file
Granule assigned past cyl count	Kill offending file
BOOT/SYS not found	Restore BOOT/SYS file
BOOT/SYS not assigned space	Restore BOOT/SYS file
DIR/SYS not found	Restore DIR/SYS file
DIR/SYS not assigned space	Restore DIR/SYS file

Also note that although many directory errors can be repaired by DIRCHECK, it is possible that certain files whose directory information was in error may be adversely affected. In other words, if a directory has bad information in it and DIRCHECK repairs it to the best of its ability, it may cause one file to have its data area lost in order to preserve the integrity of the entire directory. Sometimes the errors are simply too severe to correct without unwelcome side effects.

DISKDUMP

This is a machine-language disk sector display/modify utility.

DISKDUMP [filespec]

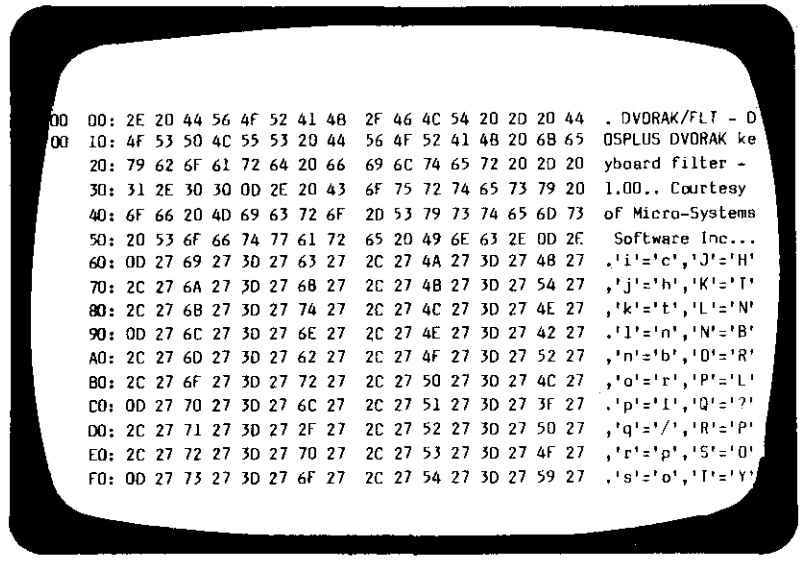
"[filespec]" is the optional name of the file to be examined/modified.

If no filespec is given on the DISKDUMP command line, the program will prompt for a filename by displaying the asterisk prompt.

Enter the file name and include all extensions and passwords, if any. DISKDUMP will now display the first sector of the file. The following commands are available:

Key	Function
;	Advance one sector
-	Go back one sector
+	Advance to end of file
=	Go to beginning of file
F	Find hexadecimal value
G	Go to specified sector
M	Enter modify mode
P	Locate address in load module file

A sector display will look like this :



The two-digit number in the upper left-hand corner of the screen indicates the disk drive device number which the file is resident upon. Note that this is not the disk drive specification, or name; it is the drive device number, and may have a value of 0-7.

Immediately below the drive device number is the number of the physical record currently displayed.

Slightly indented from the left side of the display is a column of two-digit hexadecimal numbers. These numbers indicate the relative byte number of the first byte displayed on each line. For example, in the DISKDUMP display above, the fifth row down from the top begins with the number 40. This means that the first byte on that line is relative byte 40H, the next is relative byte 41H, the eleventh byte is 4AH, etc.

To the right of the row numbers is the actual information contained within the physical record itself. This information is displayed in hexadecimal by default, in eight groups of two bytes each. This hexadecimal display may be exchanged for an ASCII character display by pressing the <F1> key on the TRS-80 keyboard.

Either the next or the previous physical record in the file may be displayed by pressing the semicolon, ";", key (for the next record) or the minus, "-", key (for the previous record. If an attempt is made to display a record outside the limits of the file, the command will be ignored.

The first record or the last record in the file may be displayed at any time by pressing the equal, "=", key (for the first record) or the plus, "+", key (for the last record).

DISKDUMP may display any given record if the "G" command is used. Simply press the "G" key, followed by the desired record number, in hexadecimal. After pressing <ENTER>, DISKDUMP will display the proper record.

DISKDUMP's "F" command is used to find any occurrences of any given hexadecimal value within a physical record. In order to find the occurrences of a byte, type "G" followed by a two-digit hexadecimal value (if in the hexadecimal display mode) or a single ASCII character (if in the ASCII display mode), and press <ENTER>. DISKDUMP will now flash a graphics block in the position occupied by any matching bytes within the record. To abort the "find" display, press any key.

Physical records may be edited by entering DISKDUMP's modify mode. To enter the modify mode, press the "M" key from the record display mode. A reverse video cursor will appear in the upper left-hand corner of the record display. This cursor may be moved about the sector display at will with the four arrow keys on the TRS-80 keyboard. When the cursor is positioned over a byte that is to be modified, simply enter the two-digit hex value (in the hexadecimal display mode) or the single ASCII character (if in the ASCII display mode) which the byte is to be changed to. The cursor will automatically advance to the next byte within the record, moving to the next line of the record display if necessary. When all changes are complete, press the <ENTER> key to write the updated record to diskette. If it is not desired to save the changes to diskette, the <BREAK> key may be depressed to cancel all changes.

When you enter the modify mode via the "M" command, you will have the following options:

Key	Function
up arrow	Decrement cursor position one row
down arrow	Increment cursor position one row
right arrow	Increment cursor position one byte
left arrow	Decrement cursor position one byte
sh up arrow	Home cursor
@	Fill entire sector with "00" bytes, starting at current cursor position
@ xx	Fill "xx" bytes with 00 value, starting at current cursor position
@ xxyy	Fill "yy" bytes with "xx" byte, starting at current cursor position

When editing program files save in load-module format, the "P" command can be extremely useful. By typing "P" followed by a hexadecimal address will cause DISKDUMP to search the file for the byte which will load into the specified address in RAM. When DISKDUMP finds the byte, it will automatically enter the modify mode and position the cursor on the proper byte. If DISKDUMP is unable to locate the address, the sector display will be cleared and the message "Invalid data" will be displayed on the screen. DISKDUMP will then return to the filename prompt.

The "@" command is used to fill a record or a portion of a record with a user-defined byte. The simplest form of this command is:

```
@ xx
```

where "xx" is a two-digit hexadecimal value which defines the number of bytes that are to be filled (starting at the current cursor location) with a 00 byte. A slightly more complex form of this command is:

```
@ xxyy
```

where "yy" is a two-digit hexadecimal value which defines the number of bytes to be filled (starting at the current cursor position) with the byte defined by "xx".

DISKZAP

This utility is a disk sector editor. DISKZAP can be used to display, modify, copy, or verify diskette sectors as well as format diskette tracks.

DISKZAP

There are no parameters for this utility

When executed, the DISKZAP utility will display its command menu on the video screen:

```
  Set
  Fill
  Copy
  Print
  Verify
  Format
* Display
```

This is the MAIN MENU. It lists all the sub-options and allows you to move between them. DISKZAP will default to certain parameters for each drive:

40 cylinders
18 sectors on track 0
Double density track 0
18 sectors on all remaining tracks
All remaining tracks double density

This is unless the drives have been "logged in" via the I command (see I) or by virtue of having been accessed already. If the drive is logged in, you may assume that DISKZAP will use the DCT information and you will not have to access the SET command at all.

If you are using hard disk, 80 track, or double sided drives, you will most likely find it VERY convenient to issue an "I,M" command before entering DISKZAP.

Any of the drive configurations may be altered via the "Set" sub-option. The asterisk that appears to the left of the "Set" option on power-up is the "control cursor". Whichever sub-option it is positioned next to is the one that will be invoked when the <ENTER> key is pressed. It may be moved up and down the list by pressing the <up arrow> and <down arrow> keys. To exit DISKZAP, from the main menu press "O" (as in "Out").

Set (Alter disk drive parameters)

If the default parameters shown above are proper for the diskette you wish to work with (or the drives are "logged in"), then you may proceed directly to the sub-option of your choice and begin the desired operation. If the diskette's characteristics differ from the default parameters, then you need to use the Set sub-option to alter the drive parameters.

To invoke this sub-option, as with any of the sub-options, simply position the control cursor to the left of the word "Set" and press ENTER. The first question to be asked will be:

Drivespec ?

Respond to this with the drivespec of the drive that you are configuring. After setting the drive, you will be asked:

Cylinder count ?

Answer this question with the number of cylinders on the disk that you are configuring. Enter the true cylinder count for the diskette. This parameter is interested in how many tracks are on the DISK, not how many your drive is capable of.

The next prompt is:

Surface count ?

Enter the number of data recording surfaces on the diskette in question. For a single-sided diskette, the proper value should be 1, and for a double-sided diskette, it should be 2. For rigid drives, this parameter will vary according to individual configuration.

Now DISKZAP will query:

CYL 0 sec/trk ?

This is requesting the number of sectors on track zero. The DISKZAP defaults to the proper sector count for 5-1/4" double-density diskettes, 18. If the diskette which is to be operated upon has a cylinder 0 sector count that differs from the default, enter the proper value now.

After the cylinder 0 sector count has been provided, DISKZAP will prompt with:

CYL 0 density ?

This parameter allows you to configure the density of cylinder 0 separately from the cylinder 0 sector count. Reply to this with an "S" for single density or a "D" for double density.

Note that many Model I double density system disks use a single density track zero. This is required by the ROM bootstrap loader. The data diskettes, however, format track zero as double density so that the granules not actually USED by the bootstrap can be freed to the system for data storage.

Following your definition of track 0, you will be given the opportunity to configure those same two parameters for all other tracks on the diskette.

The first query is:

Sectors/track ?

Answer this query with a value (0-255) to indicate how many sectors there are on all the remaining tracks. The standard for 5-1/4" diskettes, of course, will be 10 sectors per track in single density and 18 sectors per track in double density. However, there is the chance that some system could be using more or less sectors on the track without altering the density that the floppy disk controller works in. This parameter gives you the ability to configure for any eventuality

After configuring for the number of sectors, you will be queried as to the density of the remaining tracks. You will be asked:

Track density ?

Respond to this question with either "S" for single density or "D" for double density, depending, of course, on the density of the disk.

When using the set option, you only respond to as many prompts as are pertinent to you. For example, if all you wanted to do was change the diskette's track count, you could go to the set option and alter the track count. Then you could press BREAK and return the command mode immediately. There is no need to step through prompts that are irrelevant.

It is with this in mind that we have designed the set option. The parameters we felt you were going to use the most (cylinder count, surface count, etc.) are the first question which DISKZAP asks, such that they may be altered quickly and allow the user to avoid the rest of the prompts with the BREAK key.

Because pressing ENTER leaves the parameter unchanged instead of re-loading the original default, you do not need to re-enter a parameter that is set the way that you want it. Set will retain this drive configuration for as long as DISKZAP is in operation, but must be re-configured upon each new entry of the program.

Fill (Fill sectors with specified byte)

This option will allow the user to fill a sector with any particular byte that may be desired. This is useful when it is desired to erase completely old data from a sector without re-formatting the entire disk.

To invoke this sub-option, place the control cursor to the left of the word "Fill" in the main menu and press ENTER.

The first question to be asked is:

Drivespec ?

Reply to this with the name of the drive that contains the diskette to be operated on. Any valid drivespec will be allowed here. After answering that question, you will be asked:

Cylinder ?

Answer this question with the track number that contains the first (or only) sector to be filled. This cylinder number is entered in hexadecimal.

Once the cylinder is entered, you will be asked:

Sector ?

Reply to this query with the number of the first (or only) sector to be filled.

The next prompt will be:

Sector count ?

Respond to this with a value that represents the number of sectors, beginning with the sector specified in the preceding questions, to be filled. This cont must be entered in decimal, and may assume any value from 1 to 255. 1 is the default.

The final question will be:

Fill data ?

Answer this question with the byte that you wish to have the sector filled with. This one-byte value must be entered in hexadecimal. Pressing ENTER at this prompt will cause DISKZAP to use the default fill value, which is zero.

For example, if you wanted to fill tracks 4 and 5 of a particular double density diskette with the hexadecimal value "E5", you would answer the questions in the following manner:

```
Drive ? 0
Cylinder ? 4
Sector ? 0
Sector count ? 60
Fill data ? E5
```

After inputting all data and pressing ENTER on the last prompt, the drive will run and DISKZAP will display the track and sector number as it fills each sector.

Copy (Copy sectors)

This function will allow you to copy sectors from one disk to another or from one part of a disk to another. To invoke this command, place the control cursor to the left of the word "Copy" in the main menu and press ENTER. The first question to be asked is:

Drivespec ?

Answer this with the drivespec of the SOURCE drive. Next, you will be asked:

Cylinder ?

Answer this with the cylinder number that contains the first (or only) SOURCE SECTOR. This is the sector that is to be copied (or the first of many, whichever you desire). After answering that question, you will be asked:

Sector ?

This is prompting you for the number of the first (or only) source sector. After this is entered, you will be prompted for:

Drivespec ?

This time it is seeking the drivespec of the DESTINATION DRIVE (the drive to which you wish to copy).

The next prompt is:

Cylinder ?

Answer this with the number of the track on the destination drive that contains the first (or only) DESTINATION SECTOR.

After answering that, you will be queried:

Sector ?

This is prompting you for the number of the first (or only) destination sector. It does NOT necessarily have to be the same as the source sector (i.e. you can copy the last two sectors of track 4 on drive 0 into the first two sectors of track 7 on drive 1).

The last piece of data required will be:

Sector count ?

This prompt is seeking the number of sectors that you wish to copy. Enter the sector count in decimal.

Note that when you are using COPY, you are defining a "block" of sectors. You specify the starting point of this block on both the SOURCE and DESTINATION drives. The "sector count" prompt allows you to define the length of the block. Pressing ENTER will copy only a single sector. But, it must be a CONTIGUOUS block. You are copying sequentially from the source sector to the destination sector for the number of sectors you specify. What this means is, if you wish to copy 50 sectors, skip 200, and copy 50 more, you will have to copy each block of 50 separately. You may, if you wish, locate them beside each other on the destination drive, but they must be copied independently.

For example, if you wished to copy track 2, sector 5 of drive 0 into track 3, sector 12 of drive 1, you would answer the prompts in the following manner:

```
Drivespec ? 0
Cylinder ? 2
Sector ? 5
Drivespec ? 1
Cylinder ? 3
Sector ? 12
Sector count ? 1
```

If you wished to copy an entire Model 4 TRSDOS 6.0, 40 track double-density diskette from drive 0 to drive 1, you would answer the prompts in the following manner:

```
Drivespec ? 0
Cylinder ? 0
Sector ? 0
Drivespec ? 1
Cylinder ? 0
Sector ? 0
Sector count ? 720
```

After answering the "sector count" query and pressing ENTER, DISKZAP will begin the copy. When copying sectors, DISKZAP will seek to read in as many sectors as it can (up to one complete track) before writing them, as opposed to reading and writing a single sector at a time.

When copying a single sector, there will be no operational difference. However, when copying more than a track (especially an entire disk), it makes LARGE difference. DISKZAP will also displays the track and sector number of each sector as it is copied (both the SOURCE sector as it is read and the DESTINATION sector as it is written).

If DISKZAP encounters an error during the sector copy routine, it will pause and display the error discovered. It will also ask if you wish to continue. It would then write as much of the source sector as it could read into the proper destination sector and proceed from there. This will allow you to copy as much data as is absolutely possible from a disk without having to work around known bad sectors. This "proceed after error" feature becomes a key one in repairing blown diskettes. If you can copy a complete track save one sector, then you have only lost 256 bytes of data as opposed to potentially much more.

Print (Print hardcopy of selected sectors)

This command will create printed copy of the contents of specified sectors. To invoke this option, position the control cursor to the left of the word "Print" in the main menu and press ENTER.

The first question asked will be:

```
Drivespec ?
```

Answer this with the drivespec of the drive that contains the first sector to be printed. Next you will be asked:

Cylinder ?

Answer with the number of the cylinder that contains the first (or only) sector to be printed. Following that, you will be queried:

Sector ?

Enter the number of the first (or only) SECTOR TO BE PRINTED. Finally, you will be prompted:

Sector count ?

Reply to this with the number of sectors that you wish to print. Remember, just as with copy, you are dealing with contiguous blocks ONLY! You may not print 5 sectors on track 0 and then 5 on track 11 without printing them both independently of one another.

For instance, in order to print out all the directory sectors (assuming the directory was on track 11 hex) from the double density diskette in drive 0, you would:

```
Drivespec ? 0
Cylinder ? 11
Sector ? 0
Sector count ? 18
```

As each sector is printed, it will be displayed on the screen. You may tell by examining the track and sector indicators in the upper left hand corner of the screen which sector is currently being printed.

Please note that DISKZAP does NOT check for printer ready status. If you engage the print option and there is no printer available, DISKZAP will simply "lock up" and force you to either make a printer available or reset the machine.

Verify (Read and check specified sectors)

This option will allow you to read and verify any specified sectors on the disk. It will check each sector for accuracy by verifying the CRC byte. If it encounters an error, it will pause with the correct error message. Pressing ENTER will cause it to continue verifying.

To invoke this option, as with any other, position the control cursor to the left of the word "Verify" and press ENTER. The first question asked will be:

Drivespec ?

Reply to this question with the drivespec of the drive that contains the diskette that you wish to verify. The next question asked will be:

Cylinder ?

This is prompting you for the cylinder number that contains the sector you wish to begin verifying at. When verifying an entire diskette, you may press ENTER at this prompt to select track 0. After answering that, you will be asked:

Sector ?

This is asking you for the sector number on the above specified track that you wish to begin verifying at. This would allow you to begin verifying with the last two sectors of track 5. Following that, you will be prompted:

Sector count ?

This is seeking the number of sectors, in decimal, you wish to verify. Remember, if you specify more sectors for a disk than you have configured for in "Set", it will wrap around from the last configured track and begin again at track 0, sector 0 and continue from there. That is why it's important to configure for the correct track count before beginning with any diskette.

The final query that DISKZAP will ask is:

Ignore data AM ?

This question requires a yes/no answer. When answered with a "Y", DISKZAP will not display a message to inform the operator that a special type of data address mark, which is reserved for use by the directory, has been detected. If the question is answered with an "N", DISKZAP will print the following message and pause until a key is depressed whenever the special address mark is encountered:

AM/WRITE FAULT

While you are verifying a diskette, you may abort and return to the main menu by holding down the BREAK key.

As an example, suppose it were desired to verify a 35-track, single-density diskette in drive 1. The following data would be provided to the Verify command:

```
Drive ? 1
Track ? 0
Sector ? 0
Sector count ? 350
```

Once you have answered the final question and pressed ENTER, DISKZAP will begin reading the specified sectors. It will display the track and sector number as it verifies each sector. As each sector is read, the CRC value is calculated and checked and any errors reported.

Format (Format a selected track or tracks)

This sub-option allows you to format a track or series of tracks. You may, if you wish, use it to reformat a track somewhere in the middle of a disk to repair a non-readable sector. To invoke this option, position the control cursor to the left of the word "Format" in the main menu and press ENTER.

The first question is:

Drivespec ?

This is prompting you for the drivespec of the drive that contains the disk you wish to format a track on. After answering that, you will be queried:

Cylinder ?

Respond to this with the number of the cylinder at which you wish to begin formatting. The next question is:

Cylinder count ?

This is seeking the information as to how many cylinders you desire to format. Pressing ENTER at this prompt will default to one track.

The final question under the Format command is:

Interleave factor ?

The interleave factor determines the order in which sectors are numbered on the diskette, and can have a profound effect on disk access speed. The normal values for sector interleave factor are 2 for 5-1/4" single-density diskettes, and 3 for 5-1/4" double-density diskettes.

Please note that the DISKZAP Format command is not interchangeable with the TRSDOS utility FORMAT. The DISKZAP Format command is simply capable of performing cylinder formatting; the FORMAT utility not only formats a diskette but also initializes the diskette with a great deal of system information, including a bootstrap and disk directory.

Display (Display or modify diskette sectors)

This is perhaps the most often used option in DISKZAP, and the heart of the disk editor. DISKZAP uses a full screen editor that has cursor wraparound.

To invoke this sub-option, position the control cursor to the left of the word "Display" in the main menu and press ENTER.

The first question you will be asked is:

Drivespec ?

Answer this query with the drivespec of the drive that contains the diskette with the sector you wish to display/modify. The next question is:

Cylinder ?

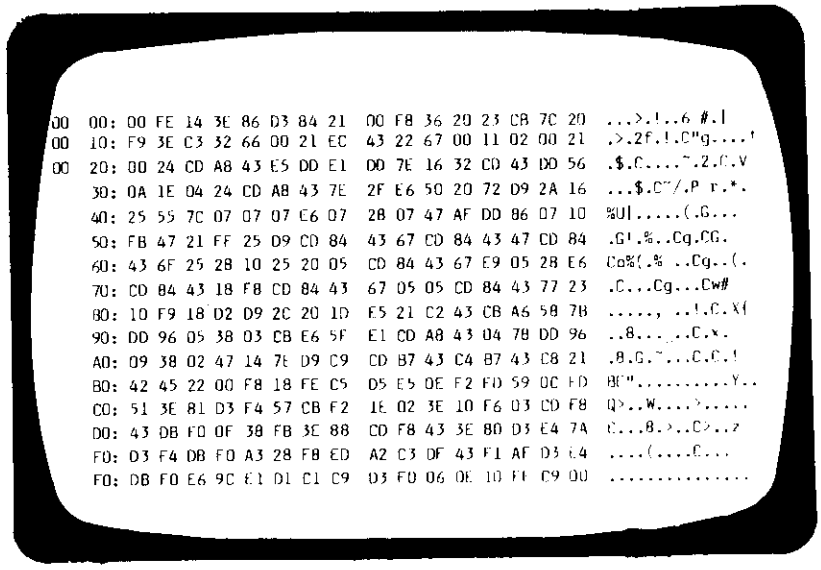
This is prompting you for the number of the cylinder on the disk that contains the sector you wish to examine.

The final question is:

Sector ?

Reply to this with the number of the sector you wish to display.

After typing in the sector number and pressing ENTER, you should see a display that looks something like this:



At the upper left-hand corner is a one-byte hexadecimal value which relates the logical device number of the disk drive currently addressed. Note that this is a device number and not a drive specification. The number listed immediately below this is the current cylinder number, and below that is the current sector number. The column of digits slightly indented from the left margin are the BEGINNING BYTE INDICATORS. Each one of those indicates the number of the first byte in that row. Then there are rows of 16 bytes each (10 hex). This is the HEXADECIMAL DISPLAY AREA. These are set in groupings of two bytes, such that you have eight columns of two separated by spaces. Immediately to the right of the hexadecimal portion of the sector display is the ASCII DISPLAY AREA. There are 16 ASCII characters on a row corresponding to the bytes in the hexadecimal display row immediately to its left. Non-ASCII characters will be displayed as periods.

At this point, you have several options, each of which is controlled by a single keystroke. They are:

Key	Function
;	Increment display position one sector
+	Increment display position one cylinder
-	Decrement display position one sector
=	Decrement display position one cylinder
BREAK	Return to main menu
M	Enter modify mode

If you select "M" to enter the modify mode, the display will change slightly and you will have several other options.

If DISKZAP encounters an error during a sector read in the display mode, it will pause and display the error discovered. It will also ask you if you wish to continue. If you respond "Y", it will display as much of the sector as it could read. You may then enter the modify mode and make any corrections possible before re-writing it. The sector will be re-written to the disk reflecting any corrections you may have made. That means there will no longer be a read error from system level. It does not mean that the data is now 100% correct. It is correct only to the level that were able to repair it, but it will read as it is now without an error. This "continue after error" feature will allow you to rescue bad sectors in part or in whole, where otherwise you would have had no chance of recovering the data.

When you enter the modify mode, a reverse video cursor will appear over the byte in the upper left hand corner. You move the cursor about within the sector by using the arrow keys. Whatever byte is the graphic block cursor is currently positioned over is referred to as the CURRENT CURSOR LOCATION. This is the byte that will be affected should you enter a change.

Upon entering the modify mode, two additional pieces of information will be displayed in the upper left hand area of the screen. Immediately underneath the current sector number will appear the current cursor location. This will change as you move the cursor about in the sector. Underneath that will be the value of the byte at the current cursor location.

At this point, you have several options, each of which is controlled by a single keystroke. They are:

Key	Function
right arrow	Increment cursor position one byte
down arrow	Increment cursor position one row
left arrow	Decrement cursor position one byte
up arrow	Decrement cursor position one row
BREAK	Aborts modify mode and returns you to the main menu without re-writing the sector. Restores original contents.
ENTER	Terminate modification mode and returns you to the display mode after writing the modified sector to the disk.
F1	Toggles sector display between hexadecimal and ASCII character display mode
@	Fills entire sector, starting at current cursor position, with "00" bytes.
@ xx	Fills "xx" bytes, starting from the current cursor position, with "00" bytes.
@ xxyy	Fills "yy" bytes within the current sector from current cursor position with data byte "yy"

To modify a byte, position the cursor over the proper byte and enter the two-digit hexadecimal value (if in the hex display mode) or single-character value (if in the ASCII character display mode) which the byte is to be changed to. When you finish modifying one byte, the cursor will move onto the next. If that was that last byte of a row, the cursor will move onto the first byte of the NEXT row. The only exception is the last byte of the last row. After modifying it, the cursor will stay right where it is. To begin with the next sector, write this one back to the disk with ENTER, advance to the next sector with ";", enter the modify mode again with "M", and return to modifying.

Note: As a general rule, DISKZAP expects all cylinder and sector addresses as well as fill data to be entered in hexadecimal format. Cylinder and sector counts, on the other hand, are assumed to be entered in decimal.

FORMAT

This utility allows you to organize a diskette and prepare it to receive data.

FORMAT :dr (param=exp...)

":dr" specifies the drive containing the disk to be formatted. If this is not given at the command line, FORMAT will prompt for it.

"param" is the optional action parameter that modifies the effect of the command.

Your parameters are:

DATE="string"	Format date.
PW="string"	Disk Master Password.
NAME="string"	Disk name.
CYLS=value	Number of cylinders.
SIDES=value	Number of sides.
DEN="string"	Track density.
USE="string"	Control the prompt "Diskette contains data, Use or not?".
VERIFY=switch	Control whether the format is verified or not.
INTER=value	Set sector interleave.
FPAT=value	Specify format data pattern.

Abbreviations:

DATE	D
PW	P
NAME	N
CYLS	C
SIDES	S
DEN	DE
USE	U
VERIFY	V
INTER	I
FPAT	F

The FORMAT utility is used to organize the diskette into tracks and sectors and prepare it to receive data. You will use this both in formatting new disks and in "starting over" with a clean slate on old ones. All disks must be formatted before they can be used by the system. If is NOT, however, necessary to format a disk before backing up to it (see the utility program BACKUP). BACKUP will format the destination disk if it is blank.

The disk to be formatted may be either blank or contain data. If you format a disk that already contains data, any data on that disk will be permanently lost. When you format a disk, DOSPLUS will check the disk for flawed granules. If it discovers any areas of the disk during format that are bad, it will "lock out" those areas and prevent the system from attempting to use them (unless you specify VERIFY=N).

To format a disk, type "FORMAT" from the DOS command mode and press ENTER. The first message to appear will be:

Target drivespec ?

Enter the drivespec of the drive that contains the disk you wish to format. You will then be asked:

Diskette name ?

Enter the name you wish to assign to that disk. Any characters are legal (numeric or alphabetic). You have a maximum of eight characters. Following that, you will see:

Format date ?

Enter today's date. You may, if you wish, use this field for something else. It will be displayed whenever you execute a CAT, DIR, or FREE upon that disk. DOSPLUS doesn't use the disk date for anything, so this area is free for you to use. Eight characters maximum. May be alphabetic or numeric. After entering this, FORMAT will prompt you:

Master password ?

Enter the desired Disk Master Password. This password will be used for a variety of functions later. Pressing ENTER will default to "null password", but from then on you will not be able to assign effective file protection. The Disk Master Password will always override the file password. If a Disk Master Password is NOT set, then specifying no password will ALWAYS get you into a file. We therefore recommend that the Disk Master Password always be set. Maximum of eight characters. Once you have answered that prompt, you will see:

Number of cylinders (35-96) ?

Enter the number of cylinders you to which you wish to format the disk. Enter the number of cylinders desired or press ENTER to default to 35 on the Model I, or 40 on the Model III. After that, you will be queried:

Number of sides ?

Enter "1" for single sided drives or "2" for double sided drives. Remember that single or double sided is limited by your drive hardware. Simply answering this prompt "2" on a machine with single-sided drives is NOT going to give you a double sided disk. After answering this, you will be asked:

Single or double density ?

Enter "S" for single density or "D" for double. Pressing ENTER defaults to "D". DOSPLUS formats 10 sectors per track in single density and 18 in double.

After you have answered all these questions, DOSPLUS will proceed with the format. If the diskette was not blank, you will be warned:

Diskette contains data, use or not ?

Enter "Y" to proceed or "N" to abort. Pressing BREAK will also abort.

If the disk was blank, or you have signalled FORMAT to use it anyway, you will see the track number displayed as first they are formatted and then verified. When the procedure is complete, you will see:

Insert SYSTEM disk [ENTER]

flashing on the screen. Make certain that a system disk is inserted and then press ENTER to return to DOSPLUS.

Specifying format options from the command line

Optionally, you may specify some or all of the FORMAT parameters right from the command line. In this respect, FORMAT has parameters. Any of the parameters not specified on the command line (with the exception of VERIFY, INTER, and FPAT) will be prompted for. One at a time, these parameters are:

DATE. This parameter allows you to specify the diskette's format date. This will be displayed with any command that displays the diskette's "pack I.D." (e.g. the disk name and creation date). These commands include CAT, DIR, FREE, MAP, and others.

The date must be given as a string variable between one and eight characters in length. It must be in quotes. You are not restricted to numeric data. For example:

```
FORMAT :1,D="07/06/83"
```

PW. This parameter allows you to specify a one to eight character diskette master password. This is very important because without a disk master password, NO files on that diskette are protected. Omitting this parameter will not give you a null password, it will just get you a prompt. To specify a null password, use PW="". You may have any legal password characters in this string. For example:

```
FORMAT :1,D="07/06/83",P="ALP"
```

CYLS. This parameter allows you to specify the number of cylinders on the disk being formatted. This should be a numeric value between 35 and 96. Do not specify a string. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40
```

SIDES. This parameter allows you to specify the number of surfaces on a diskette. Set this equal to a value of either 1 or 2 (depending on whether or not the drive is double sided). Note that standard Radio Shack disk drives as of this writing were NOT double sided. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1
```

DEN. This parameter allows you to specify the diskette density. Specify this as a string. Use "S" for single density and "D" for double. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1,DE="D"
```

USE. This parameter allows you to control FORMAT's action when a diskette contains data. If you specify "N", and the disk contains data, FORMAT will abort. If you specify "Y", "U", or "F", and the disk contains data, FORMAT will proceed. It offers you the opportunity to skip the question "Diskette contains data, use or not?". Specify this as a string value. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1,DE="D",U="Y"
```

VERIFY. This parameter allows you to format a disk and write system information to it without verifying the data written. This is only for use in special cases such as when you have been repeatedly re-formatting a diskette and are CERTAIN that it has no flaws. Do NOT use this as an every day occurrence. If you have to shut verification off in order to format a disk, there is something wrong with your system and it should be repaired.

You must specify VERIFY from the command line to use it. FORMAT will not prompt for it later. Since VERIFY defaults to on, the only thing you will ever enter is V=N to shut it off. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1,DE="D",U="Y",V=N
```

INTER. This parameter allows you to control the disk "interleave". Diskette interleaving is a term that refers to the order in which the sectors are numbered on the track. A diskette with an interleave of 2, for example, would require the DOS to read the entire track in 2 revolutions of the disk. This is extremely fast. A diskette with an interleave of 3 would require the DOS to read it in 3 revolutions of the disk.

DOSPLUS on the Model 4 runs with a diskette interleave factor of 2. For double density disk I/O, this is very rapid and accounts for much of the system's smooth and fast operation. It also causes a problem. Model III DOSPLUS and TRSDOS 6.0 can read disks with an interleave of 2, but very slowly. The sectors are coming around SO fast that it actually takes MORE revolutions to read the track.

Also, certain applications software may not be able to keep up with that fast of an interleave for data file I/O and may actually run FASTER with a SLOWER interleave. This has not been determined at this writing (since there IS no Model 4 applications software as of this writing), but as future information is available, we will make it known.

Suffice it to say that for right now, when formatting diskettes for DOSPLUS on the Model 4, do not concern yourself with this parameter and allow FORMAT to use the default of 2. When formatting disks for use with Model I or III DOSPLUS (or TRSDOS 6.0), use INTER=3. For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1,DE="D",U="Y",V=N,I=3
```

FPAT. This parameter allows you to specify the desired data pattern to be used with the FORMAT program. Normally, the DOSPLUS formatter uses a data pattern of "6C6C", which is a good general purpose pattern. However, for worst case data patterns, easier data patterns, etc., you may want to adjust it. Some suggested values:

6DB6	This is the worst case data pattern for double density formatting. In situations when you wish to be the most certain of a diskette, use this pattern.
E5E5	This is the worst case pattern for single density formatting. In situations when you wish to be the most certain of a single density diskette, use this pattern.
0000	This is a very easy format pattern. If you cannot format a diskette with one of the more difficult patterns (actually, even the worst case pattern), then either your hardware or media is not up to optimum performance specifications. We cannot recommend writing valued data to a disk that requires you to downgrade the effectiveness of the format pattern in order to get it formatted.

For the most part, the 6C6C pattern is very good in both single and double density. It may be the toughest "all around" pattern. But you may specify another if you wish. You must specify this from the command line if you wish to use it.

Remember, you are specifying a value, so you must include the "H" for hexadecimal input. Also, in order to function correctly, you must specify the second byte of a two byte value FIRST. For example, 6DB6 becomes "B66DH". For example:

```
FORMAT :1,D="07/06/83",P="ALP",C=40,S=1,DE="D",V=N,I=3,F=B66DH
```

HELP

The HELP utility on DOSPLUS is intended to provide a means of quick reference to the proper syntax and valid parameters for DOSPLUS library commands and some of the often-used utility programs.

==========

HELP
HELP [FROM] command [TO] filespec/@devicespec

"command" is the name of a DOSPLUS library command or utility.

==========

The HELP program will provide information on the following commands:

APPEND	ASSIGN	ATTRIB	AUTO	BOOT	BREAK
BUILD	CAT	CLEAR	CLOCK	CLS	CONFIG
COPY	CREATE	DATE	DEBUG	DIR	DO
DUMP	ERROR	FILTER	FORCE	FORMS	FREE
I	JOIN	KILL	LIB	LINK	LIST
LOAD	PAUSE	PROT	REMOVE	RENAME	RESET
ROUTE	RS232	SCREEN	SYSTEM	TIME	VERIFY

and the following utilities:

BACKUP	CONVERT	DIRCHECK	DISKDUMP	FORMAT	MAP
PATCH	RESTORE	SYSGEN			

To display the HELP available for any command or program, type HELP followed by the command or program name, separated from the HELP by a space. For example, typing:

HELP COPY

will result in:

COPY [FROM] fs/@ds [TO] fs/@ds (param=exp,...)
COPY [FROM] fs [TO] :dr (param=exp,...)
COPY [FROM] :dr [TO] :dr [USING] wildmask (param=exp,...)

DPW='string'	ECHO=switch	INVIS=switch	KILL=switch
MOD=switch	OVER=switch	PROMPT=switch	QUERY=switch
SPW='string'	TINY=switch	NEW=switch	OLD=switch

If HELP is requested for a any command or program not listed in the above list, HELP will display a list of valid commands and programs.

Note that HELP may send output to any device or file. For instance, typing:

HELP ATTRIB TO @PR

will output the HELP information on the ATTRIB command to the printer.

MAP

The MAP utility provides a list, by cylinder and sector, of the areas allocated to files on a diskette.

==========

MAP [FROM] :dr [TO] file/@device [USING] wildmask (param=exp...)

":dr" is a disk drive specification.

"file/@device" is a file or character-oriented device to which the output from MAP will to sent.

"wildmask" is a wildmask controlling which files will be displayed.

The allowable parameters are:

SYSTEM	Include system files in MAP listing
INVIS	Include invisible files in MAP listing
HEX	Provide cylinder & sector numbers in hexadecimal

Abbreviations:

SYSTEM	S
INVIS	I
HEX	H

==========

The MAP command may provide a file-by-file list of diskette space allocation. To display all files on a diskette, and the areas occupied by them, type:

MAP :dr

where ":dr" is the drive number which you wish to MAP. The screen will display something like the following:

```
[TRSDOS  01/26/83]
MAILLIST    001,006 - 001,011
CLICK/FLT   001,012 - 001,017
BASIC/CMD   003,000 - 003,005
```

Each filename is followed by a set of numbers. Take, for example, the case of MAILLIST, in the listing above. The MAP tells us that MAILLLIST begins on cylinder 1, sector 6 and continues through cylinder 1, sector 11. Large files may contain more than one segment of this sort. For instance, in the MAP shown below, the file FDAT is divided into five separate segments:

```
[Utility  01/31/83]
FDAT        012,000 - 017,005
            018,012 - 019,017
            021,000 - 025,017
            027,012 - 029,017
            034,000 - 036,011
```

Note that the MAP utility can provide information in hexadecimal notation as well as decimal, if the HEX parameter is specified. The fist example, above, would appear like this in hexadecimal format:

```
[TRSDOS  01/26/83]
MAILLIST    01,06 - 01,0B
CLICK/FLT   01,0C - 01,11
BASIC/CMD   03,00 - 03,05
```

The SYSTEM and INVIS parameters may be used to cause the MAP utility to display a MAP of system files and invisible files, respectively. Without these two parameters, only the visible user files will be displayed. You may abbreviate these parameters as "S" and "I".

Output from MAP is normally sent to the video display, but it may be re-directed to any other device, or to a file. For instance, to obtain a printout of a MAP of all files on drive 2, type:

```
MAP :2 @PR (SYSTEM,INVIS)
MAP :2 @PR,S,I
```

to send it to a file:

```
MAP :2 TO MAP/TXT:1 (S,I)
MAP :2 TO MAP/TXT:1,S,I
```

Note that the TO preceding the output file/device is optional.

By using the wildmask, you may restrict MAP to a certain file or group of files. For example:

```
MAP */CMD:0
```

will map all of the /CMD files on drive ":0".

```
MAP TEST/TXT!:0
```

will map the file TEST/TXT from drive ":0". Note the exclamation mark (!) to inform MAP of the wildmask. This is required if you are not going to include the USING delimiter and there are no other wildcard characters to distinguish it.

If you do not specify any options on the command line, MAP will load and prompt you with an asterisk. You may enter the proper options then.

PATCH

The PATCH/CMD file provided on DOSPLUS is a utility program which is used to apply patches, or modifications to files saved in load module format. The PATCH utility may use patch files, which contain information instructing the PATCH utility what modification to install, or PATCH will accept patch information from the keyboard in an interactive mode.

==========

PATCH filespec1 filespec2
PATCH filespec1
PATCH filespec1 patname (KILL)

"filespec1" is the name of a load-module format file.

"filespec2" is the name of a patch file.

"patname" is the name of a patch to be removed.

Your parameters are:

KILL=switch Used to inform PATCH to remove a patch from a file.

Abbreviations:

KILL K

==========

The first form is used when a patch file is to be provided to the PATCH utility. In this form, "filespec1" is the name of the load-module format file which is to be patched, and "filespec2" is the name of the file containing the patch information.

Patch files consist of one or more patch information lines. A patch information line has the following format:

A=xxxxH,F=xxxxxxxx,C=xxxxxxxx

The three parameters, A, F, and C, may be specified in any order. The A parameter is used to specify the address within the load-module file at which the patch is to be installed. This value may be given in binary, octal, decimal, or hexadecimal, by appending a B, O, D, or H, respectively, to the value. If no letter is appended to the value, decimal is assumed. If the address given with the A parameter cannot be located within the load-module file, the PATCH utility will display the message "Address not found", and will return to the DOS command level.

The F parameter is used to specify an optional string of hexadecimal or character values which PATCH will attempt to find at the address specified by the A parameter before applying any patch to the file. If matching bytes are found within the load module file at the proper address, PATCH will procede. If the bytes are not found in the proper location, PATCH will return the message "String not found", and will abort to the DOS command level. As mentioned above, the "find" string is optional within the patch line. The F parameter itself is not. This means that even if it is not desired to check for a certain pattern of bytes before applying a patch, the F parameter must still be present in the patch line. For instance, if we desired to apply a patch to a program at address 5481H, changing the bytes CD 98 55 to 00 00 00, we might supply this patch line:

```
A=5481H,F=CD9855,C=000000
```

However, if we were not interested in the current contents of the three bytes starting at 5481H, we would use this patch line:

```
A=5481H,F=,C=000000
```

If a hexadecimal string is specified, spaces between individual bytes are optional. Therefore, either of the two follwing patch lines are valid:

```
A=7438H,F=11 32 71,C=11 35 71
A=7438H,F=113271,C=113571
```

As mentioned above, the F parameter may accept a string of character values as well as hexadecimal values. The character string must be enclosed in either single or double quotation marks, as below:

```
A=65AFH,F="Exit to TRSDOS",C="Exit to System"
A=539CH,F='Drive # (0-3):',C='Drive # (0-7):'
```

The C parameter is used to specify a hexadecimal or character string which is to be placed at the address specified by the A parameter. As with the F parameter, the hexadecimal string may contain optional spaces between byte values, and character strings must be contained within either single or double quotation marks.

Comment lines, as well as patch lines, may be provided to the PATCH program. Comment lines are ignored by the PATCH utility, but are useful to document the intended purpose of the patch. A comment line is any line beginning with a period, ".", symbol. A typical patch file containing comment lines is shown below:

```
.This patch is for the EZPUTER/CMD program
.produced by Jones Computing.
A=67ADH,F=C30000,C=760000
A=7E3D,F=4940,C=1144
.end of patch
```

Patch files may be constructed with the BUILD command under DOSPLUS, or with a word processor.

To reiterate, the second form of command used to enter the PATCH utility is:

PATCH filespec

After the PATCH program loads into RAM, the program will prompt with the asterisk, "*", symbol. Patch lines identical in format to those used in a patch file should now be entered from the keyboard. After all patches have been keyed in, press the <BREAK> key to install the patches.

When PATCH installs modifications to a program, it assigns a name to the set of modifications. In the case of patches applied with a patch file, the patch name assigned is the filename (excluding any extension, password, or drivespec). For instance, if the following command is executed:

PATCH TERMINAL/CMD TERM3/PAT

the name assigned to the patch will be "TERM3". If a file is patched using patch lines entered from the keyboard, the name "*NONAME*" will be assigned to the patch.

Patch names are important for another function of the PATCH utility. This function allows a patch to be removed from a load-module file. In order to remove a patch, use the following command:

PATCH filespec patname (KILL)

where "filespec" is the name of the load module file from which to remove the patch, and "patname" is the name of the patch to be removed. PATCH will search the file for the proper patch, and remove it if it is present. If no patch by that name has been applied to the file, PATCH will display the error message "Patch not found".

RESTORE

The RESTORE utility is used to reclaim files which have been KILLed or REMOVEed.

==========

RESTORE [filespec]

"filespec" is the name of a KILLed or REMOVEed file.

==========

Note that RESTORE cannot reclaim any KILLed or REMOVEed file. Certain conditions must be met:

(1) The disk space originally allocated to the file must not have been reassigned to another file.

(2) The primary and any extended directory entries for the file must not have been altered in any way.

If either condition is not met, RESTORE will issue the message "Disk space has been re-allocated", and will abort.

If you do not specify the filespec from the command line, RESTORE will prompt you with an asterisk and you may enter the filespec at that time.

When attempting to recover a file, RESTORE will search all available drives (unless a drive specification is explicitly provided in the command line) for the file. If RESTORE locates an active file bearing the same name, it will abort with an error. If this occurs, and the second file is NOT the same as the file you want to recover (e.g. TEST that has been KILLed or REMOVEed on drive 1 and TEST that is active on drive 2), simply specify TEST:1 in the RESTORE command line. If RESTORE is able to recover the file, the utility will exit to DOS, and the RESTOREd file will be immediately useable.

NOTE: When attempting to recover a file, RESTORE will reclaim the first occurrance of the proper filename in a diskette directory. If the same filename has been created and KILLed or REMOVEed several times upon a diskette, RESTORE may not recover the same occurrance of the file as was intended. If this is the case, the improper file should be RENAMEd and KILLed or REMOVEed. The RESTORE process may then be repeated until the proper file is recovered.

SYSGEN

This utility is used to place the DOSPLUS operating system upon any DOSPLUS compatible media, such as rigid drives or double-sided floppy diskettes.

==========

SYSGEN :dr filespec (param=exp)

":dr" is the drivespec of the disk drive containing the media to be SYSGENed

"filespec" is the name of an optional bootstrap program to be placed on the diskette

Your parameters are:

XFER=value	Used to specify an address to which control will be transferred after initial system reset. Used in conjunction with an alternate system driver.
OVL=value	Used to specify the highest system overlay to be included. Automatically excludes SYS0.

Abbreviations:

XFER	X
OVL	O

==========

SYSGEN allows the user to place a DOSPLUS operating system on any type of DOSPLUS-compatible media, including 8" diskettes, double-sided diskettes, and rigid drives. In order to SYSGEN any drive, the drive must be properly configured (see the CONFIG library command) and the media must be formatted (see the FORMAT utility).

The simplest form of SYSGEN is:

```
SYSGEN :dr
```

where ":dr" is the drivespec of the disk drive containing the media to receive the DOSPLUS system files. If this command is executed, SYSGEN will place all of the DOSPLUS system files on the formatted diskette, and once complete, the diskette will be able to serve as a DOSPLUS system disk.

The optional filespec in the SYSGEN command line allows the user to place a special bootstrap program beginning on cylinder 0, sector 0 of the SYSGENed diskette. For example, if it were desired to SYSGEN a diskette in drive 2 and to place the special bootstrap program 8INCH/CIM on the diskette, the following command would be used:

```
SYSGEN :2 8INCH/CIM
```

XFER. This optional parameter is used in conjunction with the special bootstrap program. If the bootstrap contains an alternate system driver program, the XFER paremeter is used to provide DOSPLUS with an address which it will transfer control to immediately after system reset and initialization. To SYSGEN a diskette on drive 4 with the bootstrap file NEWBOOT and an XFER address of 5200H, the following command would be used:

```
SYSGEN :4 NEWBOOT (XFER=5200H)
```

Specific instructions on bootstrap programs and the proper XFER addresses will be provided with the appropriate drivers and bootstrap files.

OVL. This parameter is used to restrict which overlays that SYSGEN copies to the new diskette. When specified, SYSGEN automatically excludes SYS0. This is essentially used in two applications.

First, when SYSGENing a MEMDISK, you may specify OVL=16 and install the entire operating system except SYS0 on the MEMDISK. Since you cannot boot to a MEMDISK, the floppy was booted at some time. SYS0 was already loaded and will never be unloaded. Therefore, we can safely SYSGEN a MEMDISK with OVL=16 and transfer control to it with the ASSIGN command. For example:

```
SYSGEN :8,OVL=16
```

where drive ":8" is the MEMDISK.

Second, when SYSGENing a MINIMUM system disk (e.g. not even any library commands, just loading and executing programs), you may specify OVL=5. This copies the needed operating system overlays for program executing ONLY. You will have no library commands at all.

Of course, you may alter this to include only a few library overlays or whatever you wish. You simply may not specify exactly which overlays to use, only which one to stop at. For example:

```
SYSGEN :5,OVL=5
```

where drive ":5" has the disk to be SYSGENed. This disk will lack library commands.

Using SYSGEN is an excellent way to create "working" DOSPLUS system disks. SYSGEN the disk and then use COPY to move over whatever utilities that you need.

TRAP

The TRAP utility is a machine-language program which intercepts disk I/O errors and allows the operator to determine what action to take in the event of an error.

==========

TRAP

There are no parameters for this utility.

==========

The TRAP utility is invoked by simply typing "TRAP" at the DOSPLUS command level. It will install itself into high memory, much like a device driver. Once resident, TRAP intercepts many disk I/O errors before the error condition is returned to the program requesting disk access. If an error occurs, TRAP will allow the operating system to report the error as per usual, but then it will display the following prompt for the operator:

Abort, Continue, Retry, Ignore?

The operator may abort the current operation and return to the DOSPLUS command level by typing "A" and pressing ENTER.

By typing "C", for continue, TRAP will return error status to the program requesting disk access. The program will then handle the error in its normal manner.

TRAP may be instructed to repeat the error-causing disk I/O function by entering an "R", for retry. If the I/O operation is sucessfully carried out upon retrying, the TRAP will not intervene and the program will proceed as if no error occured. If the error re-occurs, TRAP will once again intercept the error.

By entering "I", for ignore, TRAP will return to the disk I/O-requesting program without informing the program that an error has occured. In some circumstances, this may be desirable to gain access to portions of a diskette that would be otherwise unreadable.

Note that once the TRAP program is installed, it remains active until a system reset or until a new /CFG file is loaded which does not contain the TRAP program.

MEDIC

This utility will allow you to perform many commonly used file operations from a menu-driven environment.

==========

MEDIC :ds [wildmask] (param=exp...)

:ds is the drive specification of the drive containing the disk to be operated upon.

[wildmask] is the optional wildmask that will allow you to restrict MEDIC to operating upon a particular set of files.

(param=exp...) is the optional parameter controlling which files will be displayed.

Your parameters are:

SYSTEM=switch	Controls whether or not system files (those files carrying a system attribute) will be included in the MEDIC operation.
INVIS=switch	Controls whether or not invisible files are included in the MEDIC operation.

Abbreviations:

SYSTEM	S
INVIS	I

==========

MEDIC, which stands for Menu Environment DOS Interface Controller, will always display the target files in alphabetical order, regardless whether the entire directory was asked for or a wildcard mask was used. The files will be displayed in catalog (CAT) format and the MEDIC cursor will be positioned over the first file. To the right of the copyright information, the display will show the drive number being accessed and the options that are being used, i.e. I,S.

The "cursor" referred to for the MEDIC program consists of a reverse video display of a target file. The file name will blink between reverse video and normal video. When a file is marked, it will remain solid in reverse video.

NOTE: There is no option to enter a password while using the MEDIC program. If a file is password protected, MEDIC will not grant you access. If MEDIC encounters a protected file in the middle of a multi file command, the balance of the command will be aborted. To insure complete access under the MEDIC program, the disk master password should be set to "", either when formatting or with the PROT command.

The following keys are considered control keys for the MEDIC program.

Key	Function
BREAK	BREAK is the normal exit from the MEDIC program. When BREAK is first pressed, you will queried "Break ?". To exit the program, press the BREAK key again. The BREAK key may be used to cancel any command that has not been executed. Keep in mind that once a command is being executed you will not be able to break, such as during a Copy.
ENTER or @	MEDIC will attempt to execute the file marked by the cursor. If the file contains a CMD or CFG extension, the file will be treated as a command file. Any other extension, or the lack of an extension will cause MEDIC to enter BASIC and attempt to execute the file as a BASIC program. Using the ENTER key to execute a program will have the system return to the MEDIC program after the execution of the selected program. This will take place when there is a normal exit to DOS command level. When the @ key is used to execute a program, the system will not return to MEDIC after the program is executed.
Space Bar	Will mark a file. If the file is already marked, it will be unmarked. Once a file is marked, it will be diplayed in reverse video format and will stay marked unless the mark is removed.
Left Arrow	Will move the cursor one file to the left. If the cursor is on the left most file for a row, the cursor will move to the far right file and move up one row. If the cursor is at the left of the top row, there will be no effect.
Right Arrow	Will move the cursor one file to the right. If the cursor is on the right most file for a row, the cursor will move to the far left and move down one row. If the cursor is a the right of the last row, there will be no effect.
Down Arrow	Will move the cursor one file down in the same column. If the cursor reaches the bottom of the display and there are more files to be displayed, the screen will scroll up one line. When the cursor reaches the last line of the directory entries, there will be no effect.

Up Arrow	Will move the cursor one file up in the column. If the cursor reaches the top of the display and there are previous entries in the directory, the screen will scroll down one line. When the cursor reaches the first line of directory entries, there will be no effect.
Shift Down Arrow	Will move the cursor to the last file (alphabetically) of the files requested. If the directory contains more than one screen of data, the new display will locate the cursor in the bottom row of the display and display the appropriate previous files.
Shift Up Arrow	Will move the cursor over the first file of those requested and it will be located in the upper left of the display. If there is more than one screen full of data, the display will be returned to the first file of the files requested and not the first file on the particular display.
F1	Will "FIND" the first file that begins with the letter entered immediately after pressing the F1 key. The file found will be displayed in the top row of the display.
F2	Will "MARK" all of the files requested. If the "I" and "S" parameters were used when calling MEDIC, all of the files on the target drive will be marked.
F3	Will remove the marks from all marked files.

==========

There are two types of commands available through the MEDIC program, single file commands and multi file commands. The single file commands are EXECUTE, LIST, and RENAME. The commands COPY, KILL, WRITE, and ZERO may be used in either a single file or multi file environment.

The abbreviations used to designate the desired command are the first letters of the command. The available commands will be displayed at the bottom of the screen in reverse video format on the command letter. The commands are entered by pressing the desired single letter abbreviation.

Abbreviation	Command
C	COPY
D	DIR
E	EXECUTE
K	KILL
L	LIST
R	RENAME
W	WRITE
Z	ZERO

==========

COPY

The Copy command will query you at the bottom left of the screen with "Copy to?". You may respond with the destination drivespec as ":ds" or if it is a single file copy you may enter a complete filespec. This command will accept a valid device; i.e. @pr as a destination. If any files are marked do not enter a new name, just the drivespec. Marked files include the file that is at the current cursor position. If this file is not to be copied, move the cursor to a marked file. As the files are copied, the filespecs will be displayed at the bottom of the screen. The files will be copied in alphabetical order.

DIR

The Dir command will display the directory of a drive in catalog format. The options for this command are "I" for invisible, "S" for system and wildcard masks. A prompt "Enter command line :" will appear in the lower left of the screen. You may enter a drivespec with or without the "I" or "S" parameters as desired or a wildcard mask. Failure to include a drivespec will cause MEDIC to use the current drive being accessed. If there is no current drive then it defaults to the system drive.

EXECUTE

The Execute command is strictly for executing a DO file. The "E" command will execute the file marked by the cursor as a DO file. If there is no file extension, the "E" parameter will assume a TXT extension which would generate a file not found error unless the file exists on a different drive.

KILL

The Kill command will kill all files that are marked. This will include the file that is currently being marked by the cursor. When the "K" key is pressed, you will be asked "OK to Kill?". You may respond with a "Y" to kill the marked files or a "N" to return to MEDIC. Once you respond with a "Y", you may not abort the command. The filespecs being killed will be displayed at the bottom of the screen.

LIST

The List command will list the file that is at the current cursor location. Any files that were marked will not be listed. The file will be listed to the display and there are no options to list the file to any other device.

RENAME

The Rename command will allow you to rename a file. The prompt "Rename to" will appear and you should enter the new name. If there are any files marked, only the file at the current cursor position will be renamed.

WRITE

The Write command will cause all of the files that are marked to be written to a disk file. The prompt "to file :" will appear and you should enter the name of the file that is to be used. The marked files will include the file that is at the current cursor location. The file will contain the filespecs with a carriage return after each entry. This command may be used to create a file to be used with the OFFLOAD/JCL file for backup up hard drives.

ZERO

The Zero command will fill the designated file(s) with zeros. This command will accept all marked files, including the file at the current cursor location. The prompt "OK to zero" will appear. You should answer witha "Y" to zero all of the marked files or a "N" to return to MEDIC. The BREAK will also abort the command at this time. Keep in mind that once the "Y" is entered, the command may not be aborted. Be very careful marking files as this command can destroy many files in a matter of seconds.

The following is the DOSPLUS IV Job Control Language table of contents:

Section		Page
I.	Job Control Language	4-1
	Invoking JCL	4-1
	The Keyboard Queue	4-2
II.	JCL Program Structure	4-3
	JCL Commands	4-4
	JCL Variables	4-4
	Special variables	4-5
	JCL Labels	4-5
	JCL Remarks	4-6
III.	The JCL Command Set	4-7
	/DOS	4-7
	/EXIT	4-8
	/READ	4-8
	/TYPE	4-9
	/PRINT	4-10
	/IF	4-10
	/GOTO	4-12
	/QUEUE	4-12
	/QLOAD	4-13
	/PURGE	4-14
	/JUMP	4-14
	/RESUME	4-15
	/CANCEL	4-15
	/RUN	4-16
	/DEBUG	4-16
	/VOFF	4-17
	/OPTION	4-17
	DSP	4-17
	JCL	4-17
	QUEUE	4-18
	QBYTE	4-18
	DQ	4-18
	CHR	4-18

I. - Job Control Language

DOSPLUS contains a powerful utility program called JCL, which stands for Job Control Language. Actually, JCL is more than a utility program; it is a comprehensive computer language designed to control the operation of the computer's DOS or applications programs. JCL can allow the computer to perform complex, interactive tasks completely unattended. JCL can form the foundation for a user-friendly or menu-driven "front-end" for more complex programs. JCL can even be used to create mini-utility programs for use from the operating system command level.

Invoking JCL

In order to execute JCL procedures, the JCL system must be installed on DOSPLUS. The general syntax to invoke JCL is as follows:

```
JCL (PROC=xx,QUEUE=xx)
```

The optional PROC parameter is used by JCL to determine the size of the procedure buffer area. This is the region of memory which contains JCL program text. The default size of the procedure buffer is 768 bytes. This size may be altered with the PROC parameter, up to a maximum of 4096 bytes.

The optional QUEUE parameter controls the size of the keyboard queue (see below). The default size of the queue is 256 bytes, but the QUEUE parameter may be used to alter the size of this buffer up to a maximum of 4096 bytes.

In order to execute any JCL procedure, use the following syntax:

```
EX jclprog <exp1> <exp2> <exp3>
```

That is, type the letters "EX", follwed by a space, and the name of the JCL program file. The extension /JCL is assumed. Any number of optional parameters may be typed after the JCL program name, and these values may be read by the JCL procedure if desired.

The Keyboard Queue

The heart of the DOSPLUS job control language is the keyboard queue. The keyboard queue is simply an area of memory maintained by the JCL system. JCL provides commands to place data into and retrieve data from the keyboard queue. The purpose of the queue is to substitute the characters in the keyboard queue for any characters typed on the TRS-80's keyboard. This means that if there is any data in the keyboard queue, any program that attempts to fetch data from the computer's keyboard will receive characters from the keyboard queue instead. Once the keyboard queue is emptied, any data requested from the keyboard will be fetched from the keyboard. For instance, assume the following data is present in the queue:

```
BASIC <enter>
LOAD"CALC/BAS" <enter>
LLIST <enter>
```

If the computer is at the DOSPLUS command level, it will request a line of data from the keyboard. Since there is data in the keyboard queue, JCL will provide that data first. Therefore, with DOSPLUS at the command level, it will receive the characters 'BASIC', followed by a carriage return. The DOS will then load and execute the Disk BASIC interpreter. Once BASIC is loaded, it will print its 'READY' prompt and request a line of data from the keyboard. Since there is still data in the keyboard queue (only the line 'BASIC <enter>' having been expended at this point), JCL will return the characters 'LOAD"CALC/BAS"', followed again by a carriage return. This will cause BASIC to load the file named CALC/BAS into memory. After completing that operation, BASIC will once again request a line from the keyboard. One line, 'LLIST <enter>' remains in the queue, and that line is returned to BASIC. BASIC will now execute that command, and finishing with it, request another line from the keyboard. Since the queue is now empty, JCL will not substitute any characters from the queue, and only actual keyboard input will be accepted.

This example illustrates the purpose of the keyboard queue - to provide a substitute for keyboard input. Since JCL allows the user to control what data is placed in the queue, a JCL program can be written to free an operator from the tedious chore of typing monotonous command sequences. And since JCL has decision-making capabilites, it can work in conjunction with an operator, modifying its actions based on the operator's responses.

II. - JCL Program Structure

Like any programming language, JCL has its own program components and structure. All JCL programs are composed of one or more JCL statements. A JCL statement may consist of a JCL command, a variable assignment, a label, or a remark.

JCL program files may consist of numbered or non-numbered lines. The BASIC line editor may be used to create JCL program files if the programs are saved in ASCII format (using the SAVE"filespec",A syntax). The first line of a JCL program file must contain the name of the JCL program, and it must match the name of the program file. For example, the JCL program file named KILLTXT/JCL might contain the following data:

```
KILLTEXT
/TYPE ENTER DRIVE NUMBER
/READ $D
/DOS KILL /TXT:$D,E
/EXIT
```

If the first line of the JCL program file does not match the filename, JCL will report an error. This procedure identification line may also be used to read the values of any variables that may be present on the command line passing control to JCL. For instance, if the command:

```
EX PURGE APR MAY JUN JUL AUG
```

is executed, a JCL program may pick up the data following the JCL filename (APR, MAY, etc.) with an implicit /READ by including variable names in the procedure identification line, as shown below:

```
PURGE $FIL1 $FIL2 $FIL3 $FIL4 $FIL5
```

When JCL executes this procedure, the variables $FIL1 through $FIL5 will contain the values present on the command line.

It is important to note that all JCL command words must be suffixed with a space, and JCL operators (=, EQ, NE, GT, GE, LT, LE) must be prefixed and suffixed with a space

JCL Commands

The DOSPLUS job control language features 17 JCL commands (each of which is explained in section III). Commands inform JCL to perform some action involving modifying the contents of the keyboard queue, outputting or accepting data to or from the outside world, modifying program flow, etc. All JCL commands must be prefixed with a slash symbol, '/'. Many commands can accept some argument, and some commands require an argument, which should follow the command name, separated by a space.

JCL Variables

JCL allows the use of variables. All variables must be prefixed with the dollar sign symbol, '$'. Variable names may be from 1-8 characters in length, and may contain any combination of letters and numerals. JCL variables may store as few as 0, and as many as 8 characters each.

Before JCL executes a program line, it first scans the line to locate any variables. When a variable is found, JCL removes the variable name from the line and replaces it with the value of that variable. For example, let us assume that the variable $FILENAME has the value 'SCHEDULE' assigned to it. The JCL statement:

```
/DOS LIST $FILENAME
```

would be evaluated by JCL to read:

```
/DOS LIST SCHEDULE
```

Values may be given to JCL variables either by the /READ command (see section III) or by the assignment operator, '='. For instance, the JCL statement:

```
$DSCMD = RS232
```

would assign the value 'RS232' to the variable $DSCMD.

Variables may be added together, or concatenated, by simply placing the variable names next to each other. Examine the following JCL program:

```
TESTPROG
$A = FILE
$B = NAME
$C = $A$B
```

In this program the value of the variable $C will be set to 'FILENAME'. Variables and literals may be combined in much the same fashion:

```
FILNAM
$A = FILE
$B = DAT
$C = $A/$B
```

In this case, the variable $C will have the value 'FILE/DAT'.

When naming variables, remember that JCL is only concerned with significant characters in the variable name. This means that, as far as JCL is concerned, the variable names FILE and FILENAME are identical, since the whole of the name FILE may be found within FILENAME. When it is necessary to use similar variable names, be certain that no variable name is wholly contained within another. For instance, although QUERY and QUERY1 are identical to JCL, QUERY0 and QUERY1 are not.

Special Variables

JCL provides two special variables whose value is set by the JCL system itself. These variables may be used within JCL procedures to great advantage.

The first special variable is $ERR. This variable is set upon return from any DOS library command or other program. If no error has occured, this variable should have the value "00". If any error was encountered, $ERR will contain the error code, in decimal, corresponding to the error. $ERR is used within JCL procedures to detect and trap errors.

The other special variable provided by JCL is $LEVEL. JCL procedures can request other JCL procedures, which may in turn call other JCL procedures, and so on. The $LEVEL variable indicates at which level the current procedure is executing. The top level is level 1. If a level 1 JCL program were to call another JCL program, the second program would have a $LEVEL value of 2. If this program requested some other JCL program, that program would set $LEVEL to 3. As each JCL procedure terminates and returns to the upper-level procedure, the $LEVEL variable is decremented to reflect the level currently being executed. Up to 9 levels may be used within JCL.

JCL Labels

JCL allows the use of labels to identify blocks of JCL program text. The format of a JCL label statement is:

```
-label
```

That is, a label statement is always prefixed with a minus symbol, '-'. The label itself may be from 1 to 8 characters in length, and may contain any combination of letters and numerals.

When assigning labels, remember that JCL is only concerned with significant characters in the label name. This means that, as far as JCL is concerned, the label names LOOP and LOOP1 are identical, since the whole of the name LOOP may be found within LOOP1. When it is necessary to use similar label names, be certain that no label name is wholly contained within another. For instance, although ERROR and ERROR1 are identical to JCL, ERROR0 and ERROR1 are not.

JCL Remarks

The JCL system allows the use of remarks, or comment statements within a program. Remarks are not executed by JCL; their only purpose is to allow the programmer to include comments concerning the JCL program in the program text itself. Remark lines should begin with a period, '.'. All subsequent characters (to the end of the line) will be ignored by JCL. For example, look at the following program:

```
JCLPROG
.This program is used to perform a
.global kill of all /TXT files
/TYPE Press enter to kill all /TXT files
```

The lines beginning with a period simply describe the purpose of the program, or document the workings of sections of program code.

It should be noted that remarks will use up procedure area and thus detract from the overall size of the JCL program that can be executed. When the program becomes too large for the procedure area, you will receive an "%% Out of memory %%" error. At that point, you may remove JCL from memory by a system reset or loading a configuration file that does NOT contain JCL and reload JCL with a larger procedure area, or you may edit the program and remove remark statements.

The JCL Command Set

The DOSPLUS job control language contains 17 JCL commands. Each command initiates a particular action within JCL, resulting in a change in the JCL program's status, execution flow, input or output with the outside world, etc. Each JCL command is detailed below.

/DOS

General format: /DOS <expression>

The /DOS command is one of the most often-used commands in JCL. Its purpose is to allow JCL to pass a DOSPLUS command line to the DOSPLUS command interpreter. As an example, the command:

```
/DOS DIR :1
```

would cause JCL to instruct DOSPLUS to perform a directory on the diskette mounted in drive number 1. Variables may be used with the /DOS command, as they may be used with any JCL command. Assuming the variable $DRIVE to have the value ':3', the JCL statement:

```
/DOS FREE $DRIVE
```

would be evaluated as:

```
/DOS FREE :3
```

and therefore display a free space map on the diskette mounted in drive 3.

The /DOS command may be used to execute programs from the DOS command level as well. For instance, the command:

```
/DOS BACKUP :0 :1,USE=Y,DATE="01/31/83"
```

would cause the BACKUP program to be executed.

/EXIT

General format: /EXIT

The /EXIT command is used to terminate execution of a JCL procedure. When the /EXIT is executed, JCL returns control to the next upper-level JCL procedure, if any, or to the DOS command level if there are no upper-level procedures pending.

/READ

General format: /READ $var1 $var2 $var3 . . .

This command is used to accept input from the keyboard or the keyboard queue. The data is placed into one or more JCL variables. When the /READ command is executed, JCL will retrieve a line from the keyboard queue, or it will wait for a line to be entered on the keyboard. When /READ scans a line, it considers the space character as a delimiter; that is, when /READ assigns values to variables, it regards the space as a terminating character. Take, for example, the command:

```
/READ $A1 $A2 $A3
```

If this command is used to /READ the line:

```
DATAFILE :0 :3
```

the variables will take on the following values:

```
$A1= DATAFILE
$A2= :0
$A3= :3
```

Many other characters are also considered delimiters, such as the slash, "/", the colon, ":", comma, ",", and the period, ".". For instance, if the following statement:

```
/READ $FILENAME $EXTEN $PW $DRIVE
```

is executed, and the following data is supplied:

```
PAYROLL/DAT.PAYDAY:3
```

JCL will assign the following values to each variable:

```
$FILENAME=   PAYROLL
$EXTEN=      DAT
$PW=         PAYDAY
$DRIVE=      3
```

Note that each time the /READ command is executed, an entire line of input is expended. Therefore, if the command:

```
/READ $VAR1 $VAR2
```

is executed on a line such as:

```
FILENAM1 FILENAM2 FILENAM3 :1
```

the variables $VAR1 and $VAR2 will contain the values "FILENAM1" and "FILENAM2", respectively. The rest of the data on the line, "FILENAM3" and ":1", however, will be lost. Subsequent /READs will read in data from new lines, not from the remainder of the expended line.

If a /READ command attempts to acquire more data than is present on a line, such as would be the case with the statement:

```
/READ $VAR1 $VAR2 $VAR3
```

and the data:

```
TRANSACT/DAT
```

any fields which are not satisfied will become null; that is, they will contain no data. In the example above, $VAR1 would contain "TRANSACT", and $VAR2 would have the value "DAT". $VAR3, however, would be null.

==========

==========

/TYPE

General format: /TYPE <expression>

The /TYPE JCL command is used to display messages on the computer's video display. For example, the command:

```
/TYPE Press (ENTER) when ready
```

will display the message "Press (ENTER) when ready" on the computer's screen. Of course, variables may be used in conjunction with the /TYPE command. Examine the following JCL procedure:

```
/TYPE Enter filename:
.Read filename from keyboard
/READ $FILE $EXT
$SLASH =
/IF .$EXT NE . $SLASH = /
.Use COPY command on file
/DOS COPY $FILE$SLASH$EXT $FILE/BAK
.Display message for operator
/TYPE File: $FILE Copied
/EXIT
```

This is a simple JCL program which will print the message "Enter filename:" on the CRT, prompting the operator to enter a filename. The filename is placed in the variable $FILE, the extension (if any) in $EXT, and then the COPY command is used to duplicate the file specified by $FILE and $EXT into another file specified by $FILE (with the extension /BAK added). After the operation is complete, the JCL program types the message "File: filespec Copied" on the screen, where "filespec" is the value of the variable $FILE.

The /TYPE command may be used without any text or variable to provide a blank line on the video display.

/PRINT

General format: /PRINT <expression>

The /PRINT command function much as the /TYPE command, above, except the output generated by /PRINT is directed to the system lineprinter instead of the video display. For example, the command:

```
/PRINT Deleted Files:
```

will print the message "Deleted Files:" on the lineprinter. As with /TYPE, variables are often used with the /PRINT command:

```
/PRINT Error code $ERR during file COPY
```

will print the message "Error code xx during file copy" on the lineprinter, where "xx" is the current value of the special variable $ERR.

The /PRINT command may be used without text or variables to provide a blank line on the printer.

/IF

General format: /IF <exp1> <relation> <exp2> /JCL command

The /IF command allows JCL to make logical comparisions of JCL variables and to make decisions based on the outcome of the comparisions. The /IF command recognizes six relational operators: EQ (equality), NE (non-equality), GE (greater than or equal to), GT (greater than), LE (less than or equal to), and LT (less than). These relational operators may be used to compare any two JCL expressions which may be composed of JCL variables and/or constants. Examine the following JCL procedure:

```
LISTER
.This JCL program produces ASCII file
.listings on the CRT or lineprinter.
/TYPE FILE LISTER
/TYPE
/TYPE Enter file name:
/READ $FILE
/TYPE Listing to video or printer? (V/P):
/READ $OUTPUT
/IF .$OUTPUT EQ .V /DOS LIST $FILE
/IF .$OUTPUT EQ .P /DOS LIST $FILE TO @PR
/EXIT
```

In this program, the variable $OUTPUT is used to determine whether a file listing should be outputted to the video screen or to the system lineprinter. When a logical condition is true, JCL will execute the JCL command following the /IF command. This JCL command is contained within the same statement as the /IF command itself. If the logical condition specified by the /IF is not true, JCL will skip the remainder of the /IF statement and continue with the next statement in the procedure. The following simple program illustrates the mechanics of the /IF command:

```
IFPROG
/TYPE DIRECTORY OR CATALOG (D/C)?
/READ $DIRCAT
/IF .$DIRCAT EQ .D /DOS DIR :0
/IF .$DIRCAT EQ .C /DOS CAT :0
/EXIT
```

In this example, if the condition $DIRCAT = D is true, JCL will execute the command /DOS DIR :0. If $DIRCAT is not equal to D and is equal to C, JCL will execute the next statement, /DOS CAT :0.

Note the use of the period symbol in the /IF statements above. This is to guard against the possibility of a null variable in the /IF command. Whenever there is a possibility of a null variable (such as those variables whose values are taken from the keyboard or the queue), the JCL program must make provision for such an eventuality. For example, examine the JCL statements below:

```
/READ $INPUT
/IF $INPUT EQ QUIT /EXIT
```

If the variable $INPUT is null (this can occur if the operator simply presses <ENTER> when prompted for input), JCL will substitute the null value of $INPUT into the line before executing it, yielding:

```
/IF EQ QUIT /EXIT
```

This is, needless to say, meaningless. By placing any character (in addition to the values to be compared) on both sides of the /IF statement, we prevent the possibility of such an error without altering the outcome of the logical test. Modifying the previous example, we obtain:

```
/READ $INPUT
/IF .$INPUT EQ .QUIT /EXIT
```

Now, if the variable $INPUT assumes a null value, JCL will evaluate the line as:

```
/IF . EQ .QUIT /EXIT
```

This is valid, and in this case, the condition is false ("." <> ".QUIT").

/GOTO

General format: /GOTO -label

This command is used to alter normal program flow. With it, program execution may be diverted to any label within a JCL procedure. /GOTO is often used in conjunction with the /IF command, above. For example:

```
GOTOPROG
-LOOP0
/TYPE Enter drive #:
/READ $DRIVE
/IF .$DRIVE EQ . /GOTO -ALLDONE
-LOOP1
/DOS CAT :$DRIVE
/IF $ERR$ NE 00 /GOTO ERROR
/GOTO LOOP0
-ERROR
/TYPE
/TYPE Error code $ERR has occured.
/TYPE Abort or Re-try (A/R):
/READ $INPUT
/IF .$INPUT EQ .R /GOTO -LOOP1
/TYPE Operation aborted
/EXIT
-ALLDONE
/TYPE Procedure terminated
/EXIT
```

In this program, the /GOTO command is used with the /IF command in order to perform a complex series of JCL commands if an /IF condition is met.

/QUEUE

General format: /QUEUE <expression>

The /QUEUE command provides a means of placing data into the keyboard queue. For instance, to place the data "BASIC (F=3)" into the queue, execute the command:

```
/QUEUE BASIC (F=3)
```

Variables may be used with the /QUEUE command. The routine below illustrates the use of variables with the /QUEUE command:

```
/TYPE Enter filename to list:
/READ $FILE $EXT $DRIVE
.create any needed delimters
/IF .$EXT NE . $SLASH = /
/IF .$DRIVE NE . $COLON = :
.load queue with BASIC commands
/QUEUE LOAD"$FILE$SLASH$EXT$COLON$DRIVE"
/QUEUE LPRINT "$FILE - Program listing"
/QUEUE LPRINT
/QUEUE LLIST
.set FORMS and enter BASIC
/DOS FORMS (P=66,L=60,W=80)
/DOS BASIC
/EXIT
```

This program uses the /QUEUE command to place BASIC commands in the keyboard queue. In the example, this is used to create titled BASIC program listings.

==========

==========

/QLOAD

General format: /QLOAD filespec

This command is used to load the keyboard queue with data stored in a disk file. For example, a file could be created with the BUILD command under DOSPLUS (or with a word processor) that contains filenames. A JCL procedure such as that one shown below can load the filenames into the keyboard queue and perform some useful function, such as setting the INV flag with the DOSPLUS ATTRIB command:

```
INVIS $FILE
-GETFIL
/IF .$FILE NE . /GOTO -FILOK
/TYPE Enter filename:
/READ $FILE
/GOTO -GETFIL
-FILOK
/QLOAD $FILE
/QUEUE ???
-MAKINV
/READ $FILNAM $EXT $DRIVE
/IF $FILNAM EQ ??? /GOTO -ALLDONE
$SLASH =
$COLON =
/IF .$EXT NE . $SLASH = /
/IF .$DRIVE NE . $COLON = :
/DOS ATTRIB $FILNAM$SLASH$EXT$COLON$DRIVE,INV
/GOTO -MAKINV
-ALLDONE
/TYPE Procedure complete
/EXIT
```

The /QLOAD command assumes an extension of /TXT on all files unless otherwise specified.

/PURGE

General format: /PURGE

The /PURGE command performs the simple function of emptying, or purging, the keyboard queue. Any data in the queue before /PURGE is destroyed. The command is useful when a JCL procedure has placed data into the queue, that for one reason or another, needs to be removed from the queue (such as an early, abnormal procedure termination).

/JUMP

General format: /QUEUE /JUMP -label

The /JUMP command is not a command in the same sense as a /GOTO, a /TYPE, or an /IF. Rather, the /JUMP command is used in conjunction with the /QUEUE command to place a special character in the keyboard queue. When the special character is retrieved from the queue by another program, JCL will interrupt program execution and transfer control to a user-specified label within the JCL procedure. To illustrate, consider the JCL program below:

```
JUMPPROG
/TYPE Entering BASIC . . .
/TYPE
/QUEUE /JUMP -LOADPROG
/DOS BASIC
/EXIT
.intercept BASIC here
-LOADPROG
/TYPE Enter program to edit:
/READ $FILENAM
/QUEUE LOAD"$FILENAM"
/QUEUE CMD"SR","PRINT","LPRINT"
/RESUME
```

This program will load the DOSPLUS Disk BASIC interpreter. Since the /JUMP command is loaded into the queue, the next time BASIC attempts to retrieve a character from the keyboard driver, it will receive the /JUMP command, causing JCL to take control. In this case, control is transferred to the label -LOADPROG, and the routine located there replaces all PRINT commands in the BASIC program with LPRINTs.

/RESUME

General format: /RESUME

The /RESUME command is used to return back into a program interrupted with the /JUMP command, above. After any desired /JCL processing has taken place, execution of the /RESUME command will cause JCL to transfer control back to the interrupted program at the point JCL intervened. The program example shown under /JUMP illustrates the use of the /RESUME command.

/CANCEL

General format: /CANCEL

The purpose of the /CANCEL command is to allow a JCL procedure which is entered through the /JUMP command to avoid returning to the intercepted program. When the /CANCEL command is executed, JCL returns into the JCL procedure at the line following the command during which the /JUMP command was read from the queue.

For example:

```
PATCH
/TYPE Enter the name of the
/TYPE file to be patched:
/READ $FN $DR
/IF .$DR NE . $DLIM = :
.install patch data in queue
/QUEUE A=4411H,F=,C=00E0
/QUEUE /JUMP -ANYMORE
.invoke patch utility
/DOS PATCH $FN/CMD$DLIM$DR
/TYPE Patch procedure completed
/EXIT
-ANYMORE
/TYPE Mandatory patch(es) installed
/TYPE Do you have any other patches
/TYPE to apply to $FN/CMD (Y/N):
/READ $YESNO
/IF .$YESNO EQ .N /CANCEL
/IF .$YESNO NE .Y /GOTO -ANYMORE
/TYPE Transferring control to Patch
/TYPE utility . . . press <BREAK>
/TYPE when all patches installed.
/TYPE
/RESUME
```

/RUN

General format: /RUN filespec <exp1> <exp2> <exp3>

The /RUN command is used within a JCL procedure to execute another JCL program. When another JCL procedure is executed using the /RUN command, the current JCL procedure's status is saved (including all variables) and the special variable $LEVEL is incremented by one. Since each JCL procedure has a totally separate set of variables (even if the variables have the same name), the only means with which the JCL programs may communicate with one another is through the keyboard queue. Optional parameters may be passed to a JCL program by including them on the /RUN command line. These variables are picked up in the procedure identification line, as previously described.

The following program, which creates five files on a user-specified drive, illustrates the use of the /RUN command:

```
RECURSV $DR
/IF .$DR NE . /GOTO -CREATE
/TYPE %% Missing drive # %%
/EXIT
-CREATE
/TYPE Creating file TEST$LEVEL/DAT:$DR
/DOS CREATE TEST$LEVEL/DAT:$DR
/IF $ERR EQ 00 /GOTO -NOERR
/TYPE %% Error $ERR has occured %%
/TYPE %% Procedure aborted %%
/EXIT
-NOERR
/IF $LEVEL NE 5 /RUN RECURSV $DR
/EXIT
```

/DEBUG

General format: /DEBUG

This command is used to invoke DOSPLUS's DEBUG monitor. When the /DEBUG command is executed, the DEBUG monitor will immediately load and assume control of the computer.

/VOFF

General format: /VOFF

/VOFF is used with the /QUEUE command in order to suppress the usual display of data read from the keyboard queue. If the /VOFF is placed into the queue before the data to be read is placed in the queue, the automatic display will be suppressed. For example, if the command "/QUEUE BASIC" is executed, the queue will be loaded with the data "BASIC". If the computer then returns to the DOS command level, the word "BASIC" will be displayed, and the DOS will load and execute the program. If, however, the command "/QUEUE /VOFF" is given before "/QUEUE BASIC", the word "BASIC" will not be displayed when it is read from the queue.

/OPTION

General format: /OPTION <param>/switch <param>/switch <param>/switch

This command is actually six commands in one. It allows a JCL program to (1) enable or disable output to the video display, (2) enable or disable the JCL statement trace, (3) turn the queue on or off, (4) enable or disable the queue for single-character requests, (5) direct all displayed data into the queue, or (6) recognize or ignore special characters.

Each subcommand under the /OPTION command may be turned on or off by specifying a switch after the subcommand name. For instance:

```
/OPTION QUEUE/Y
/OPTION JCL/N
/OPTION DSP/N
```

The switch, as shown above, consists of a single character, either "Y" (for yes) or "N" (for no), separated from the subcommand name by a slash, "/". Note that several subcommands may be specified on with a single /OPTION command:

```
/OPTION DSP/N QUEUE/N DQ/Y
```

DSP

This subcommand controls whether any data is displayed on the computer's CRT. Normally, the DSP parameter is on, but if the "/OPTION DSP/N" command is executed, all output to the video display is halted until the "/OPTION DSP/Y" command is isssued.

JCL

The JCL subcommand, when enabled, causes JCL to print a trace of its activities while JCL procedures execute. This means that JCL will list each statement before it is executed, and it is a very handy aid for debugging JCL programs. The trace is enabled by executing "OPTION JCL/Y" and it may be disabled by the command "OPTION JCL/N".

QUEUE

This subcommand is used to turn the keyboard queue on and off. In other words, executing the command "OPTION QUEUE/N" will disable the keyboard queue, and any keyboard input requests that may occur while the queue is off must be serviced by the keyboard itself. When the queue is on (its normal state), data in the queue is used to service keyboard data requests.

QBYTE

Many programs require the user to press a single key to perform some function. An example would be the following BASIC program:

```
10 PRINT"PRESS (A) TO ABORT, (C) TO CONTINUE"
20 A$=INKEY$:IF A$="" THEN 20
30 IF A$="A" GOTO 1000
40 IF A$="C" GOTO 2000
50 GOTO 20
```

This program waits for the operator to press either the "A" or the "C" key to perform some operation. The QBYTE subcommand is used to enable or disable the queue from responding to such single-character requests. If "/OPTION QBYTE/N" is executed, the keyboard queue will not provide characters to satisfy single-character requests. Requests for complete lines of data are unaffected.

DQ

The DQ subcommand is used to copy all data output to the video display into the queue. For instance, look at the following program:

```
QCAT
/OPTION DQ/Y
/DOS CAT :1
/OPTION DQ/N
/TYPE File catalog stored in queue
```

This program will place a copy of the file catalog of drive 1 into the keyboard queue. Please note that DQ will still place displayed data into the queue even if the command "/OPTION DSP/N" has been executed. Although the dat is not physically displayed on the CRT, it will still be directed into the queue.

CHR

The CHR subcommand is used to determine how JCL handles "special characters" read from the keyboard queue. Special characters are any characters other than the alphanumeric set (A-Z, a-z, & 0-9). When the command "/OPTION CHR/N" is executed, special characters in the queue are treated as delimiters; that is, they terminate any data being /READ, and they are skipped. Normally, special characters may be /READ into a JCL variable, with the exception of those characters previously mentioned (slash, colon, period, comma).

DOSPLUS IV BASIC Enhancement package

Section	Page #
Introduction	5-1
Installation	5-1
Display changes	5-2
Shorthand commands	5-3
DI (Delete and insert)	5-4
DU (Duplicate)	5-4
DR (Delete and insert with renum)	5-5
REF (Cross referencer)	5-6
SR (Global search and replace)	5-8
SORT (Order BASIC arrays)	5-9
RESOLVE (Remove labels)	5-12
INPUT@ (Controlled screen inputting)	5-13
Label addressing in programs	5-16
OPTION (Model III to 4 BASIC compatibility)	5-18
Error messages (Detailed display of)	5-20

BASIC interpreter enhancement package

Introduction

This portion of the DOSPLUS manual documents those enhancements we provide for the MicroSoft Disk BASIC interpreter. DOSPLUS is shipped with this Disk BASIC interpreter in an unmodified state due to license obligations. The first thing you should do is install the enhancements. This documentation covers the enhancements ONLY. For standard BASIC operation, you will have to purchase a Disk BASIC manual from Radio Shack. This package consists of the following files:

SR	Global text editor
REF	BASIC cross referencer
RESOLVE	Label resolver
SORT	BASIC array sort
BE1	General enhancements #1
BE2	General enhancements #2

Installing your enhancements

Before you can actually use BASIC with DOSPLUS, the BASIC enhancements must be "installed". Which is to say that you must first alter the BASIC/CMD file itself. This is done via the library command APPEND. First, however, you must decide which of the two general enhancement programs you wish to use (i.e. BE1/CMD or BE2/CMD).

As you know, any sort of enhancements to BASIC are going to use some memory. We have tried to keep the amount of memory used by the enhancements to a minimum, going in many cases to an external utility. However, BASIC enhanced by DOSPLUS will not have quite as much free memory as the standard Model 4 BASIC.

However, it is our belief that you will discover programming with the 6.0 PLUS BASIC Enhancements will save you much more memory in eliminated program statements than it will ever cost you to install.

However, to further aid you in your selection, we have included TWO versions of the general BASIC enhancements, BE1 and BE2. The program BE1 includes the INPUT@ controlled screen input routine and, of course, BE2 does not. This represents to you savings of about 200 bytes of memory. It is our suggestion that you read the documentation where it discusses INPUT@ and make the decision, based on that information, whether or not you wish to spend the additional memory.

The specifics

Place a backup copy of the DOSPLUS series Master diskette in drive 0 and reset the machine. Now, decide which enhancement you will use. If it is BE1, enter the command:

```
APPEND BE1/CMD.CMD BASIC/CMD.CMD (CMD)
```

If you decide on BE2 (no INPUT@), use the following commands:

```
APPEND BE2/CMD.CMD BASIC/CMD.CMD (CMD)
```

This will create a copy of BASIC on the disk that contains the enhancements. This copy of BASIC may be overlaid onto existing copies if you use the password "CMD" when entering the COPY command.

Once they're installed

Remember, you cannot use any of the other enhancements unless you install one of the two general enhancement packages. Also remember that the ONLY difference between the two is the presence or absence of INPUT@.

When you enter BASIC now, the screen will not clear. In addition to the standard MicroSoft and Tandy copyright notice, you will see a notice to the effect that the enhancements are copyrighted by Micro-Systems Software Inc.

Finally, the current amount of free memory and the number of file buffers allocated will be displayed as part of the header.

BASIC is not entered or operated in any new fashion. There are no steps to be re-learned. The DOSPLUS enhancements is invisible to the user until they implement one of the extended commands.

SHORTHAND

Our BASIC enhancements include some highly convenient shorthand for most of your commonly used commands and statements. They are divided into two areas: immediate commands and abbreviated statements.

Immediate commands:

Command	Function
up arrow	List preceding line of program
down arrow	List next line of program
shift up arrow	List first line of program
shift down arrow	List last line of program
; (Semi colon)	List first line of program
/ (Slash mark)	List last line of program
. (Period)	List current line of program
, (Comma)	Edit current line of program.

Abbreviated statements:

Abbreviation	Statement
A	AUTO
D	DELETE
E	EDIT
G	GOTO
I	INPUT
K"	KILL
L	LIST
L"	LOAD
N	NAME
R or R"	RUN
S"	SAVE
!	SYSTEM

Any of the immediate commands must be the first character typed for that line. In other words, if you already typed a character and backspaced, you should still press ENTER to get a "fresh" command line before using shorthand commands.

Abbreviated statements (L,E,D,L",S,etc...) may appear anywhere within a program line. When BASIC encounters them, it will expand them to their normal state.

There are certain conditions that will disable shorthand commands. If JCL is active, shorthand commands are disallowed. Also, if you have implemented the "protected program" option, shorthand will not function.

When using shorthand for abbreviated statements, it is no necessary that you put spaces around them as you are typing. When they are expanded, the spaces will be inserted for you.

DI (delete and insert BASIC program line)

This command will allow you to remove a BASIC program line from one location and insert it in another.

==========

DI pln,nln

pln is the present line number.

nln is the new line number.

==========

This command is used to delete a line number in a BASIC program and insert that line into the program at another point.

Example

DI 100,122

This will copy line 100 to line 122 and then delete line 100.

DU (duplicate BASIC program line)

This command will duplicate a BASIC program line from one location to another.

==========

DU pln,nln

pln is the present line number.

nln is the new line number.

==========

This command is used to copy a line number in a BASIC program to another point in the program. The line will now exist at both the old and the new point.

Example

DU 100,122

This will copy line 100 to line 122 and still preserve line 100.

DR (delete and insert BASIC program line with renumber option)

This command performs the same function as DI, but it will alter all references to the line to reflect its new location.

DR pln,nln

pln is the present line number

nln is the new line number

This command is used to copy a BASIC program line from one point in the program to another. It will delete the line at its old location. In that respect, it functions in the same manner as the DI command. However, DR will also renumber any references to the line moved so that program flow is not interrupted.

What this means is that if you have a line 100 that states "GOTO 1000" and you use DR to move line 1000 to line 1050, line 100 will be altered to refer "GOTO 1050". In cases where this is not desired, use DI. By including both commands, we hope to cover all situations.

Example

```
DR 100,122
```

This will copy line 100 to line 122 and then delete line 100. After that, it will alter any and all references to line 100 to reflect its move to line 122.

REF (reference BASIC program text)

This command gives you a comprehensive BASIC program cross referencer.

SYSTEM"REF",option,option...

option can be any of the parameters legal for this command.

Param	Function
S	Single variable, line, or keyword
V	All variables
L	All line numbers
K	All keywords
P	Printer output

This will allow you to reference your BASIC program for line numbers (L), variables (V), or keywords (K). For example:

```
SYSTEM"REF",K,L,V
```

This will reference the program for all three. If you specify a P also (SYSTEM"REF",K,L,V,P), REF will do the same thing, but it will output it to the line printer.

To display a single variable you use the "S" parameter. For example:

```
SYSTEM"REF",S,A
```

Every time the variable "A" occurs in the text will be listed for you. It becomes as specific as you are. For example:

```
SYSTEM"REF",S,A$
```

will only hunt up references to the variable "A" when it is being used as a string variable. And still further:

```
SYSTEM"REF",S,A$(
```

will look up only references to "A" as an ARRAY string variable. It will also take complex variable names like:

```
SYSTEM"REF",S,FINDIT
```

This will look for all occurrences of the variable "FINDIT". The same syntax applies for a single line number or a single keyword. For example:

```
SYSTEM"REF",S,PRINT
```

will reference all the PRINT statements. Please note that when the "S" parameter is specified, the only other information that may appear in the options list is the item being referenced. For example:

```
SYSTEM"REF",S,FNAME$,L
```

will not reference "FNAME$" and then all line numbers. It will produce an illegal function call error. You may, however, include the "P" parameter to get hardcopy of your reference. For example:

```
SYSTEM"REF",S,FNAME$,P
```

is legal.

Please note that the "AS" keyword in FIELD statements is regarded by BASIC as a variable. Therefore, we also see it as a variable. Do not let this surprise you when it happens. It is simply an "oddity" of BASIC.

SR (global editing of BASIC text)

This command allows you to search for and display or replace any ASCII string literal or expression that occurs in BASIC text.

SYSTEM"SR",sexp,rexp,sln-eln

> sexp is the search expression. This can be any valid string expression, a literal, or a combination of both string variables AND literals.
>
> rexp is the replace expression. This can also be any valid string expression, literal, or combination of both.
>
> sln-eln are the optional starting and ending line number. This allows you to restrict your editing to one block of text. If not present, it will be a global edit of the entire text. If you specify "sln" only, it will do only that line number. If you specify "sln-", it will begin at that line number and go to the end of text.

This is a great programmer's tool. It will search for and display or replace any string variable or expression. To engage it, type SYSTEM"SR" (for search and replace) followed by a literal ASCII string, OR any character string or other string variable. After it alters a line, it will list that line showing the change. It operates in two modes: Search mode and Search and Replace mode. For example, if you type:

```
SYSTEM"SR","Test"
```

It will look through the text and list every line with the word "Test" in it. If you type:

```
SYSTEM"SR","Test","NewTest"
```

It will look through the text and every time that it finds the word "Test", it will replace it with the word "NewTest". If you type:

```
SYSTEM"SR","Test","NewTest",100-200
```

It will confine this procedure to lines 100 through 200. You can also use a combination of variables and literals. For example:

```
SYSTEM"SR",":",CHR$(10)+":"
```

This will go through the whole text and insert a line feed in front of every colon.

NOTE: Because BASIC will not allow some control characters to be imbedded in strings (or program text) and printed, be certain that you do NOT use SR to replace ASCII characters with control characters (with two exceptions, line feed as in the above example and horizontal tabs; these two are allowed). To print control characters, you must use the CHR$(x) function of BASIC. You may, if you wish, use a single character control sequence and use SR to replace all of those with the proper CHR$(x) command.

SORT (sort BASIC arrays)

This command allows you to sort BASIC arrays of any type in ascending or descending order. This utility is upward compatible with the CMD"O" array sort in Model III Disk BASIC.

==========

SYSTEM"SORT",exp,(+ or -)AN$(se)+KA$-KA%,TA#,TA!

exp expression to indicate number of elements to be sorted (integer).

+ or - indicates primary key array to be sorted in ascending or descending order. Optional, if omitted ascending order will be assumed.

AN$(se) primary key array. Subscript indicates starting element number.

KA$ next key array. Plus (+) indicates ascending order.

KA% next key array. Minus (-) indicates descending order.

TA# First tag array.

TA! Next tag array.

==========

A "key" array is defined as being an array that SORT will consider when sorting. A "tag" array, on the other hand, is simply "along for the ride". When SORT finds two elements of a key array that need to be swapped, it will swap the corresponding elements of all other key arrays and all tag arrays.

You may have up to ten key arrays, counting the primary array, and up to twenty tag arrays for a total of up to thirty arrays.

You must completely define all "KEY" arrays prior to defining "TAG" arrays. Please note that all key arrays are preceded with a plus (+) or a minus (-) to indicate ascending or descending order. Do not use commas. After you append the first array with a comma, SORT will assume that you are beginning the tag arrays and will consider no more key arrays.

Differing from this is the PRIMARY KEY ARRAY. The primary key array is separated from the element count by a comma for Model III TRSDOS compatibility. If you wish descending order, you may insert an optional minus (-) between the comma and the primary key array name. A plus (+) is also legal but not needed as ascending order is assumed.

When SORT is ordering the arrays, you may interchange string and numeric arrays as you have need. In the example above, we have attempted to illustrate this. You may sort strings only or numbers only if you wish, but the option to mix them together is open to you.

Also, when SORT is ordering the arrays, if it finds a discrepancy in one of the key arrays, it will swap the elements and stop there. In other words, if the primary key array needs to be swapped, it will not even look at any of the other elements. Only when two array elements are the same will a secondary key array be considered for the sort.

Examples

```
SYSTEM"SORT",100,A$(1)+B$-C$,D$,E$,F$
```

This command line would instruct SORT to sort 100 elements of string array beginning with the first element in the array "A$". If it finds a match there, it will attempt to sort by the corresponding two elements in "B$". If it finds a match there, it will sort by the corresponding two elements in "C$". However, "C$" is sorted in descending order. Any time that it swaps an element in any of the key arrays, it swaps it in all the other key arrays and then it also swaps the corresponding elements in "D$", "E$", and "F$" (although the order of these is not important).

The "corresponding element" is defined as being those elements with the same position number. For example, the corresponding elements in the above example would be:

* A$(1) - B$(1) - C$(1) - D$(1) - E$(1) - F$(1)
* A$(9) - B$(9) - C$(9) - D$(9) - E$(9) - F$(9)

This sort is also capable of sorting integer, single precision, and double precision arrays. You may mix and match arrays. For example, to return to the sort command above, you could make "C$", "C#" with no problem. The syntax is identical.

```
SYSTEM"SORT",N%,A$(1)
```

This will function exactly the same as the old Model III Disk BASIC sort. It will sort "A$" in ascending order starting at element one and proceeding for "N%" elements.

Technical notes: Please note that the arrays may ONLY be single dimension and you may not specify a starting element number for any array other than the primary key array.

Sample use

This sample program will create a sorted index for a mailing list.

```
5 CLEAR 2000 : CLS
10 OPEN"R",1,"MAIL/DAT",52
20 FIELD 1,10 AS DUMMY$,20 AS FNME$
30 EF=LOF(1) : DIM A$(EF),RN%(EF)
40 FOR I=1 TO EF
50 GET 1,I
60 A$(I)=FNME$ : RN%(I)=LOC(1) : NEXT I
70 CLOSE
80 SYSTEM"SORT",EF,A$(1),RN%
90 OPEN"R",1,"MAIL/INX",2
100 FIELD 1,2 AS NR%
110 FOR I=1 TO EF
120 LSET NR%=MKI$(RN%(I)) : PUT 1,I : NEXT I
130 CLOSE
```

After this, whenever you want an alphabetical listing of your file, simply open the file "MAIL/INX". Those two byte records contain integer record numbers. Get each record in turn and then get the data record that it points to. Print that data and you will have an alphabetical listing.

Finally, a couple of short notes. If you have used OPTION BASE to alter the base array starting element, this is supported. In other words, if you have set the base starting element for 1, attempting to sort an element 0 will cause an error. Also, please note that long variable names are allowed within SORT.

RESOLVE (remove labels)

This utility allows you to remove label addressing from a program.

SYSTEM"RESOLVE"

There are no parameters for this utility

Label addressing is a great advantage when you are developing software and frees you from many of the constraints that having to use line numbers imposed. If you are going to continue to use this program under a BASIC enhanced with our options, then there will never be a need to remove the labels.

However, some of you will be using the labels for software development and then expecting this program to run under a BASIC not modified by our enhancements. This requires that there be some way to remove the labels.

RESOLVE will remove label addressing from a program and resolve all references to that label to the proper line number. It works completely in memory.

On the first pass, RESOLVE will turn all of the label references into line numbers. On the second pass, it will remove all of the NAME statements. If a NAME statement is the only item on a line, it will be replaced with a remark token.

Example

Assume that you had the following:

```
10 FOR I=1 TO 10
20 GOSUB TEST
30 NEXT I
40 END
50 NAME TEST
60 PRINT "Hello!"
70 RETURN
```

If you were to then execute:

```
SYSTEM"RESOLVE"
```

you would have:

```
10 FOR I=1 TO 10
20 GOSUB 50
30 NEXT I
40 END
50 '
60 PRINT "Hello!"
70 RETURN
```

INPUT@ (controlled screen input)

This command allows you to input string data from anywhere on the screen with control of format and character entry.

INPUT@<pos>,"prompt",fl,it$;var$

> <pos> is the screen position you wish to input at. It will begin here with the prompt string if one is specified.
>
> "prompt" is the prompt message you wish to have displayed on the screen in front of your input field. This must be a literal.
>
> fl is an integer expression that defines field length.
>
> it$ is the item type flag. Should be "$" for alphanumeric or "#" for numeric. If you append an asterisk to this, you set the "return on full field" mode. This may be a literal OR an expression.
>
> var$ string variable that data typed into the input field is passed to BASIC in. Must be a string even if input was restricted to numeric only. Note that this option is separated by a semi colon. This is NOT an option. A comma will not work in that location.

This utility will serve to replace many of the tiresome INKEY$ subroutines that you now have to use. Your current routines (using INKEY$) are being slowed down by BASIC's string handling functions. That fact that you are collecting data via a subroutine that handles strings and is interpreted in BASIC results in very slow keyboard response. INPUT@ will banish these problems.

Although INPUT@ does only limited error checking of itself, it does allow you to do as complex an error check as you wish later.

Your parameters are:

<pos> (Screen position). This can be anywhere from 0 to 1919 (or you may optionally use the "row,col" format). It is the same as a PRINT@ location. Whatever this value is, that is the location that INPUT@ will print the prompt string. If no prompt string was defined, then INPUT@ will put the input field start at that location.

"prompt" (Prompt message). This must be a literal. It will be printed at the location specified by <pos>. If this is not specified, it will be skipped and the input field will begin at that position instead.

fl (Field length). This defines the length of your input field. In can be a value between 1 and 240. INPUT@ will create a visible field of underline characters for this field. It will NOT allow you to overtype the field. Unless you set the "return on full field" option (described next), it will simply pause and refuse to except any more characters.

it (Item type). This controls what type of input will be allowed. You may use a literal or a string expression here. You have two options:

* "$" - Any alphanumeric characters
* "#" - Numeric characters only

"Numeric" characters are defined as:

0-9, decimal (.), plus (+), or minus (-)

By appending an asterisk to the field type specifier, you set the "return on full field" mode. That means when the last character in the field in entered, the statement proceeds. Otherwise, it will wait for an ENTER to be pressed to proceed. For example:

"$*" - Alphanumeric field, return when full.

If ENTER alone is pressed, you will be returned a null string. Also, there will not be a carriage return at the end of a string simply because of ENTER being used to terminate the input.

If the only thing that you press at the prompt is one of the function keys, it will abort at once and return you the value of that function key in the input string. Otherwise, if they are not the first item on the line, then they will simply be part of the received input string.

var$ (Return variable). This is a string variable that you specify for INPUT@ to return the input field to you in. It MUST be a string. Even if you input numeric data only, it will still be returned as a string and you must get the value of it (see VAL in your BASIC manual). This variable must be set off from the list by a semi colon. A comma will not work.

Examples-

```
INPUT@512,"Type in your name: ",20,"$";NA$
```

This will print the defined string at screen position 512, and then print a 20 character field of underlines after it and accept any alphanumeric data into it. It will wait until ENTER is pressed to exit and will return the input data in "NA$".

```
INPUT@(10,30),40,"$*";SI$
```

This will not print a prompt string because one was not defined. With INPUT@, it is not necessary to leave in the extra comma. Simply ignore the prompt field if you don't wish to use it. It will print a 40 character field at screen position row 10, column 30 and terminate when the 40th character is typed.

The return on full field is useful when you are prompting for a single key entry and you wish to preclude the continual pressing of the ENTER key. You simply define a field length of one and to return when field full and as soon as they type a key, off you go.

While inputting data, you may use the following:

* Repeating keys.
* Backspace.
* Erase line (shift back arrow).

Label addressing (indirect branching within BASIC programs)

This function allows you to use indirect addressing within your BASIC programs. To accomplish this, we replaced the NAME function of BASIC with our own. To rename a file from BASIC under DOSPLUS, use the SYSTEM"RENAME" function.

NAME label
GOTO label
GOSUB label

> NAME assigns the specified label to the line on which the NAME statement appears. After that, you reference the label EXACTLY as you would a line number using GOTO and GOSUB statements.

Labels may now be used in place of line numbers. This frees you from having to remember the exact line number that a particular subroutine was located at. Simply assign the subroutine a unique name and reference it by that.

The only restrictions are: (1) labels may NOT contain any reserved words when under OPTION S (see OPTION) and (2) labels may not exceed 240 characters in length.

We have also altered BASIC's RENUM function such that it will not regard labels when renumbering a program.

Labels may contain the letters A-Z, the numbers 0-9, @, or . (period). Labels may only begin with the letters A-Z or the "@" character. When starting a label with the "@" character, you must follow it with a letter in the A-Z range. You should not use labels with the following BASIC statements:

ELSE
THEN
ERL
DELETE
RUN

That doesn't mean that you cannot use the statement "IF A=1 THEN GOTO TEST". The "GOTO" will set off the label. You should not, however, use "IF A=1 THEN TEST". Don't use labels directly with those statements.

Please note that a label must be the first statement on a line. For example:

```
10 NAME TEST:FOR A=1 TO 10
20 Other program here ...
100 GOTO TEST
```

The NAME statement is the first item on the line. If this is not the case, the name statement will be regarded as a comment and any attempt to reference it will result in an error.

Examples

```
10 CLEAR 1000 : DEFINT I
20 NAME START
Other program lines ...
1000 GOTO START
```

In this example, line 20 has been assigned the label "START". Later, at line 1000, the program issues the command to "GOTO START". This would send program control back to line 20.

```
10 NAME READINPUT
```

This is an example of an invalid label under OPTION S. This label contains the reserved words "READ" and "INPUT". BASIC will reject this as a label. Again, consult the enhancements to the OPTION command to learn about reserved words in labels.

OPTION (engage special options)

This command, present in standard BASIC, has been enhanced to allow Model III BASIC upward compatibility. It will remove the need for spaces around keywords.

==========

OPTION (param)

param is the optional indicator.

Your parameters are:

S	"Short form". Compatible with Model III BASIC.
L	"Long form". Standard Model 4 configuration. This is the default value.

==========

Standard Model 4 BASIC requires that you enter spaces around keywords. This is done so that you may use reserved words in long variables (and now labels, too). However, this also means that programs that have been written with the Model III may not convert easily to the Model 4.

Without the spaces, Model 4 BASIC doesn't know keywords. This allows keywords in variables, since without the spaces there is nothing special about the string of characters. We now allow you to alter this.

In the short form, reserved words themselves are once again significant with or without spaces. This is like the Model III. In the long form, of course, BASIC will react in the standard method.

These also cause the same effect on labels. Under OPTION S, you may not use a reserved word in a label. Under OPTION L, this is allowed.

This causes NO other changes. 40 character variable names are still valid. It only affects keyword recognition without spaces.

In order to transfer a program to Model 4 BASIC using this option, save the file in ASCII and transfer it (either directly or via CONV) to your TRSDOS 6.0. Enter BASIC in the standard manner and type:

OPTION S <ENTER>

Then load the file. The needed spaces will be inserted as the file is loading. Once the file is loaded, you may return to the standard form by typing:

OPTION L <ENTER>

You may switch back and forth at will.

Also, when under OPTION S, there is no need to include spaces as you are typing in the command line. For example, under OPTION L (the default), the line:

10 FORI=1TO10

will be seen as the variable "FORI" being equal to the value "1TO10". The same statement under OPTION S would expand to:

10 FOR I=1 TO 10

automatically. However, under OPTION S the label:

20 NAME READKEY

would become:

20 NAME READ KEY

and its value as a label would be lost because "READ" is a keyword. Under OPTION L, it would be left alone.

So you see, each method has advantages and disadvantages. However, without OPTION S, you could never load and run a program that did not already have spaces around the keywords. Therefore, if you use OPTION S for nothing but loading existing programs, it is STILL very useful.

Error messages (detailed error message display)

This feature of our BASIC Enhancements is not really a command in the usual sense, but rather is a manner in which the system functions that deserves to be documented.

When an error occurs under BASIC, the error message will be printed on the screen along with the offending statement. An arrow will identify the statement that contains the error.

For example:

```
10 FOR I=1 TO 10
20 PRINT "THIS IS A TEST",
30 X=C+2 : GOSUB : NEXT I
```

In this example, there is a "Undefined line number" error in line 30. In the middle of our loop, we issue a GOSUB with no line number or label.

The printout will appear something like this:

```
Undefined line number in 30
--->GOSUB : NEXT I
```

The arrow always points to the statement that contains the error, no matter how large the line. It does not point to the element within the statement that is incorrect. That is for you to determine.

DOSPLUS IV Drivers and Filters

The following are the drivers and filters included with DOSPLUS IV:

Drivers:

Driver	Description	Page #
FILE	(Create pseudo disk inside a disk file)	6-1
MEMDISK	(Create pseudo disk in memory)	6-2
MKEY	(Multiply one keystroke into many characters)	6-9
SPOOL	(Set up printer spooler in memory)	6-12

Filters:

Filter	Description	Page #
DVORAK	(Re-define TRS-80 QWERTY keyboard into Dvorak)	6-15
EPSON	(Offset printer graphics codes for Epson MX-80)	6-16

FILE

This driver will allow you to create a pseudo disk drive with a disk file.

ASSIGN :ds FILE filename param=exp...

:ds is the drivespec of the slot you wish to use for the FILEDISK. This will be any valid two character drive name.

FILE is the name of the driver being installed.

filename is the name of the FILEDISK. If no extension is given, the extension /PDS (for "P"artitioned "D"ata "S"et) will be used.

param=exp... is one or more of several optional parameters to indicate when or how big to create the FILEDISK.

Your parameters are:

SIZE=value	Indicates the size (in records) of the FILEDISK.
INST	Indicates that the FILE driver is already installed. By specifying INST=N, you may suppress a second loading of the driver.

Abbreviations:

SIZE	S
INST	I

A FILEDISK is a disk file that is used by the system exactly as one might use a disk drive. It can store and retrieve programs and data just as a standard disk drive. The usage of directory space is much more efficient, though, because you have files within a file. A FILEDISK may hold up to 112 files (at least two of which will always be reserved for system use), but the FILEDISK itself will only use one directory space on the actual drive.

Generally, FILEDISKs are used to store number of smaller, unchanging programs (like utilities) that would otherwise occupy more space and use up the directory in a rapid fashion. You would create the FILEDISK, copy all of the desired files into it and then turn it on and off as you needed the files.

To install the FILEDISK, select one of the slots in the CONFIG display (see CONFIG) that does not currently have a drive assigned to it (e.g. is set to "NIL"). It is not required that you do this, but if you install the FILEDISK on a drive slot that currently has a drive active, the ability to access that drive will be lost (any reference to that drive name will go to the FILEDISK instead).

Once you have selected which drive slot you are going to allocate to the FILEDISK, you must decide which disk is going to contain the file. The disk must have enough free space on it to hold the entire file (whatever size you have specified).

Use the ASSIGN command to install the driver and at that time specify the disk drive to be used and the filename desired for the FILEDISK. For example, if we desired to create a FILEDISK that was 1000 sectors in length, we must have 1000 sectors free on the disk drive. Since a standard single sided 40 track disk only has 720 sectors to start with and some of these are reserved for system usage, a 1000 sector FILEDISK will usually exist on a double sided 80 track drive or a hard disk. For this example, let us assume that drive ":2" is a double sided 80 track floppy disk drive. We have decided that drive slot ":9" is currently empty and will be used for the FILEDISK. We have decided to name the FILEDISK "UTILITY/PDS". The /PDS extension is often used to signal that the file is in fact a FILEDISK (FILEDISKs sometime being referred to as "Partitioned Data Sets"). The command used would be:

```
ASSIGN :9 FILE UTILITY:2 SIZE=1000,INST=Y
```

By specifying ":9", we inform the driver what drive slot to use. The "UTILITY:2" file specification informs the driver what disk drive to install the FILEDISK on and what to name the FILEDISK. The "SIZE=1000" parameter indicates to the drive to create the file and use 1000 sectors of disk space. The "INST=Y" parameter indicates that the driver itself is not yet in memory and needs to be loaded. Note that INST defaults to "Y" and if you wish the driver to install itself, you may simply omit the INST parameter altogether. It is only when you wish to suppress the loading of the driver that you need to specify INST=N.

You have two parameters with FILE. They indicate how large to create the FILEDISK and whether or not to install the driver.

SIZE. This parameter, if present, informs the FILE driver that we are going to create a new FILEDISK and gives it the size (in sectors) of the FILEDISK. This parameter is only effective when you are first creating the FILEDISK. If it is included when the file is already in existence, it will be ignored. SIZE must be a value between 48 and 3200. If there is not enough space on the disk for the FILEDISK you have specified, you will be given a "Disk space full" error.

Note: In the event that an error occurs during the creation of a FILEDISK, the file that was specified will have been created in the directory, but no space will be allocated to it. You must kill this file before re-installing the FILE driver because as far as the driver is concerned, that file exists and it will not re-create it.

INST. This parameter will instruct the FILE driver whether or not to load itself into memory. When using FILEDISKs, there will be many times that you wish to assign the drivespec to another FILEDISK (i.e. you started with UTILITY/PDS and now you wish to switch to UTILITY1/PDS). Since you have already installed the FILE driver for that drivespec, there should be no reason that you would have to do this again. By specifying the INST=N parameter the second time that you load the FILE driver (for each separate drivespec), you will avoid using more memory than is necessary by not reloading the same driver time and time again.

Every time that you call the FILE driver, it will display the FILE driver program header to indicate that it is active. If you are initializing a file for the first time, the message "File initialized" will appear. If you are installing the driver, the message "Driver installed at xxxxH" will inform you where in memory FILE driver has loaded. If the FILEDISK is already created and the driver already loaded (e.g. no SIZE or INST=N parameters), then only the FILE driver header will appear. Note these messages, because they inform you as to what actions the FILE driver is taking.

You may perform any functions on a FILEDISK that you might on a normal disk drive. The only exceptions would be floppy disk only operations like BACKUP and FORMAT. You may even use DISKZAP and DIRCHECK with FILEDISKs. FILEDISKs will be included in any global search operations just as any standard disk drive might be.

If you have specified the SIZE parameter because you believe that you are initializing a new file and the message "File initialized" does NOT appear, then one of two items has happened. Either: 1) the FILEDISK is already there and the parameter was not needed or 2) some OTHER file was there and the FILE driver will not function with it. When you are in doubt as to whether a file is a FILEDISK or not, attempt to access it after FILE driver is installed. If you receive errors and cannot access the file, then more than likely it is not a FILEDISK.

If you do NOT specify the SIZE parameter and the FILEDISK does not exist, FILE driver will create the FILEDISK with a default size of 100 sectors.

Anytime that you do not specify the INST=N parameter, FILE driver is going to load itself into memory. Repeated use of the parameter for the same drivespec is wasteful. You must, however, use a separate installation of the FILE driver for each drivespec used. For example, assume that we created the FILEDISK as in the example above. That was done using drivespec ":9". If we wished to assign drivespec ":8" to a second FILEDISK and have them both operational at the same time, we would not use the INST parameter (allowing it to default to INST=Y and load the driver). If we wished to re-assign the drivespec ":9" to another FILEDISK, we would use the INST=N parameter in the command. So then, the driver is installed once, and ONLY once, for each drivespec used with a FILEDISK.

If you specify the INST=N parameter to suppress the loading of the driver and the FILE driver is NOT already installed for the specified drivespec, the system will not function properly, if at all.

You may save a configuration file (see SYSTEM) with the FILE driver active and installed if you wish. However, if you do this, you must not kill or alter the FILEDISK that is active in any way. When the configuration file is reloaded, the system will expect find the same FILEDISK at the same location on the real disk.

So then, in summary, the FILEDISK is most useful in environments where the extremely high capacity of the disk drives cause you to run out of available directory space before using all the disk space. By using a FILEDISK to "pack" several smaller files inside one larger file, you make disk allocation and directory space usage MUCH more efficient. By constantly shifting between FILEDISKs, you can alter what programs you have access to in seconds.

With hard disks especially, space is allocated to each file in large chunks called "granules". On a hard disk, these may sometimes be as large as 16 or 24 sectors each. What this means is that every time a file needs more space allocated to it, this space will be allocated in fairly large pieces. With the kind of large, ever growing data files that hard disks were designed for, this is most efficient. For smaller, unchanging files, this can sometimes cause more disk space to be used than is optimum. Within a FILEDISK, the granules are 2 sectors in length. While you are in no danger of running out of space on a hard disk with large granules, packing smaller files into FILEDISKs will result in a more efficient overall use of the hard disk.

All the user needs to remember is that the FILEDISK, once assigned, works just like a disk drive to the operating system. Use of the FILEDISK will help the user maximize the efficiency of disk and directory space usage on high capacity disk drives. Think of them as what they are; pseudo disks that allow you to place many smaller files inside of one larger file. In both theory and implementation, they are both simple and effective to use.

Examples:

```
ASSIGN :6 FILE FDISK/LIB:4 SIZE=500,INST
ASSIGN :6 FILE FDISK/LIB:4 S=500,I
ASSIGN :6 FILE FDISK/LIB:4 S=500
```

This command will create a 500 sector FILEDISK named FDISK/LIB on the disk drive ":4" (assuming there is room). It will also cause the FILE driver to install itself for drive ":6".

```
ASSIGN :6 FILE FDISK1/LIB:4 INST=N
ASSIGN :6 FILE FDISK1/LIB:4 I=N
```

This command will switch the FILEDISK that drive ":6" is addressing in the example above. This command assumes that FDISK1/LIB already exists on drive ":4". It will not reload the driver because it has already been loaded. If the FILEDISK FDISK1/LIB did not happen to exist, it would be created with a default size of 200 sectors.

```
ASSIGN :9 FILE UTILITY:7 SIZE=100000,INST
ASSIGN :9 FILE UTILITY:7 S=100000,I
ASSIGN :9 FILE UTILITY:7 S=100000
```

This command is incorrect because the maximum size of a FILEDISK is 3200 sectors. The value given for the parameter is TOO large.

MEMDISK

This driver will allow you to create a pseudo disk drive in memory.

ASSIGN :ds MEMDISK param=exp...

:ds is the drivespec of the slot you wish to use for the MEMDISK. This will be any valid two character drive name.

MEMDISK is the name of the driver being installed.

param=exp... is one or more of several optional parameters to indicate where or how big to create the MEMDISK.

Your parameters are:

KILO=value	Indicates the size (in kilobytes) of the MEMDISK when you wish to locate it in primary memory.
BANK1	Indicates that you wish to allocate the lower 32K of the additional 64K (if present) to the MEMDISK.
BANK2	Indicates that you wish to assign the upper 32K of the additional 64K (again, if present) to the MEMDISK.

Abbreviations:

KILO	K
BANK1	B1
BANK2	B2

A MEMDISK is a portion of memory that is used by the system exactly as one might use a disk drive. It can store and retrieve programs and data just as a standard disk drive. The speed of data transfer is of course much faster, however all data is lost when the machine is turned off.

Generally, MEMDISKs are used as temporary storage. You will install the MEMDISK, load it with data from a floppy, manipulate the data, and at the end of the session offload the data to the floppy for permanent storage.

To install the MEMDISK, select one of the slots in the CONFIG display (see CONFIG) that does not currently have a drive assigned to it (e.g. is set to "NIL"). It is not required that you do this, but if you install the MEMDISK on a drive slot that currently has a drive active, the ability to access that drive will be lost (any reference to that drive name will go to the MEMDISK instead).

Once you have selected the slot for the MEMDISK, simply use the ASSIGN command (see ASSIGN) and install the MEMDISK for that drive. For example, if you selected a slot that had a drive name of ":9", the command would be:

```
ASSIGN :9 MEMDISK B1
```

This would install a 32K MEMDISK referenced by the name ":9". It would be located in the first 32K of the extra 64K (if installed). If the additional memory is NOT installed, an error will result.

You have three parameters with MEMDISK. They are used to indicate where in memory to locate the MEMDISK and how large to make it.

KILO. This parameter, if present, must be set equal to a value that tells the driver, in kilobytes, how large to make the MEMDISK. If you use the KILO parameter, MEMDISK will assume that you wish to use what is called "primary memory". This is your standard 64K memory area. Machines with greater than 64K may use the upper memory for the MEMDISK if desired. However, with having this parameter, 64K machines could not use the MEMDISK at all. KILO must be a value between 6 and 20. If there is insufficient free memory to hold the MEMDISK you have specified, you will be given an error and have to try again with a smaller value.

Note: You cannot combine the KILO and either BANK parameter. If you are using primary memory for a particular MEMDISK, you cannot have the same MEMDISK also using one of the upper banks. In other words, one MEMDISK cannot exist in both 64K sections of memory at the same time. You may, if you wish, combine both BANK parameters for a 64K MEMDISK, but you cannot use both primary and additional memory for the same MEMDISK.

BANK1. This parameter is used to indicate that you wish to install the MEMDISK in the lower 32K of the additional 64K, if that 64K is present. This parameter takes no value. If given, MEMDISK will use the entire 32K bank. Remember, if you have not had the additional 64K installed in your machine (for a total of 128K), this cannot be used as the memory it allocates isn't there.

BANK2. This parameter is used to indicate that you wish to install the MEMDISK in the upper 32K of the additional 64K, if that 64K is present. This parameter also takes no value. If given, MEMDISK will use the entire 32K bank. As with the BANK1 parameter, if the machine doesn't have the extra 64K, use of this parameter will cause an error.

You may combine the BANK1 and BANK2 parameters to create on continuous 64K MEMDISK that uses the entire additional section of memory. If either bank is already in use, or for some other reason not available, MEMDISK will report an error and abort.

MEMDISKs function, to the DOS, exactly as a regular disk drive (only MUCH faster!). You may have a maximum of 46 files in a MEMDISK directory. The largest possible MEMDISK is 64K. This would be using both banks of the additional memory. You may perform any functions on a MEMDISK that you might on a normal disk drive. The only exceptions are floppy disk only operations such as FORMAT and BACKUP. You may even use DISKZAP and DIRCHECK with MEMDISKs. MEMDISKs will be included in all global operations, just as a standard disk drive might be.

The most MEMDISKs you could have at any one time is three. One in primary memory, and one each in the additional banks. When you install the MEMDISK driver, if you do not specify any parameters (i.e. ASSIGN :9 MEMDISK), you will be prompted with regard to which area of memory to use and how large to make the MEMDISK (if applicable). If you are being prompted, it will not prompt you for any invalid areas of memory. In other words, it will not ask you to use one of the additional banks if you only have a 64K machine.

When the MEMDISK installs, it will display three items of information on the screen for you. One, it will tell you where in memory the driver itself loaded (i.e. Driver installed at xxxxH). Two, it will tell you what area of memory contains the MEMDISK (i.e. Queue placed at xxxxH). Third, it will tell you if it has used either or both of the banks (i.e. Bank x now in use). Once this display is done, the MEMDISK is installed and you will be returned to the DOS command level. At that point, you may begin to use the MEMDISK.

Please do NOT save a configuration file with a MEMDISK installed. They should NOT be a permanent part of the system at all. Once a MEMDISK is installed, you may only remove it by rebooting or loading a configuration file that removes it. Be certain to offload any desired data before removing the MEMDISK or turning off the machine. MEMDISKs are by no means permanent!

Within a MEMDISK, space allocation is extremely efficient. In the DOSPLUS system, space is allocated in chunks called "granules". On a standard 40 track single sided double density diskette, these granules are 6 sectors in length. This means that every time a file needs new space allocated to it, this is done in 6 sector chunks. For normal operations, this is the most efficient configuration possible. To use smaller granules would cause the system to constantly be moving to the directory to allocate new space. This would prove too slow.

In a MEMDISK, however, space is at a premium. And the one item we will never be short of is speed. Therefore, we can make these granules much smaller. Inside a MEMDISK, granules are 2 sectors in length. When copying files to a MEMDISK, do not be surprised to see that the files get smaller in allocated space. Because of the highly efficient 2 sector allocation, less can become more where disk space is concerned.

Examples:

```
ASSIGN :6 MEMDISK KILO=10
ASSIGN :6 MEMDISK K=10
```

This command will cause the MEMDISK to install in primary memory. The MEMDISK will be 10K in size.

```
ASSIGN :9 MEMDISK BANK1
ASSIGN :9 MEMDISK B1
```

This command will create a MEMDISK in the additional 64K memory. It will be 32K in size and located in the first bank. You will reference it as drive ":9".

```
ASSIGN :AA MEMDISK KILO=15,BANK1
ASSIGN :AA MEMDISK K=15,B1
```

This is an invalid command. You cannot use both primary and additional memory for the same MEMDISK.

```
ASSIGN :15 MEMDISK BANK1,BANK2
ASSIGN :15 MEMDISK B1,B2
```

This command will create the largest possible single MEMDISK (e.g. 64K). It will be located in the additional memory and use both banks.

```
ASSIGN :8 MEMDISK KILO=3
ASSIGN :8 MEMDISK K=3
```

This is invalid because the minimum size of a MEMDISK is 6K. The value for the parameter is TOO small.

```
ASSIGN :8 MEMDISK KILO=30
ASSIGN :8 MEMDISK K=30
```

This command is invalid to the other extreme. The maximum size of a MEMDISK in primary memory is 20K. The value given for the parameter is TOO large.

```
ASSIGN :B MEMDISK BANK1=10
ASSIGN :B MEMDISK B1=10
```

This is incorrect because when specifying either of the bank parameters, MEMDISK assumes you wish to use the entire 32K bank. You may not give the bank parameter a value. Therefore, this command is invalid.

```
ASSIGN :6 MEMDISK BANK2
ASSIGN :6 MEMDISK B2
```

This command will create a 32K MEMDISK in the additional memory area. The second bank will be used.

MKEY

This driver will allow you to load and use a MacroKEY file.

ASSIGN @KI MKEY filename

@KI is the default name of the keyboard device. If you have altered this with RENAME, use the correct name here.

MKEY is the name of the drier being installed.

filename is the name of the file that contains the MacroKEYs.

A MacroKEY is any single key value that has sequence of keystrokes assigned to it. When this single key value is entered (via the CLEAR-key method), the sequence of keystrokes is actually transmitted. Hence, the letter "A" could become the word "APPEND". MacroKEYs may be up to 80 characters in length and contain imbedded carraige returns. MacroKEYs may also be linked together to form one larger statement.

Thus, MacroKEYs actually multiply your keystrokes. One keystroke can result in potentially hundreds of characters. To use the MacroKEYs, you must first create a text file on the disk that contains the information for the MacroKEYs. This text file may be created with the BUILD command (see BUILD) or with a word processor. Be certain that if you use a word processor, it will save with an exact end of file and that it doesn't just use some internal pointer to mark the end of text. If you are not certain about the word processor you intend to use, use the BUILD command instead.

You transmit the MacroKEYs by pressing the CLEAR key and while holding the CLEAR key down, press the key that has the Macro defined. There are four exceptions to this rule. The BREAK and function keys do not require CLEAR to transmit. We will explain this in a moment.

In this text file, you will adhere to the format "key=macro". For example, consider these:

```
"A"=APPEND
41=APPEND
```

Notice that in the first example, you are using a quoted string literal. Either single or double quotes are legal. The important item is that if you wish to express the value as a literal, it must be encased in quotes. You may also specify the value to be considered as a two digit hexadecimal value (Note: Do NOT place an "H" on the end of the value. Hexadecimal is assumed here). This allows you to assign MacroKEYs to keys that you could not otherwise enter because they could not be typed from the keyboard. In our example, because 41H is "A", the above two samples are identical in function.

An example of needing to enter values you cannot easily type are the function keys. F1 returns the value 81H when pressed, F2 returns 82H, and F3 returns 83H. The BREAK key, when pressed, returns 80H. All of these values are outside the standard range of ASCII characters (which ends at 7FH). Therefore, they cannot be entered as literals. That is also the reason they do not require you to hold down the CLEAR key to send. But if you define a MacroKEY as:

```
81=BASIC MENU/BAS (F=1)
```

When you press the F1 key, the text "BASIC MENU/BAS (F=1)" will be sent. By defining the the BREAK key as a Macro, you can in effect alter the function of the key to do what you wish.

When creating your MacroKEY text file, you have two special characters: & and \. The ampersand (&) indicates that you want to link two Macros together. The backslash (\) is an implied carraige return (Note: To obtain a backslash, type CTL-/. In other words, hold down the control key and press the slash (/) key.). To illustrate this:

```
"A"=The quick brown fox jumped over the &B
"B"=lazy red dog.\
```

When CLEAR-A is pressed, the phrase "The quick brown fox jumped over the " will be sent. MKEY will see the link to the Macro for "B" and transmit "lazy red dog." followed by a carraige return because of the backslash at the end. Simply put, the ampersand accomplishes the same effect as pressing another key immediately after the first is done and the backslash has the same effect as pressing ENTER.

So, in review, the steps to using a MacroKEY are:

(1) Create, using BUILD or a word processor, a text file that contains the list of keys defined and the macros defined for them. Use the format "key=macro" as explained above and the special characters (& and \) as needed.

(2) Install the MKEY driver using the ASSIGN command as shown above. Specify the name of the text file that contains the macros at this time.

(3) To transmit the macros after installation, press the CLEAR key and while holding it press the key that has the macro defined for it. Exceptions are the BREAK, F1, F2, and F3 keys (80H, 81H, 82H, and 83H, respectively) which do not require the CLEAR key.

The MKEY driver will install for the keyboard (@KI) device only. Upon installation, it checks to see that it has been correctly installed. You may rename this device, although not suggested, but any attempt to install this for anything other than system device $01 will produce an error.

MKEY will not honor case in key definitions. There is no difference, in an MacroKEY text file, between "A" and "a". Case is honored in the macro itself, just not in the key value being defined. As with all rules, there is one exception. The function keys (formerly 81H, 82H, and 83H respectively), when shifted return new values (91H, 92H, and 93H respectively). This means that in a macro file such as this:

81=DIR
91=FREE

The text "DIR" will be sent when F1 is pressed and "FREE" will be sent when the SHIFT and F1 keys are pressed together. You have a total of 76 possible keys to define as macros. Always keep in mind that you may re-program the BREAK and functions keys by assigning the proper values.

To exchange one set of macros for another is simple. All you must do is load a second text file. Assume that you have created two text files: MKEY/TXT and MKEY1/TXT. To install the former, use the command:

ASSIGN @KI MKEY MKEY

Note that the first "MKEY" uses the default extension /DVR for the driver and the second "MKEY" uses the default extension /TXT for the text file. This command will install the driver and load the macros. To change to the latter of the two files, use the command:

ASSIGN @KI MKEY MKEY1

This time, MKEY will detect that it is already installed and only exchange the table of macros (e.g. load the text file). It will allocate or de-allocate memory as needed. This dynamic allocation of memory means that if you load a smaller text file after a large one, you will actually use less overall memory. The same amount for the driver, because it didn't go anywhere, but less for the table.

When the MKEY driver installs, it will display two items of information. One, it will tell you where in memory the driver is located (i.e. Driver installed at xxxxH). Two, it will indicate where in memory the macro table is located (i.e. Table located at xxxxH). When only the table has been changed (e.g. the driver was already installed), only the latter statement will appear. Once these statements have appeared, the driver is installed and you will be returned to the DOS command level and may begin using the MacroKEYs.

You may include the MKEY driver in a configuration file if you wish. Once the file is loaded, all macros will again be in effect. Once a MacroKEY file is loaded, though, the only way to de-install the driver (e.g. remove it from the system completely) is to either re-boot or load a configuration file that doesn't contain the MKEY driver.

Examples:

ASSIGN @KI MKEY MKEY

This command will assign the keyboard device (@KI) the MacroKEY driver (MKEY/DVR) and load the text file (MKEY/TXT) to define the macros. It will scan all drives for the file.

ASSIGN @KI MKEY

This is an invalid command because no text file was specified. You must specify the text file in the command line. MKEY will not prompt you.

SPOOL

This driver will allow you, when installed, to spool the output of the printer device.

ASSIGN @PR SPOOL param=exp...

@PR is the default name of the printer device. If you have altered this name with RENAME, indicate the new name here instead.

SPOOL is the name of the driver being installed.

param=exp... is one or more of several optional parameters to indicate where or how big to create the spooler.

Your parameters are:

CHRS=value	Indicates the size (in characters) of the spooler when you wish to locate it in primary memory.
BANK1	Indicates that you wish to allocate the lower 32K of the additional 64K (if present) to the spooler.
BANK2	Indicates that you wish to allocate the upper 32K of the additional 64K (again, if present) to the spooler.

Abbreviations:

CHRS	C
BANK1	B1
BANK2	B2

A spooler is a program that captures the data intended for the printer and holds it in an area of memory called the spool buffer. It then outputs this data to the printer whenever the printer is ready to receive it. This has the advantage of allowing the software to output to the spooler at the maximum speed of the computer and allowing the spooler to output to the slower printer as the printer is able to keep up.

Generally, a spooler will be used with an applications package that requires a great deal of printer output. Printers being slower than computers (with regards to the output speed of the computer as opposed to the print speed of the printer), you will often times spend as much time waiting on a printout as processing data. With the spooler, you may print to the buffer and continue with the program while the printer is catching up.

To install the spooler, simply use the ASSIGN command (see ASSIGN) and install the spooler on the printer device. The spooler will ONLY work with the printer device. It is not designed to spool RS232 output or anything else, for that matter. The spooler will check and make certain that it is being installed on system device $02. To install the spooler, in a general manner, the command would be:

```
ASSIGN @PR SPOOL B1
```

This would install a 32K spooler in the first 32K of the extra 64K (if installed). If the additional memory is NOT installed, an error will result.

You have three parameters with spooler. They are used to indicate where in memory to locate the spooler and how large to make it.

CHRS. This parameter, if present, must be set equal to a value that tells the driver, in characters, how large to make the spooler. If you use the CHRS parameter, SPOOL will assume that you wish to use what is called "primary memory". This is your standard 64K memory area. Machines with greater than 64K may use the upper memory for the spooler if desired. However, with having this parameter, 64K machines could not use the spooler at all. CHRS must be a value between 2 and 20,000. If there is insufficient free memory to hold the spooler you have specified, you will be given an error and have to try again with a smaller value.

Note: You cannot combine the CHRS and either BANK parameter. If you are using primary memory for a particular spool buffer, you cannot have the same spooler also using one of the upper banks. In other words, one spooler cannot exist in both 64K sections of memory at the same time. You may, if you wish, combine both BANK parameters for a 64K spool buffer, but you cannot use both primary and additional memory for the same spooler.

BANK1. This parameter is used to indicate that you wish to install the spooler in the lower 32K of the additional 64K, if that 64K is present. This parameter takes no value. If given, spooler will use the entire 32K bank. Remember, if you have not had the additional 64K installed in your machine (for a total of 128K), this cannot be used as the memory it allocates isn't there.

BANK2. This parameter is used to indicate that you wish to install the spooler in the upper 32K of the additional 64K, if that 64K is present. This parameter also takes no value. If given, spooler will use the entire 32K bank. As with the BANK1 parameter, if the machine doesn't have the extra 64K, use of this parameter will cause an error.

You may combine the BANK1 and BANK2 parameters to create on continuous 64K spooler that uses the entire additional section of memory. If either bank is already in use, or for some other reason not available, SPOOL will report an error and abort.

The spooler's function, for the most part, will be invisible to the user. It is what is referred to as a "background task". In other words, while you are running your program (doing something else in the "foreground"), the spooler is outputting characters to the printer (in the "background"). When you install the spooler, SPOOL checks to see if it has already been installed. If it has, it will report an error and abort. This means that you cannot alter the size of the spool buffer once it is installed. You must remove the spooler altogether and re-install it.

You cannot have more than one spooler at a time, even if you have used ASSIGN and LINK to create a second printer device. The spooler will only work with device slot number $02, which is reserved for the printer. If you have located the spooler in primary memory, you cannot locate it also in the upper banks. The reverse is also true.

If you do not specify any parameters when you assign the spooler (i.e. ASSIGN @PR SPOOL), SPOOL will prompt you for all needed information. You will be prompted as to which areas of memory to use and how large to make the buffer. SPOOL will not prompt you for any invalid data. In other words, it will not ask you to use one of the additional banks if you only have a 64K machine.

If you desire, you may save a configuration file with the spooler installed such that it becomes a permanent part of your system. Each time you load that file afterwards, the spooler will be installed. It is not a good idea to save a configuration file while the spooler is active, though. Allow it to cease printing and empty itself before saving the file. When the spooler is running, if you wish to stop printing use the FORMS command with the EMPTY parameter (see FORMS). This will suspend printing and flush the spool buffer.

When the spooler installs, it will display three items of information on the screen. One, it will indicate where in memory the spool driver itself loaded (i.e. Driver installed at xxxxH). Two, it will tell you what area of memory contains the spool buffer (i.e. Queue placed at xxxxH). Third, it will tell you if it has used either or both banks of memory (i.e. Bank x now in use). Once this display is done, SPOOL is installed and you will be returned to the DOS command level. At that point, the spooler will begin buffering all printer output.

One final note, please be certain to understand that this is an in memory spooler only. You may NOT spool to a disk file. You may use the alternate 64K if it is installed, but you may not spool to disk. To capture printer output on disk, FORCE or ROUTE the printer device to a disk file.

Examples:

```
ASSIGN @PR SPOOL CHRS=10000
ASSIGN @PR SPOOL C=10000
```

This will set up a 10,000 character spool buffer in primary memory.

```
ASSIGN @PR SPOOL BANK1
ASSIGN @PR SPOOL B1
```

This will set up a 32K spool buffer using the first 32K of the alternate 64K. If the additional memory is not there or available, you will get an error.

```
ASSIGN @PR SPOOL BANK1,BANK2
ASSIGN @PR SPOOL B1,B2
```

This command will set up a 64K spool buffer using both banks of the alternate 64K. Remember that 64K is the maximum spool buffer size.

```
ASSIGN @PR SPOOL CHRS=5000,BANK1
ASSIGN @PR SPOOL C=5000,B1
```

This is an invalid command. You cannot use both primary and additional memory for the same spool buffer. Also, since you can only have one spooler at a time, you musy actually choose either primary or additional memory and locate the spooler there.

DVORAK

This is a filter file intended for use with the keyboard, @KI, device.

FILTER [FROM] @KI [TO] DVORAK

@KI is the name of the device being filtered.

DVORAK is the name of the filter file. The extension /FLT is assumed in this case.

The Dvorak Simplified Keyboard is a special arrangement and layout of keys that, in theory at least, greatly enhance typing speed and ease by placing the most often used characters on the home row.

It is possible for you to, by using stickers of some kind, alter the appearance of your TRS-80 QWERTY (so named because of the key order in the upper row) keyboard to that of a Dvorak keyboard. That alone does not give you a Dvorak keyboard.

Even if the key READS one thing, when you press that key, you still get the old value. Therefore, simply altering the keycaps does not a new keyboard make. It must be redefined through software as well.

That is what this Dvorak filter will do. When assigned to the keyboard device, it will actually redefine the TRS-80 keyboard to the Dvorak configuration. At that point, you have a Dvorak keyboard. The actual key values themselves will be altered to their Dvorak counterparts.

Note: There is one small quirk. When using the Dvorak configuration with the standard TRS-80 keyboard, the semi colon (;) key becomes "S". Normally, caps lock on the TRS-80 will cause all alphabetic characters to go to capitals with shift lower case and leave numeric and special characters alone. The TRS-80 caps lock will not affect this new "S" key, because as far as it is concerned, that key is a special character. What that means is that when caps lock is engaged, you will still receive a lower case "s" when pressing that key. Software cannot compensate for this. It is therefore our advice that when using the Dvorak keyboard, use it in the standard typewriter (lower case with shift upper) mode to avoid any confusion.

Example:

```
ASSIGN @KI DVORAK
```

This installs the Dvorak keyboard filter.

```
ASSIGN @DO DVORAK
```

This is invalid. Altering the display does not affect the actual key values. This must be installed for the keyboard device.

EPSON

This is a filter file intended for use with the printer, @PR, device.

FILTER [FROM] @PR [TO] EPSON

@PR is the name of the device being filtered.

EPSON is the name of the filter file. The extension /FLT is assumed here.

The Epson MX-80 series of lineprinters are capable of reproducing the TRS-80's block graphics. There are, however, a couple of oddities that this filter will help smooth out.

The TRS-80 uses the codes 128-191 for its block graphics. The Epson prints these characters normally in the range 140-223. You may bring these codes together in one of two ways.

First of all, you may set your Epson to print the block graphics from 128-191 and have it be compatible with the TRS-80 codes. This is not always the best method, though. When the Epson is using those values for block graphics, the printer control functions normally assigned to the codes 128-139 are lost to the user.

Secondly, and perhaps the best method, is to install this filter on the printer device. This will cause DOSPLUS to offset all the codes being sent to the printer by a value great enough to move it into the range that Epson normally reserves for the TRS-80 graphics. So you get the best of both worlds. TRS-80 graphics without altering the applications software or re-configuring the Epson and losing the control codes.

Examples:

```
ASSIGN @PR EPSON
```

This command will install the Epson filter for the printer device.

```
ASSIGN @DO EPSON
```

This is invalid. You do NOT filter the display device, just the lineprinter. You want the codes to display with their normal values on the screen.

This portion of the DOSPLUS IV manual contains information concerning internal system routines, data storage areas, and diskette formats. This information is useful to the machine-language programmer wishing to write software for use under DOSPLUS IV.

This manual is divided into several sections:

Section	Subject	Page
I.	Rigid drive partitioning	T/2
II.	System files and memory useage	T/5
III.	Directory structure	T/7
	GAT organization	T/7
	HIT organization	T/10
	File entry organization	T/11
IV.	System disk BOOT/SYS sector 2 data	T/14
V.	DCB organization	T/15
	@KI DCB	T/18
	@DO DCB	T/18
	@PR DCB	T/19
	@RS DCB	T/19
VI.	FCB Structure	T/20
VII.	DCT Organization	T/22
VIII.	Supervisor Call System	T/25
	Device I/O Functions	T/32
	File Handler Functions	T/41
	System Control/Info Functions	T/51
	Interrupt Task Functions	T/67
	Base Conversion Functions	T/70
	Arithmetic Functions	T/72
	Command Parsing Functions	T/74
	Disk I/O Functions	T/81
	Miscellaneous Functions	T/89
IX.	Writing drivers and filters for DOSPLUS IV	T/95
	Disk drivers	T/96
	Device drivers	T/97
	Filter programs	T/98
X.	DOSPLUS IV Error codes	T/99
XI.	Keyboard/Video characters and codes	T/101
XII.	Technical glossary	T/103
XIII.	Example programs	
	Disk driver	T/109
	Character I/O driver	T/120
	Filter program	T/131

I. - Rigid Drive Partitioning

With DOSPLUS IV, it is possible to partition, or segregate, a single rigid drive into two or more volumes. Each volume may then be used as if it were a totally independent disk drive. This feature of DOSPLUS IV has several important uses:

1. It makes possible the use of rigid drives with a cylinder count in excess of 200 (the maximum allowable cylinder count under DOSPLUS IV) by providing a means of partitioning the drive into two or more volumes with a cylinder count less than or equal to 200.

2. It allows the user to sub-divide a rather large rigid drive into smaller volumes. Each separate volume may then be assigned a certain function by the user, i.e., one volume for Accounts Receivable programs & data, another volume for payroll information, and yet another for mailing list data.

3. The user may choose to maximize either disk access speed or disk space allocation efficiency by partitioning the drive by cylinder offset or head offset.

Before discussing the effects and advantages of drive partitioning any further, we should understand a little about how partitioning is accomplished. We should begin by describing the basic operation of rigid drives.

A rigid drive consists of one or more flat magnetic disks spinning at high speed. These individual disks are called platters. Each platter has two recording surfaces, one on the top and the other on the bottom. As the platter spins, a read/write head on a movable arm hovers over each surface. The head may be positioned over any of several (usually about 150-400) concentric circular tracks. All like-numbered tracks make up a cylinder. For instance, a drive may contain 3 platters, and therefore 6 surfaces. Each surface may contain 306 concentric tracks. The first track on each surface makes up cylinder #0. The 51st track on each surface, taken together, is cylinder #50, and cylinder #284 is composed of the six tracks numbered 283. At any one moment, all of the read/write heads on the rigid drive are positioned over all of the tracks of some cylinder.

As we have seen, each cylinder is numbered. One way we may partition a drive is by specifying the cylinder number at which each volume begins. For example, if it is desired to partition a 2-platter, 152-cylinder drive into 2 equal-size volumes, we could do this by indicating a cylinder offset using the CONFIG command. The first volume of the rigid drive would begin at cylinder 0, and therefore, the cylinder offset would be 0. The second volume of the drive would begin at cylinder 76, half the total number of cylinders on the drive. Therefore, the cylinder offset for the second volume should be 76. In this way, we have created two 2-platter, 76-cylinder volumes.

Another way to partition a drive is by specifying a head offset. As explained above, rigid drives have a number of read/write heads, each used to record or retrieve data from one surface of the drive. By indicating a head offset, we tell DOSPLUS which surface is the first surface belonging to a volume. To take an example, imagine 3-platter (6-surface), 152-cylinder drive. Let us partition this drive into three equal-size volumes, using the head offset method. Since we have 6 surfaces, we may assign 2 surfaces to each of the three volumes. The first volume would have a head offset of 0, since it will use heads 0 and 1. The second volume, using heads 2 and 3,

will have a head offset of 2, and to the third volume we will assign an offset of 4. We have now created, in effect, three 2-surface, 152 cylinder volumes.

Getting back to the advantages of drive partitioning mentioned above, in the first instance we mentioned that drive partitioning allows DOSPLUS IV to use rigid drives with a cylinder count in excess of 200. DOSPLUS, being a TRSDOS-compatible system, may address up to 200 cylinders per volume (see the explanation of the GAT, section III). If, for example, we attempted to address a 230-cylinder drive as a single volume, only 200 of the cylinders on the drive could actually be accessed by DOSPLUS, thereby wasting 30 cylinders of available storage. With partitioning, however, we may address the drive as two 115-cylinder drives, or one 115-cylinder drive, one 57-cylinder drive, and a 58-cylinder drive, or any other combination desired. In this way, all of the drive's potential capacity is accessible.

The second advantage of drive partitioning, although less technical in nature, is a matter of convenience. Rigid drives afford a great deal of storage - literally millions or tens of millions of bytes. However, it is often annoying to find that all of your files are lumped together on a single rigid drive volume. When using floppy diskettes, most users categorize the diskettes by the type of programs and files contained thereon. Typically, one might have a diskette that contains inventory data and programs, another diskette that holds engineering programs, and a diskette with word processor text files. Partitioning makes it possible for the rigid drive user to set aside individual volumes of a single drive for specific purposes. Take, for example, the case of a 152-cylinder hard drive. By partitioning it into four 38-cylinder volumes, the user may allocate one volume to word processing, another volume to accounting programs and data, a volume to a mailing list database, and still have one left over for general use.

Partitioning also affects how DOSPLUS IV allocates space to its files. When DOSPLUS creates space on a diskette for a file, it allocates that space in units called granules, or grans. The size of a granule is determined, in part, by the way in which a drive is partitioned. When partitioning a drive using the cylinder offset method, all of the like-numbered tracks on the drive comprise a cylinder, and therefore, a cylinder contains a great many sectors. When partitioning by the head offset method, less tracks belong to each cylinder, and less sectors are contained in the cylinder. Generally speaking, the greater the number of sectors per cylinder, the larger DOSPLUS will calculate the granule size to be. Smaller granule sizes result in more efficient use of disk space, but also result in more frequent space allocation when dealing with files of changing size.

The cylinder offset method also maximizes disk access speed by ensuring that the largest cylinder possible is being used (remember that an entire cylinder is accessible at any one time by the read/write heads). The greater the amount of storage immediately available to the hard drive (that is, with no need to re-position the heads), the faster the disk access time. When partitioning by head offset, the size of the cylinder is minimized, and the number of cylinders generally maximized. This results in a great deal of head movement, and slower disk access.

In light of these facts, it can be seen that rigid drive partitioning must always be a compromise between access time and storage efficiency. If many small files are to be stored on the rigid drive, it may be advantageous to minimize cylinder size by using the head offset method of partitioning. If storage efficiency is not so important, or if the files to be stored on the rigid drive are few and large, the cylinder offset

method will afford greater speed. Of course, the two methods may be combined to achieve both a small granule size and good access speed.

II. - System Files

The DOSPLUS IV system is constructed on an overlay scheme. This means that only portions of the complete operating system are resident in the computer's RAM at any given moment. Because of this overlay concept, the operating system may offer many powerful and sophisticated features, since the actual amount of available memory is not an absolutely limiting factor.

DOSPLUS IV is composed of several system files. These system files are divided into three groups:

1. The system executive, SYS0. This is the program that handles the loading and execution of the other system programs, basic I/O to and from the floppy disk drives and other system devices, and provides other functions that must be permanently resident in RAM. SYS0 must be present on all DOSPLUS IV system diskettes for proper operation. SYS0 resides from 0000H to $LLOW-1 (see @FLAGS)

   The low overlay group, SYS1-SYS8. These system modules perform basic system functions such as command evaluation, file and device OPENs and CLOSEs, READs and WRITEs, error messages, and other middle-level functions. The low overlay group occupies the region between 1E00H-23FFH. All low overlay group system files should be present on all DOSPLUS IV system diskettes, or improper operation can result.

3. The high overlay group, SYS9-SYS16. The files contain DOSPLUS's library commands, such as DIR, COPY, DO, etc. The high overlay group is resident in the area 2600H-2FFFH. If desired, the user may delete modules from this group, but only at risk. Once a module is deleted, those library commands contained in that module will cease to function, and if any such command is attempted, the system's proper behavior cannot be guaranteed.

The following table details the functions of each of the overlay programs:

Low overlay group

System File	Function(s)
SYS1	Command interpreter, filespec evaluation routines.
SYS2	@OPEN, @INIT, hash code, trapdoor code generation.
SYS3	@CLOSE, @KILL.
SYS4	Granule system - allocates disk space.
SYS5	Error posting & trap
SYS6	DEBUG monitor package.
SYS7	@EVEL, @WILD.
SYS8	@RAMDIR, @CAT, @DODIR, @SORT.

High overlay group

System File	Function(s)
SYS9	CAT, DIR, FREE.
SYS10	APPEND, COPY, LIST.
SYS11	ASSIGN, FILTER, FORCE, JOIN, RESET.
SYS12	AUTO, BREAK, CLOCK, DATE, DEBUG, ERROR, I, LIB, PAUSE, SCREEN, TIME, VERIFY.
SYS13	DO, FORMS, RS232.
SYS14	ATTRIB, KILL, PROT.
SYS15	BUILD, CLEAR, CREATE, DUMP, LOAD, RENAME.
SYS16	CONFIG, SYSTEM.

System RAM

There exists a block of RAM in low memory which is termed System RAM. This block of RAM is used to hold newly-created DCBs, DCTs, and other data (or programs) which must be resident in system RAM (that is, they must not 'disappear' when @BANK is used to switch in new banks of RAM in the upper 32k of RAM). The user may place data in this area, if desired. Two pointers $LLOW and $LHIGH (see @FLAGS) indicate where this area of RAM currently resides. It is the user's responsibility to adjust these pointers in order to protect whatever program or data has been installed in system RAM.

III. - Directory Structure

Each DOSPLUS IV diskette, whether it is a system disk or a data disk, contains a file called DIR/SYS, the diskette directory. The directory contains information that describes the names, location, length, protection status, and other important attributes of all the files present on the diskette. The directory is composed of three tables:

1. The granule allocation table, or GAT. The GAT contains information concerning free and allocated space on the diskette as well as other assorted data.

2. The hash index table, or HIT. The HIT contains the hash codes for each file in the diskette's directory, and it is used by the system to locate a file in the diskette directory.

3. The file entry table. This table, usually several sectors in length, contains information on individual files, such as the filename, extension, password code, protection level, file length, etc.

GAT Organization

The granule allocation table, or GAT, is located in sector 0 of the file DIR/SYS, and is one sector in length. As the name implies, the GAT contains information about the allocation of granules; that is, the GAT tells us which granules are used and which are not currently allocated. The GAT also contains other information, which will be presented below.

Before launching into an explanation of the allocation table, let us define the term granule. A granule is the smallest unit of storage that the operating system may assign to a file. The actual size of a granule, in sectors, varies with the nature of the diskette being considered. On floppy disks, the following table applies:

Diskette Type	Granule Size (Sectors/Gran)
5" SDEN SSIDE/DSIDE	5
5" DDEN SSIDE/DSIDE	6
8" SDEN SSIDE/DSIDE	8
8" DDEN SSIDE	6
8" DDEN DSIDE	10

(SDEN=Single density, DDEN=Double density,
SSIDE=Single sided, DSIDE=Double sided)

On rigid drives, the granule size is computed by the operating system. The granule size is calculated from the sectors/track and number of sides information set with the CONFIG command to yield the smallest possible granule size (within the dual constraints that the number of sectors/cylinder must be evenly divisible by the granule size, and that no more than 8 granules/cylinder may exist).

Whenever DOSPLUS IV assigns disk space to a file, it does so in terms of granules. For instance, if we were to create a file 1286 bytes long on a 5" DDEN diskette (granule size=6 sectors, from the table above), DOSPLUS would assign an entire granule to the file, 1536 bytes. In this way, files may be expanded without continuously allocating more sectors to the file. It is the purpose of the GAT to maintain a record of which granules have been assigned to files and which granules are free for allocation.

The allocation table provides a map of all granules on the diskette. Starting at byte 00H and continuing through byte 5FH (for floppy configurations), each byte in the table corresponds to an individual cylinder on the diskette. For instance, byte 00H represents cylinder 0, byte 13H corresponds to cylinder 13H (or 19 decimal), and so on. Each bit within the byte is used to indicate which granules within the cylinder are allocated or free. Bit 0 relates the status of granule 0, bit 3 represents granule 3, etc. A reset bit (logic 0) indicates a free granule, and a set bit (logic 1) indicates an allocated, or used granule.

Referring to the sample GAT in figure 1, examine byte 0FH. This byte has the value F9H. Converting this byte into its binary equivalent, we get:

F9H = 1111 1001

Recalling that a set bit indicates allocated, or unavailable granules, and that a reset bit indicates free granules, we can see that cylinder 0FH has two free granules, numbers 1 & 2. All of the other granules on the cylinder are allocated or otherwise unavailable. Non-existent granules are flagged as allocated in this table.

Taking another example, look at byte 10H in the allocation table. This byte has a value of F8H, and converting into binary representation:

F8H = 1111 1000

This time, we can see that cylinder 10H has three free granules, numbers 0, 1, & 2.

On rigid drive configurations, the allocation table is larger, to allow for the greater number of cylinders available on rigid drives. The allocation table then extends from byte 00H through byte C7H. In the case of rigid drives, the granule lock-out table (see below) is not implemented, and locked-out granules are simply treated as allocated granules.

Bytes 60H through BFH are called the lock-out table. This table is similar in structure to the allocation table, but its purpose is somewhat different. During formatting of a floppy diskette, flaws are sometimes found in certain areas of the media. Rather than discard the entire diskette as unusable, DOSPLUS IV will lock out, or render inaccessible, the flawed granule(s). It is the purpose of the lock-out table to map these flawed, locked-out granules. Once again, each byte in the lock-out table corresponds to a cylinder on the diskette, and each bit within each byte of the table represents a single granule on a cylinder. A set bit indicates a locked-out granule, and a reset bit indicates a useable granule. All granules marked as locked-out are also mapped as allocated in the allocation table during format. As is the case with the allocation table, any non-existent grans are mapped as locked-out.

Following the lock-out table, byte CBH contains a DOS version number which may be used to determine what version of the DOSPLUS operating system originally formatted the diskette. This version number is stored as a single byte BCD value. The leftmost 4 bits represent the operating system version number, and the rightmost 4 bits provide the release number. All diskettes formatted under DOSPLUS IV will bear a version code of 60H.

Byte CCH contains a value that indicates the formatted cylinder count of the diskette. Thirty-five is added to this value to obtain the actual cylinder count on the diskette. For example, in figure 1, this byte contains an 05H. Adding 35, we discover that the sample GAT belongs to a 40-track diskette.

Byte CDH, bit 5 is used to indicate single- or double-sided diskettes. Bit 5 set indicates a double-sided diskette. Reset, it means single-sided. The remaining bits are reserved for future use.

The trapdoor code for the diskette master password is stored in bytes CEH and CFH.

The diskette name is located in bytes D0H-D7H, followed by the date field in bytes D8H-DFH. Both fields are left-justified and padded with blanks on the right. In the sample GAT, the diskette name is "DOS IV" and the date is "07/01/83".

The remainder of the GAT sector, bytes E0H-FFH are used to store the AUTO command executed upon boot-up. This can be any ASCII string, terminated with a carriage return, 0DH. In the sample GAT, the AUTO command string has been set to !date. If a carriage return is present in the first byte of the AUTO command field, the diskette effectively has no AUTO command set.

NOTE: It is possible to place an auto command on the GAT of a data disk, either using the DOSPLUS command AUTO, or by means of a user program. However, the AUTO command is only executed from the system disk used to boot the computer. The AUTO command stored on a data diskette will be preserved when and if the diskette is SYSGENed, at which time the AUTO command will become active.

Sample Granule Allocation Table

```
06   00: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFF9 ................
00   10: F8FE FFFF FFFF FFFF FFFF FFFF FFFE FFFF ................
     20: F8F8 F8F8 FCF9 F8F8 FFFF FFFF FFFF FFFF ................
     30: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     40: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     50: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     60: F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 ................
     70: F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 F8F8 ................
     80: F8F8 F8F8 F8F8 F8F8 FFFF FFFF FFFF FFFF ................
     90: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     A0: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     B0: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ................
     C0: FFFF FFFF FFFF FFFF FFFF FF60 0542 9642 ..........  .B.B
     D0: 444F 533A 4956 2020 3037 2F30 362F 3833 DOS:IV  07/01/83
     E0: 2164 6174 650D 2020 2020 2020 2020 2020 !date.
     F0: 2020 2020 2020 2020 2020 2020 2020 2020
```

Figure 1

HIT Organization

The hash index table, or HIT is located in sector 1 of the file DIR/SYS, and is one sector in length. DOSPLUS IV uses the HIT to locate files in the directory. When a file is created, the operating system generates a 1-byte hash code from the filename and extension. This hash code has a value between 01H and FFH (a code of 00H indicates an unused HIT entry). The hash code is then placed into some location in the HIT. The location of the hash code is used to determine the location of the file primary directory entry. Whenever DOSPLUS searches for that file in the future, it first calculates the hash code and searches for a matching hash code in the HIT. When a matching code is found, the system reads the proper file directory entry and, since there may be more than one matching hash code in the HIT, compares the actual filenames. If the names do not match, the system continues to search the HIT for another matching hash code (different filenames may yield identical hash codes). Using this hashing method, the operating system can locate any file entry in the directory very quickly, as opposed to a sequential search of the directory file.

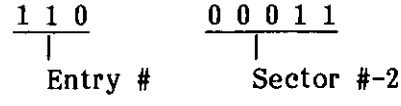
As mentioned above, the position of a hash code entry in the HIT table is used to determine the location of the file's primary directory entry. The position of the hash code in the HIT is referred to as the logical file number, or LFN. For example, let us assume that the hash code generated for the filename TEST/TXT is 30H. Referring to the sample HIT in figure 2, we see that there is an entry at byte C3H that contains a 30H (and in this case, there is only one matching code in the table). Therefore, the logical file number for the file TEST/TXT is C3H.

Sample Hash Index Table

```
06   00: A2C4 2E2F 2C2D 2A2B 2829 2627 27A7 26A6 .../,-*+()&''.&.
01   10: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     20: 25A5 2426 0000 00C5 F2D0 EE92 0A00 55FD %.$&..........U.
     30: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     40: 6580 00E1 0008 F456 964B 00EE 8ED2 5E38 e......V.K....^8
     50: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     60: B668 0000 0000 0000 0000 0000 FC00 FA00 .h..............
     70: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     80: 0100 0000 0000 0000 0000 1600 0000 0000 ................
     90: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     A0: 0F00 0000 0000 0000 0000 0000 0000 0000 ................
     B0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     C0: 0000 0030 F000 0000 0000 0000 0000 0000 ...0............
     D0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     E0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
     F0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Figure 2

The LFN contains two pieces of information: The directory sector number, and the directory entry number. The LFN may be broken down as follows:



That is, bits 0-4 provide the sector number offset (add 2 for the actual sector number) containing the directory entry, and bits 5-7 indicate the proper entry within the sector. In the previous example, the position of the hash code located at C3H

indicates that the file primary directory entry is located on directory sector 05H in entry 6.

File Entry Organization

File directory entries are stored in the file DIR/SYS starting at sector 2 and continuing for the remainder of the file. Each sector contains eight 32-byte directory entries, which begin at relative bytes 00H, 20H, 40H, 60H, 80H, A0H, C0H, and E0H. An entry may be either of two types: File primary directory entries (FPDE), or file extended directory entries (FXDE). The general structure of both types of entries are similar:

Byte	Description (FPDE)	Description (FXDE)
Entry+00H	Flags Bit: 7: 0=FPDE, 1=FXDE 6: 0=User file, 1=System file 5: Reserved 4: 0=KILLed file, 1=Active file 3: 0=Visible file, 1=Invisible file Bits 0-2: Protection level, 0-7	Flags Bit: 7: - Same - 6: - Unused - 5: - Unused - 4: - Same - 3: - Unused - Bits 0-2: - Unused -
Entry+01H	Flags Bit: 7: 0=Shrinkable, 1=Non-shrinkable 6: 0=File unmodified, 1=File modified 5: Reserved 4: Reserved Bits 0-3: Month	Reverse linking LFN pointer
Entry+02H	Date information Bits 3-7: Day Bits 0-2: Year-1980	- Unused -
Entry+03H	End of file byte	- Unused -
Entry+04H	Logical record length (0=256)	- Unused -
Entry+05H	Filename, 8 characters, left-justified	- Unused -
Entry+0DH	Extension, 3 characters, left-justified	- Unused -
Entry+10H	Access password trapdoor code, 2 bytes	- Unused -
Entry+12H	Update password trapdoor code, 2 bytes	- Unused -
Entry+14H	Ending record number, 2 bytes	- Unused -
Entry+16H	Segment descriptor list, 8 bytes	- Same -
Entry+1EH	Seg. desc. terminator/linking LFN, 2 bytes	- Same -

Entry+00H

This byte contains the file's protection level and several flags. Bit 7, when set, flags an entry as a file extended directory entry (FXDE). Extended entries are required when a file a large enough or segmented enough to require more segment descriptors than a single directory entry can provide. FXDE's do not contain any

information in bytes 02-15H, but byte 01H does contain a logical file number which points to the file's previous FPDE or FXDE.

Bit 6 is used to differentiate between system files and user files. Only DOSPLUS IV /SYS modules normally have this bit set. For all other files, this bit should be reset.

Bit 4 is used to indicate whether a directory entry belongs to a currently active file, in which case the bit is set, or if it belongs to a KILLed file, in which case the bit is reset. Under DOSPLUS IV, directory entries are not destroyed when files are killed; they are merely marked as KILLed. This allows the RESTORE utility to reconstruct the file if desired. Note that KILLed directory entries may be re-used by the system as required.

Bit 3 flags the visible/invisible status of the file. When set, the file is invisible; when reset, the file is visible.

Bits 0-2 contain the protection level of the file, which may assume any value from 0-7.

Entry+01H

In a FPDE, this byte contains two flags and a portion of the date of the file's last update. Bit 7 reflects the file's shrinkable/non-shrinkable status. Normally, DOSPLUS IV will allow files to decrease in size, or shrink, if so instructed by a user program. If bit 7 is set, this automatic file shrinkage will be inhibited. When reset, the system will reduce the size of the file if needed.

Bit 6 is used to indicate if a file has been modified, or written to, since the last time this flag was cleared (i.e., during a BACKUP, ATTRIB fs (MOD=N), etc.). When this bit is set, the file has been modified, and when reset, no new data has been written to the file.

Bits 0-3 are used to store the month portion of the date of the last file update. This value will normally fall between 1 and 12, indicating a date from January through December.

As mentioned above, this byte is also used in FXDE's as a reverse linking LFN. That is, this byte will hold the logical file number of the directory entry that linked into the FXDE.

Entry+02H

This byte contains the balance of the file's date information. Bits 3-7 contain the day of the month, and bits 0-2 contain the year. All year information is based on a starting year of 1980. Therefore, if this byte contained the value 7BH, this would represent the 15th day of the month (the month is stored in REC+01H) in 1983.

Entry+03H

The value of this byte indicates how many bytes the file extends into its final record. For instance, a file of logical record length 37 with 65 records is 2405 bytes long. Such a file would completely fill 9 256-byte sectors, and a portion of a tenth sector. This byte is used to tell us how much of that final sector is used. In this example, the file would extend 101 bytes into the last sector, and therefore the

end-of-file (EOF) byte would be 64H (100 decimal). An EOF byte of 0 means that the entire sector is filled.

Entry+04H

This byte contains the logical record length, or LRL, with which the file was originally created. A value of zero indicates a LRL of 256.

Entry+05H

The 8-character filename is stored in the bytes entry+05H through entry+0CH. The filename is left-justified and padded with blanks on the right.

Entry+0DH

The 3-character extension is contained in entry+0DH through entry+0FH. The extension is left-justified and padded with blanks on the right.

Entry+10H

This two-byte value contains the trapdoor code for the file's update password.

Entry+12H

This two-byte value contains the trapdoor code for the file's access password.

Entry+14H

These two bytes contain the total number of sectors occupied by the file. Both complete and partial sectors are included in this count.

Entry+16H

The bytes from entry+16H through entry+1DH make up the segment descriptor list. The segment descriptor list is a set of four 2-byte values that describe the location of the various portions, or segments, of the file. The first byte of each set of two contains the cylinder number at which the segment begins. The second byte contains two pieces of information: Bits 5-7 indicate the granule number within the cylinder at which the segment begins, and bits 0-4 contain the number of contiguous granules that are in the segment. This value is offset by 1; that is, a count of 0 means that the segment contains 1 granule, a count of 27 means that the segment contains 28 granules, etc.

If the first byte of any of the four segment descriptors is FFH, the file has no more segments. If none of the four descriptors contains an FFH, then the two bytes at entry+1EH and entry+1FH must contain either the FFH termination code or a link to a FXDE.

Entry+1EH

The purpose of these two final bytes of the directory entry is to (a) signal the end of the segment descriptor list, or (b) provide a pointer to a file extended directory entry which contains another segment descriptor list. If entry+1EH contains an FFH, then there are no more segment descriptors to follow. If the value is an FEH, the

following byte contains a logical file number which points to a FXDE containing more segment descriptors.

IV. - BOOT/SYS Organization

All DOSPLUS IV diskettes, both system and data disks, contain a file named BOOT/SYS. This system file occupies the first few sectors on each diskette. On data diskettes, the BOOT/SYS file contains information about the location of the diskette's directory. System diskettes and rigid drives contain more data in the BOOT/SYS file. Before describing the contents of the file, let us explore the purpose of BOOT/SYS.

In the case of system disks, the first and foremost responsibility of the BOOT/SYS file, or bootstrap, is to provide a small program which is used to load the DOSPLUS IV system file SYS0/SYS. When the computer is booted, a routine in the Model 4's ROM reads sector 1 from cylinder 0 of the system diskette into RAM at 4300H. This sector should contain a Z-80 object code program, not to exceed 256 bytes in length, saved in core image format. Since data diskettes are not needed to boot the computer, the bootstrap program is not present in the BOOT/SYS file of such diskettes.

After loading the 256-byte sector from BOOT/SYS, the ROM will transfer control to the program loaded at 4300H. This 256-byte bootstrap program must now perform several functions:

(a) It must read sector 2 of the bootstrap file into RAM, to obtain the important perishable DCT information on the system disk. This information is of great consequence when reading subsequent information from the diskette.

(b) It must read any alternate system driver program from sectors 3 - x into RAM for execution after SYS0 is loaded and initialized (optional).

(c) It must locate and load the SYS0/SYS file, and transfer program control to it.

The actual bootstrap program is only part of the information contained in the BOOT/SYS file. Sector 0, byte 02H always contains a value that indicates the cylinder which contains the diskette's directory. For instance, if a diskette's directory were located on cylinder 20, this byte would have a value of 14H (20 decimal). On previous versions of DOSPLUS, bit 7 of this byte would be set to indicate a double-density diskette, and reset to indicate single-density. This convention is no longer followed, but compatibility with older disks may be maintained by ignoring the status of bit 7.

Sector 2 of the bootstrap file contains a great deal of information. Beginning at byte 00H and continuing through byte 08H is a duplicate of the perishable portion of the drive control table (DCT) for the diskette. See section VII of this manual for details on the DCT. Floppy data diskettes do not contain a DCT in the BOOT/SYS file.

Sector 2, byte 10H contains several flags. Bits 4-7 are reserved for future use. Bit 3 controls whether or not DOSPLUS IV will prompt the operator to input the current time upon boot-up. Set, this bit instructs the DOS to prompt for the time; reset, the time question is skipped. Bit 2 performs a similar function for the date prompt. If this bit is set, the operator will be asked to enter the date; if reset, the question will not be asked. Bit 1 determines whether DOSPLUS IV will display the graphic DOSPLUS

"billboard" logo on the CRT when booted. If set, the logo will be displayed, if reset, the display will be suppressed.

Bit 0 of byte 10H is used to flag the presence of an alternate system driver. If set, this means that the bootstrap program loads a special disk device driver program into RAM during the bootup process. The system uses this bit in conjunction with the word stored in bytes 11H and 12H. When bit 0 of byte 10H is set, the system will transfer control to the address specified by the two bytes stored in 11H and 12H. If bit 0 is not set, bytes 11H and 12H are ignored.

Byte 13H controls the blink/no blink status of the cursor. If this is a non-zero value, the cursor will blink on and off. The cursor will remain steadily on if this byte contains a zero.

Byte 14H contains the value of the character used as a cursor.

Byte 15H is holds the caps status to be used after bootup. That is, it determines whether alphabetic keys will produce upper- or lower-case letters in the unshifted mode. If this byte contains a non-zero value, unshifted keys will produce upper-case letters. If the byte is a zero, lower-case will result.

Byte 16H is the default step rate code for all floppy drives. Unless a drive has been re-CONFIGured with a new step rate code or /CFG file, this byte will determine the rate at which the drive's head stepper motor is operated.

Byte 17H and 18H are used to output a user-specified value to a user-specified port upon bootup. Byte 17H contains the port address, and byte 18H contain the value to be output to the port.

V. - DCB Organization

Device Control Blocks (DCBs)

A device control block, or DCB, is an area of RAM that contains certain data that is used to control the flow of data to and from character-oriented devices. Eight devices are available under DOSPLUS IV: @KI (the keyboard), @DO (the video display), @PR (the printer), @RS (the RS232 serial interface), two user-defined devices, @U1 & @U2, and the standard input/output devices, @SI and @SO.

DCBs may be of varying length, but they do share a common structure, which is diagrammed below:

Address	Data
DCB+00H	DCB type flags Bit 7: File Control Block. 0=Device, 1=File Bit 6: FILTER flag. 0=No FILTER, 1=FILTER Bit 5: JOIN flag. 0=No JOIN, 1=JOINed Bit 4: FORCE flag. 0=No FORCE, 1=FORCEd Bit 3: NIL flag. 0=Active, 1=NIL Bit 2: Ctl type. 0=No control I/O, 1=Control I/O Bit 1: Output type. 0=No output, 1=Output Bit 0: Input type. 0=No input, 1=Input
DCB+01H	2-byte driver address
DCB+03H	2-byte FORCE/JOIN DCB address
DCB+05H	2-byte filter table address

The first byte of all DCBs contains 8 flags that describe the current status of the device. Bit 7, when set, indicates that the control block belongs to a file rather then a device. That is, if bit 7 of the DCB type is set, an FCB, or file control block, follows.

Bit 6 is used to inform the device driver that a filter table is installed for the device and that it is active. DOSPLUS IV's character I/O system automatically performs any required character translation, and therefore the device driver need not concern itself with that responsibility.

Bit 5 is set whenever the device is currently JOINed to another device or a file.

Bit 4 is used to flag an active FORCE to another device or a file.

Bit 3 is used to indicate that a device is currently NIL. A NIL device does not output characters, nor does it pass any input characters.

Bits 0 through 2 are used to indicate the type of I/O that a driver is capable of performing. Bit 2, when set, means that the driver is capable of accepting or providing control data. By control, we mean any data transfer operation that does not fall into the classifications of input or output. Control data transfers are typically used to set or query the status of devices.

Bit 1 flags a driver as capable of accepting data for output to a device. A printer driver, for example, would have this bit set, meaning that the driver accepts data for output to the printer.

Bit 0 indicates that a driver can provide data that is input from a device. A keyboard driver would be an example of such a device; it receives data from a device and passes it back to the requesting program.

DCB+01H and DCB+02H contain the address of the device driver program. In order to pass data to a device, or accept data from a device, a program points a Z-80 register pair to the address of the proper DCB. A call is then made to a supervisor call named character input/output, or CHNIO. This routine prepares for input or output from or to the device and then transfers control to the driver address contained in the DCB.

DCB+03H and DCB+04H contain the address of another DCB or an FCB if the device has been FORCEd or JOINed to another device or file.

DCB+05H and DCB+06H contain the address of a character translation table used to alter the values of characters passing between the driver and external programs. The first byte of the table is a value which indicates the number of 2-byte entries the table holds. The following bytes are arranged in 2-byte pairs; the first byte of which represents the character value which is to be altered, and the second byte of which represents the value into which the character will be translated.

DOSPLUS IV supports eight character-oriented devices. Two of these devices, @U1 & @U2 are user-defined and have no pre-defined DCB area within the system. @KI, @DO, @PR, & @RS all have DCB locations reserved in system RAM. The address of the various DCBs may be ascertained through the use of the @LOCDCB or @GTDCB supervisor calls. If desired, the user may relocate and redefine the DCBs. The information presented below refers to the meaning of the information within them.

Device name: @KI

Byte	Meaning
DCB+00H	DCB type (1 byte)
DCB+01H	Driver address (2 bytes)
DCB+03H	FORCE/JOIN DCB address (2 bytes)
DCB+05H	Filter table address (2 bytes)
DCB+07H	Internal use (1 byte)
DCB+08H	Caps flag. 00H=No L/C, 20H=U/C (1 byte)
DCB+09H	Flags Bit: 3 - Disable [BREAK] key 2 - Enable keyclick 1 - Suppress debounce 0 - Suppress graphics screen print
DCB+0AH	Internal use (2 bytes)
DCB+0CH	Delay before key auto-repeat (2 bytes)
DCB+0EH	Auto-repeat rate (2 bytes)

Device name: @DO

Byte	Meaning
DCB+00H	DCB type (1 byte)
DCB+01H	Driver address (2 bytes)
DCB+03H	FORCE/JOIN DCB address (2 bytes)
DCB+05H	Filter table address (2 bytes)
DCB+07H	Flags Bit: 7 - Display control character on CRT 6 - Display special characters
DCB+08H	Cursor status. 00H=Off, else=Character under cursor (1 byte)
DCB+09H	Cursor character (1 byte)
DCB+0AH	Video mode 00H=Normal, 80H=Inverse video (1 byte)
DCB+0BH	Cursor address (2 bytes)
DCB+0DH	1st line video window address (2 bytes)
DCB+0FH	2nd line video window address (2 bytes)
DCB+11H	# of charactes in video window - 80 (2 bytes)

Device name: @PR

Byte	Meaning
DCB+00H	DCB type (1 byte)
DCB+01H	Driver address (2 bytes)
DCB+03H	FORCE/JOIN DCB address (2 bytes)
DCB+05H	Filter table address (2 bytes)
DCB+07H	Horizontal tab table address (2 bytes)
DCB+09H	Character output vector (2 bytes)
DCB+0BH	Flags Bit: 7 - Ignore null lines 6 - Translate FF, VT, & HT to LF's & SPACEs
DCB+0CH	Page length (1 byte)
DCB+0DH	Lines printed/page (1 byte)
DCB+0EH	Maximum width (1 byte)
DCB+0FH	Indent count(1 byte)
DCB+10H	Character count (1 byte)
DCB+11H	Line count (1 byte)

Device name: @RS

Byte	Meaning
DCB+00H	DCB type (1 byte)
DCB+01H	Driver address (2 bytes)
DCB+03H	FORCE/JOIN DCB address (2 bytes)
DCB+05H	Filter table address (2 bytes)
DCB+07H	Reserved
DCB+08H	Baud rate code (1 byte)
DCB+09H	UART configuration code (1 byte)

VI. - FCB Structure

One of the most important functions of a disk operating system is the manipulation of disk data files. Under DOSPLUS IV, data files are accessed via the various system file handling routines, described in section VIII of this manual. Each time a system routine performs a function on a file, it references that file through a portion of RAM known as a file control block, or FCB. In this respect, the FCB is much like a DCB; it is used to control the flow of data to or from the file. In fact, DCBs and FCBs can be interchanged for character I/O functions (see @GET and @PUT, section VIII).

All file control blocks have a common structure. Before a file is OPENed or INITed for I/O, the DCB is a 32-byte area of RAM that should contain the filename, extension (if any), password (if any), and drivespec (if needed) of the file to be referenced. This filespec should be terminated with a carriage return (0DH) or an ETX character (03H). After an OPEN or INIT, the FCB contains the following information:

Byte	Meaning
FCB+00H	FCB Type, 80H (1 byte)
FCB+01H	Flags (1 byte) Bit 7: Blocked records Bit 6: Random access Bit 5: Buffer=NRN Bit 4: Buffer updated Bit 3: Reserved Bits 0-2: Access code
FCB+02H	Flags (1 byte) Bit 7: Non-shrinkable file Bit 6: Modify flag Bits 0-5: Reserved
FCB+03H	File I/O buffer address (2 bytes)
FCB+05H	Next record number offset (1 byte)
FCB+06H	Device # (1 byte)
FCB+07H	Logical file number (1 byte)
FCB+08H	End of file byte (1 byte)
FCB+09H	Logical record length (1 byte)
FCB+0AH	Next record number (2 bytes)
FCB+0CH	Ending record number (2 bytes)
FCB+0EH	Segment descriptor list (17 bytes)

The byte located at FCB+00H is called the FCB type byte, and it is used to distinguish an ECB from a DCB. In the case of an FCB, this byte will always have the value 80H. DCB type bytes (located at DCB+00H) may never assume this value.

FCB+01H contains several flags reflecting the status of the file. Bit 7 indicates that the file is made up of blocked records. A blocked record is a logical record which shares a physical record with one or more other logical records. When bit 7 is set, it means that the DOS is automatically performing record blocking (placing multiple logical records into a single physical record) and unblocking (retrieving individual logical records from a physical record). User programs sometimes reset this bit to force the operating system to write an entire physical record to diskette when working with logical records of less than 256 bytes in length.

Bit 6 flags the mode in which a file is accessed. When a file is first OPENed or INITed, this flag is reset, indicating sequential access. The flag is set when a random access operation is performed on the file (the @POSN system routine). If a file is written to and this flag reset, the operating system will shrink the length of the file to reflect the last byte written.

If the file I/O buffer specified at time of OPEN or INIT contains the next logical record in the file, bit 5 will be set. If the buffer does not contain the next record, the bit will be reset to indicate that the operating system must perform a READ to access that record.

Bit 4 is used to indicate that the contents of the file I/O buffer have been updated or modified since the last READ or WRITE from or to the file.

Bits 0-2 contain the access level code under which the file was OPENed. For instance, imagine a file called TEST/CMD that has an update password of "PW" and an access password of "TEST", and the protection level is set to 5. When this file is OPENed as "TEST/CMD.PW", the access level code in the FCB will contain the value 0, to indicate total access. If the file is opened with the filespec "TEST/CMD.TEST", the access level code will contain a 5, reflecting the protection level authorized by the access password.

FCB+01H contains two more important flags. Bit 7 is used to inform the operating system that it should not attempt to shrink the file at CLOSE time if the length of the file has decreased.

Bit 6 is set whenever the file has been written to, or modified. If no data has been output to the file, this bit will remain reset.

FCB+03H and FCB+04H point to a 256-byte file I/O buffer which is specified at the time of OPEN or INIT. All data transferred between the computer and the file must pass through this buffer.

FCB+05H is a single-byte value that contains an offset to the beginning of the next logical record within the current physical record. If the beginning of the next logical record lies at the start of the next physical record, this byte will have a value of 0.

FCB+06H contains the drive device number upon which the file is resident.

FCB+07H is a one byte value which reflects the logical file number, or LFN, of the file's primary directory entry.

FCB+08H contains the end of file byte, or EOF. This byte tells the operating system how far the file extends into the final sector of the file. A value of 0 means that the entire sector is occupied by the file.

FCB+09H is the logical record length of the file, as determined by OPEN or INIT. Files with an LRL of 256 will contain a 0 in this byte.

FCB+0AH and FCB+0BH contain the next record number, or NRN. This is simply the number of the next physical record in the file.

FCB+0CH and FCB+0DH contain the total number of complete sectors in the file. Please note that partial sectors are not included in this count.

FCB+0EH through FCB+1FH contain a segment descriptor list used by the operating system to retrieve portions of the file from diskette.

VII. - DCT Organization

DOSPLUS IV maintains sixteen drive control tables, or DCTs, in which all of the information pertaining to each disk drive device in the system is stored. Since these DCTs may be located at any location in RAM, DOSPLUS provides two SVCs named @GTDCT and @LOCDCT which can find the DCT for any given logical drive number.

The DCT itself is a 20-byte long region of RAM which contains all of the information necessary for the operating system and its associated disk drive device drivers to operate the drive. The following information details the structure of the DCTs used by DOSPLUS IV's drivers. The individual user may create a DCT using any structure desired, assuming drivers are written to interface to such DCTs, but we encourage the use of this standard DCT arrangement, as the CONFIG library command assumes the following structure is used:

Non-perishable DCT information

Byte	Meaning
DCT+00H	DCT type (1 byte)
DCT+01H	Driver address (2 bytes)
DCT+03H	Flags (1 byte) Bit 7: 5"/8" switch. 0=5", 1=8" Bit 6: Write protect. 0=No prot., 1=Protected Bit 5: Floppy/rigid switch. 0=Floppy, 1=Rigid Bit 4: Motor delay switch. 0=No delay, 1=Delay Bit 3: Head load delay switch. 0=No delay, 1=Delay Bit 2: Skip switch. 0=No skip, 1=Skip Bit 1: Fixed/removable switch. 0=Fixed, 1=Removable Bit 0: Log disk switch. 0=No log, 1=Log disk
DCT+04H	Step rate code (1 byte)
DCT+05H	Head offset (1 byte)
DCT+06H	Cylinder offset (2 bytes)
DCT+08H	Sector offset (1 byte)
DCT+09H	Head location (1 byte)
DCT+0AH	Physical drive number (1 byte)

Perishable DCT information

Byte	Meaning
DCT+0BH	Flags (1 byte) Bit 7: Single/double density switch. 0=Single, 1=Double Bit 6: Directory protect switch. 0=No prot., 1=Protected Bit 0-5: Reserved
DCT+0CH	Surface count (1 byte)
DCT+0DH	Sectors/track (1 byte)
DCT+0EH	Directory length (1 byte)
DCT+0FH	Sectors/gran (1 byte)
DCT+10H	Grans/cylinder (1 byte)
DCT+11H	Sectors/cylinder (1 byte)
DCT+12H	Directory location (1 byte)
DCT+13H	Cylinder count (1 byte)

As can be seen from the table, the DCT is divided into two parts, termed the Non-perishable and the perishable data. Non-perishable data includes the address of the device driver and most of the physical characteristics of the drive. Perishable data includes that information which is peculiar to the actual diskette, such as density, directory location, surface count, etc. Note that perishable data is subject to automatic change after log-in of a disk drive.

Non-perishable data

DCT+00H is called the DCB type byte. This is analogous to the DCB and FCB type byte previously discussed, and is used to identify the DCT entry and distinguish it from an FCB or DCB. The normal value of this byte, for an active drive, is 40H. If the drive device is NIL, bit 3 will also be set, yielding a value of 48H.

DCT+01H and DCT+02H contain the address of the disk drive device driver program.

DCT+03H contains eight flags that describe the status of the drive device. Bit 7 is used to flag whether the drive is a 5 inch or and 8 inch drive. Reset, 5 inch is indicated, and set, 8 inch.

Bit 6 is used as a software write protect switch. When this bit is set, it means that the user has set the WP parameter for this drive using the CONFIG command. It is the responsibility of the driver to respond to this bit.

Bit 5 is used to indicate whether the DCT describes a floppy drive or a rigid drive. When reset, a floppy drive is described, and when set, a rigid drive description is contained in the DCT.

Bit 4 is used to inform the driver whether a drive requires a delay after the drive motor is started. When set, the driver should provide a delay.

Bit 3 indicates to the disk device driver whether or not a delay is required for read/write head loading. When set, the driver should provide a delay.

When Bit 2 is set, it instructs the device driver to step the read/write head twice as far as normal when seeking any specified cylinder. This allows DOSPLUS IV to read and write 40-cylinder floppy diskettes in 80-cylinder drives.

Bit 1 is used to inform the device driver whether a rigid disk is of the fixed or removable type. When set, this bit indicates a removable type.

Bit 0 instructs the driver to log the diskette during the next access to the drive. Logging the diskette means that the driver will re-read information (such as side count) from the diskette during the next read or write operation on the drive.

DCT+04H contains the step rate code for the drive.

DCT+05H is a one-byte value that contains the head offset for partitioned volumes as specified under the CONFIG library command.

DCT+06H and DCT+07H contain the two-byte cylinder offset for partitioned volumes.

DCT+08H contains a one-byte sector offset. The sector offset is simply the beginning sector number on a diskette track.

DCT+09H contains the current cylinder number over which the drive's read/write head(s) are located.

DCT+0AH is the physical drive number of the disk drive device.

Perishable data

DCT+0BH contains flags relating to the nature of the diskette in the disk drive. Bit 7 of this byte indicates the recording density of the diskette. Set, this bit flags double-density media, and reset it means that single-density media is in use.

Bit 6 is used to indicate whether the diskette possesses a protected directory. A protected directory is a directory which is recorded with a special data address mark. Normally, floppy diskette directories are protected, but rigid drives often do not have facilities to create special address marks. Therefore, this bit informs the driver what type of directory to expect when reading a diskette.

DCT+0CH is a one-byte value that contains the number of surfaces present on a diskette.

DCT+0DH contains the number of sectors on each track of the diskette.

DCT+0EH is a one-byte value that contains the length of the diskette directory.

DCT+0FH contains the granule size that applies to the drive, in sectors/granule.

DCT+10H is the number of granules present per cylinder.

DCT+11H contains the total number of sectors per cylinder

DCT+12H is a one-byte value that contains an offset from the beginning cylinder of the volume (DCT+06H and DCT+07H) to the directory cylinder.

DCT+13H contains the total number of cylinders for the diskette or volume.

XIII. - Supervisor Call System

DOSPLUS IV provides a set of useful built-in routines termed Supervisor Calls, or SVCs which may by accessed by the user to perform common functions such as obtaining a keystroke from the console, displaying data on the CRT, reading and writing data to/from disk files, etc. This section of the manual details these Supervisor Calls and their use.

The Supervisor Call system of DOSPLUS IV is largely compatible with that of TRSDOS 6.0. Some of TRSDOS's less sensible SVCs have been omitted, while several powerful SVCs have been added to DOSPLUS IV.

Users of the DOSPLUS operating system for the TRS-80 Model I or III may find SVCs to be a new concept. On these earlier machines, DOS routines were accessed by performing a CALL to a specific location in the operating system's RAM. Under DOSPLUS IV, both the operating system and the user are freed from concerning themselves with fixed RAM addresses. Instead, each user-accessible DOS routine is assigned a number, called an SVC number. To call the function, the SVC number is loaded into the Z-80 microprocessor's accumulator and a RST 28H instructing is executed. For example, under DOSPLUS 3.5 (a TRS-80 Model I/III system), these procedure to OPEN a file for I/O would be as follows:

```
LD    B,0         ;LRL=256
LD    DE,FCB      ;DE=>FILE CONTROL BLOCK
LD    HL,BUFF     ;HL=>FILE BUFFER
CALL  4424H       ;CALL OPEN ROUTINE
```

Under DOSPLUS IV, the equivalent procedure is:

```
LD    B,0         ;LRL=256
LD    DE,FCB      ;DE=>FILE CONTROL BLOCK
LD    HL,BUFF     ;HL=>FILE BUFFER
LD    A,59        ;OPEN SVC CODE
RST   28H         ;OPEN FILE
```

In general, the steps involved in executing a Supervisor Call are:

1. Set up the Z-80 registers and/or RAM locations as required for the desired SVC.

2. Load the accumulator, or A register, with the SVC code.

3. Execute a RST 28H to invoke the SVC.

Throughout this manual, we will observe the following symbolic conventions:

Symbol	Meaning
n = xx	8-bit register **n** contains the value **x.**
nn = xxxx	16-bit register pair **nn** contains the value **xxxx.**
nn =>xxxx	16-bit register pair **nn** contains a pointer to the address **xxxx.**
ZF	The Z-80 Zero flag is set.
NZ	The Z-80 Zero flag is reset.
CF	The Z-80 Carry flag is set (also refer to register C).
NC	The Z-80 Carry flag is reset.

DOSPLUS IV Supervisor Calls - Table of Contents
(By Category)

I. Device I/O Functions

A. Basic byte I/O functions

1.	@GET	(3,	03H)	T/33
2.	@PUT	(4,	04H)	T/34
3.	@CTL	(5,	05H)	T/35
4.	@CHNIO	(20,	14H)	T/36
5.	@MSG	(13,	0DH)	T/36

B. Keyboard functions

1.	@KEY	(1,	01H)	T/37
2.	@KBD	(8,	08H)	T/37
3.	@KEYIN	(9,	09H)	T/38

C. Display functions

1.	@DSP	(2,	02H)	T/38
2.	@DSPLY	(10,	0AH)	T/39

D. Printer functions

1.	@PRT	(6,	06H)	T/39
2.	@PRINT	(14,	0EH)	T/40

II. File Handler Functions

A. File control

1.	@RENAM	(56,	38H)	T/41
2.	@REMOV	(57,	39H)	T/42
3.	@INIT	(58,	3AH)	T/42
4.	@OPEN	(59,	3BH)	T/43
5.	@CLOSE	(60,	3CH)	T/44
6.	@FEXT	(79,	4FH)	T/44
7.	@FNAME	(80,	50H)	T/45

B. File positioning functions

1.	@BKSP	(61,	3DH)	T/45
2.	@CKEOF	(62,	3EH)	T/45
3.	@LOC	(63,	3FH)	T/46
4.	@LOF	(64,	40H)	T/47
5.	@PEOF	(65,	41H)	T/47
6.	@POSN	(66,	42H)	T/48
7.	@REW	(68,	44H)	T/48
8.	@SKIP	(72,	48H)	T/48
9.	@WEOF	(74,	4AH)	T/49

C. File I/O functions
1. @READ (67, 43H) T/49
2. @VER (73, 49H) T/49
3. @WRITE (75, 4BH) T/50

III. System Control & Information Functions

A. System control functions
1. @IPL (0, 00H) T/51
2. @ABORT (21, 15H) T/51
3. @EXIT (22, 16H) T/52
4. @CMNDI (24, 18H) T/52
5. @CMNDR (25, 19H) T/52
6. @ERROR (26, 1AH) T/53
7. @DEBUG (27, 1BH) T/53
8. @DODIR (34, 22H) T/54
9. @RAMDIR (35, 23H) T/55
10. @LOAD (76, 4CH) T/56
11. @RUN (77, 4DH) T/57
12. @BANK (102, 66H) T/58
13. @BREAK (103, 67H) T/60

B. System information functions
1. @DATE (18, 12H) T/60
2. @TIME (19, 13H) T/61
3. @DCSTAT (40, 28H) T/61
4. @GTDCT (81, 51H) T/61
5. @GTDCB (82, 52H) T/62
6. @HIGH$ (100, 64H) T/62
7. @FLAGS (101, 65H) T/63
8. @LOCDCT (122, 7AH) T/65
9. @LOCDCB (123, 7BH) T/66
10. @LOCDEV (126, 7EH) T/66

IV. Interrupt Task Functions

1. @CKTSK (28, 1CH) T/67
2. @ADTSK (29, 1DH) T/68
3. @RMTSK (30, 1EH) T/69
4. @RPTSK (31, 1FH) T/69
5. @KLTSK (32, 20H) T/69

V. Base Conversion Functions

1. @DECBIN (96, 60H) T/70
2. @BINDEC (97, 61H) T/70
3. @HEX8 (98, 62H) T/71
4. @HEX16 (99, 63H) T/71

VI. Arithmetic Functions

1.	@MUL8	(90, 5AH)	T/72
2.	@MUL16	(91, 5BH)	T/72
3.	@DIVD8	(93, 5DH)	T/73
4.	@DIVD16	(94, 5EH)	T/73

VII. Command Parsing Functions

1.	@PARAM	(17, 11H)	T/74
2.	@FSPEC	(78, 4EH)	T/78
3.	@EVAL	(124, 7CH)	T/78

VIII. Disk I/O Functions

1.	@CKDRV	(33, 21H)	T/81
2.	@SEEK	(46, 2EH)	T/81
3.	@RDHDR	(48, 30H)	T/82
4.	@RDSEC	(49, 31H)	T/82
5.	@VRSEC	(50, 32H)	T/83
6.	@RDTRK	(51, 33H)	T/83
7.	@WRSEC	(53, 35H)	T/83
8.	@WRSSC	(54, 36H)	T/84
9.	@WRTRK	(55, 37H)	T/84
10.	@RDSSC	(85, 55H)	T/85
11.	@DIRRD	(87, 57H)	T/85
12.	@DIRWR	(88, 58H)	T/86
13.	@DISKIO	(121, 79H)	T/86

IX. Miscellaneous Functions

1.	@WHERE	(7, 07H)	T/89
2.	@LOGER	(11, 0BH)	T/89
3.	@LOGOT	(12, 0CH)	T/89
4.	@VDCTL	(15, 0FH)	T/90
5.	@PAUSE	(16, 10H)	T/92
6.	@SOUND	(104, 68H)	T/92
7.	@WILD	(125, 7DH)	T/94
8.	@SORT	(127, 7FH)	T/94

DOSPLUS IV Supervisor Calls - Table of Contents
(Alphabetical order)

SVC Name	Code (Dec)	Code (Hex)	Page #	SVC Name	Code (Dec)	Code (Hex)	Page #
@ABORT	21	15H	T/51	@LOC	63	3FH	T/46
@ADTSK	29	1DH	T/68	@LOCDCB	123	7BH	T/66
@BANK	102	66H	T/58	@LOCDCT	122	7AH	T/65
@BINDEC	97	61H	T/70	@LOCDEV	126	7EH	T/66
@BKSP	61	3DH	T/45	@LOF	64	40H	T/47
@BREAK	103	67H	T/60	@LOGER	11	0BH	T/89
@CHNIO	20	14H	T/36	@LOGOT	12	0CH	T/89
@CKDRV	33	21H	T/81	@MSG	13	0DH	T/36
@CKEOF	62	3EH	T/45	@MUL16	91	5BH	T/72
@CKTSK	28	1CH	T/67	@MUL8	90	5AH	T/72
@CLOSE	60	3CH	T/44	@OPEN	59	3BH	T/43
@CMNDI	24	18H	T/52	@PARAM	17	11H	T/74
@CMNDR	25	19H	T/52	@PAUSE	16	10H	T/92
@CTL	5	05H	T/35	@PEOF	65	41H	T/47
@DATE	18	12H	T/60	@POSN	66	42H	T/48
@DCSTAT	40	28H	T/61	@PRINT	14	0EH	T/40
@DEBUG	27	1BH	T/53	@PRT	6	06H	T/39
@DECBIN	96	60H	T/70	@PUT	4	04H	T/34
@DIRRD	87	57H	T/85	@RAMDIR	35	23H	T/55
@DIRWR	88	58H	T/86	@RDHDR	48	30H	T/82
@DISKIO	121	79H	T/86	@RDSEC	49	31H	T/82
@DIVD16	94	5EH	T/73	@RDSSC	85	55H	T/85
@DIVD8	93	5DH	T/73	@RDTRK	51	33H	T/83
@DODIR	34	22H	T/54	@READ	67	43H	T/49
@DSP	2	02H	T/38	@REMOV	57	39H	T/42
@DSPLY	10	0AH	T/39	@RENAM	56	38H	T/41
@ERROR	26	1AH	T/53	@REW	68	44H	T/48
@EVAL	124	7CH	T/78	@RMTSK	30	1EH	T/69
@EXIT	22	16H	T/52	@RPTSK	31	1FH	T/69
@FEXT	79	4FH	T/44	@RUN	77	4DH	T/57
@FLAGS	101	65H	T/63	@SEEK	46	2EH	T/81
@FNAME	80	50H	T/45	@SKIP	72	48H	T/48
@FSPEC	78	4EH	T/78	@SORT	127	7FH	T/94
@GET	3	03H	T/33	@SOUND	104	68H	T/92
@GTDCB	82	52H	T/62	@TIME	19	13H	T/61
@GTDCT	81	51H	T/61	@VDCTL	15	0FH	T/90
@HEX16	99	63H	T/71	@VER	73	49H	T/49
@HEX8	98	62H	T/71	@VRSEC	50	32H	T/83
@HIGH$	100	64H	T/62	@WEOF	74	4AH	T/49
@INIT	58	3AH	T/42	@WHERE	7	07H	T/89
@IPL	0	00H	T/51	@WILD	125	7DH	T/93
@KBD	8	08H	T/37	@WRITE	75	4BH	T/50
@KEY	1	01H	T/37	@WRSEC	53	35H	T/83
@KEYIN	9	09H	T/38	@WRSSC	54	36H	T/84
@KLTSK	32	20H	T/69	@WRTRK	55	37H	T/84
@LOAD	76	4CH	T/56				

DOSPLUS IV Supervisor Calls - Table of Contents (Numerical order)

SVC Name	Code (Dec)	Code (Hex)	Page #	SVC Name	Code (Dec)	Code (Hex)	Page #
@IPL	0	00H	T/51	@INIT	58	3AH	T/42
@KEY	1	01H	T/37	@OPEN	59	3BH	T/43
@DSP	2	02H	T/38	@CLOSE	60	3CH	T/44
@GET	3	03H	T/33	@BKSP	61	3DH	T/45
@PUT	4	04H	T/34	@CKEOF	62	3EH	T/45
@CTL	5	05H	T/35	@LOC	63	3FH	T/46
@PRT	6	06H	T/39	@LOF	64	40H	T/47
@WHERE	7	07H	T/89	@PEOF	65	41H	T/47
@KBD	8	08H	T/37	@POSN	66	42H	T/48
@KEYIN	9	09H	T/38	@READ	67	43H	T/49
@DSPLY	10	0AH	T/39	@REW	68	44H	T/48
@LOGER	11	0BH	T/89	@SKIP	72	48H	T/48
@LOGOT	12	0CH	T/89	@VER	73	49H	T/49
@MSG	13	0DH	T/36	@WEOF	74	4AH	T/49
@PRINT	14	0EH	T/40	@WRITE	75	4BH	T/50
@VDCTL	15	0FH	T/90	@LOAD	76	4CH	T/56
@PAUSE	16	10H	T/92	@RUN	77	4DH	T/57
@PARAM	17	11H	T/74	@FSPEC	78	4EH	T/78
@DATE	18	12H	T/60	@FEXT	79	4FH	T/44
@TIME	19	13H	T/61	@FNAME	80	50H	T/45
@CHNIO	20	14H	T/36	@GTDCT	81	51H	T/61
@ABORT	21	15H	T/51	@GTDCB	82	52H	T/62
@EXIT	22	16H	T/52	@RDSSC	85	55H	T/85
@CMNDI	24	18H	T/52	@DIRRD	87	57H	T/85
@CMNDR	25	19H	T/52	@DIRWR	88	58H	T/86
@ERROR	26	1AH	T/53	@MUL8	90	5AH	T/72
@DEBUG	27	1BH	T/53	@MUL16	91	5BH	T/72
@CKTSK	28	1CH	T/67	@DIVD8	93	5DH	T/73
@ADTSK	29	1DH	T/68	@DIVD16	94	5EH	T/73
@RMTSK	30	1EH	T/69	@DECBIN	96	60H	T/70
@RPTSK	31	1FH	T/69	@BINDEC	97	61H	T/70
@KLTSK	32	20H	T/69	@HEX8	98	62H	T/71
@CKDRV	33	21H	T/81	@HEX16	99	63H	T/71
@DODIR	34	22H	T/54	@HIGH$	100	64H	T/62
@RAMDIR	35	23H	T/55	@FLAGS	101	65H	T/63
@DCSTAT	40	28H	T/61	@BANK	102	66H	T/58
@SEEK	46	2EH	T/81	@BREAK	103	67H	T/60
@RDHDR	48	30H	T/82	@SOUND	104	68H	T/92
@RDSEC	49	31H	T/82	@DISKIO	121	79H	T/86
@VRSEC	50	32H	T/83	@LOCDCT	122	7AH	T/65
@RDTRK	51	33H	T/83	@LOCDCB	123	7BH	T/66
@WRSEC	53	35H	T/83	@EVAL	124	7CH	T/78
@WRSSC	54	36H	T/84	@WILD	125	7DH	T/93
@WRTRK	55	37H	T/84	@LOCDEV	126	7EH	T/66
@RENAM	56	38H	T/41	@SORT	127	7FH	T/94
@REMOV	57	39H	T/42				

Device I/O Functions -

These twelve SVCs are responsible for providing I/O to and from the character- or byte-oriented devices supported by DOSPLUS IV (@KI, @DO, @PR, @RS, @SO, @SI, @U1, & @U2), as well as files. Five of these SVCs (@GET, @PUT, @CTL, @CHNIO, @MSG) are applicable to any device or file. The remaining SVCs each address a specific DOSPLUS system device. The general byte I/O SVCs are:

@GET	@PUT
@CTL	@CHNIO
@MSG	

The keyboard-related SVCs are:

@KEY	@KBD
@KEYIN	

The video display-oriented SVCs are:

@DSP	@DSPLY

And finally, the printer-related SVCs are:

@PRT	@PRINT

Note that all device I/O operations are subject to FORCEing, JOINing, and FILTERing.

@GET
SVC: 3 Dec/03 Hex

@GET is used to fetch a single byte from a character-oriented device or from a file. In use, the DE register must point to the DCB (if fetching a character from a device) or the FCB (if from a file) desired. After executing the @GET, the Z flag indicates whether or not the operation was successful.

ENTRY: A = 3 Dec/03 Hex
DE => DCB or FCB

EXIT: If ZF, function successful
A=Character
If NZ: If A=0, no character available.
If A≠0, A = Error code.

EXAMPLE:

```
;
;       WAIT FOR CHR FROM DEVICE
;
;       ENTRY:
;       C=LOGICAL DEVICE NUMBER
;
;       EXIT:
;       IF ZF, A=CHARACTER FROM DEVICE
;       IF NZ, A=ERROR CODE
;
GETCHR LD     A,@LOCDCB     ;FIND DCB
       RST    28H           ;DE=>DCB
GETCH0 LD     A,@GET        ;LOOK FOR CHR
       RST    28H           ;
       RET    Z             ;GOT CHR - DONE
       OR     A             ;NO CHR AVAILABLE?
       JR     Z,GETCH0      ;YES - TRY AGAIN
       RET                  ;RETURN ERROR IN A
```

@PUT
SVC: 4 Dec/04 Hex

The @PUT SVC is used to output a single byte to a character-oriented device or file. The DE register pair points to a DCB or FCB, and the C register contains the byte to be output. On return, the Z flag indicates the success or failure of the operation.

ENTRY: A = 4 Dec/04 Hex
C = Byte to output
DE => DCB or FCB

EXIT: If ZF: @PUT successful.
If NZ: A = Error code.

EXAMPLE:

```
;
;       OUTPUT A BLOCK OF DATA TO A DEVICE
;
;       ENTRY:
;       C=DEVICE NUMBER
;       HL=>DATA TO OUTPUT, TERMINATED WITH 00H
;
OUTPUT LD     A,@LOCDCB     ;LOCATE DCB
       RST    28H           ;DE=>DCB
OUTPT0 LD      A,(HL)       ;GET DATA
       INC    HL            ;HL=>NEXT BYTE
       OR     A             ;DATA=00H?
       RET    Z             ;IF DONE
       LD     C,A           ;PUT DATA IN C
       LD     A,@PUT        ;OUTPUT CHR TO DEVICE
       RST    28H
       JR     Z,OUTPT0      ;TILL DONE
       RET                  ;IF ERROR
```

==========

@CTL
SVC: 5 Dec/05 Hex

This SVC is used to perform I/O in the CTL mode. Typically, this mode is used to control the status of physical devices or their driver programs. The following CTL codes are TRSDOS standards, although individual device drivers may redefine them or support additional codes.

CTL code	Driver Action
0	Fetch device status
1	Generate an interrupt or BREAK
2	Initialize driver
3	Flush driver buffers
4	Set driver interrupt vector (IY=address, or 0 to reset)
8	Fetch next character from device

ENTRY: A = 5 Dec/05 Hex
C = CTL function code
DE => DCB

EXIT: If C=0: If ZF, device is ready
If NZ, device is not ready
A = device status code

If C=1,2,3: If ZF, function successful
If NZ, A=Error code

If C=4: If ZF, function successful
IY=old vector address
If NZ, A=Error code

If C=8: If ZF, A=next character
If NZ: If A=0, no character available
If A≠0, A=Error code

==========

@CHNIO
SVC: 20 Dec/14 Hex

The CHNIO SVC is used to provide any of the three basic character I/O functions @GET, @PUT, or @CTL. In use, the IX register pair points to the DCB or FCB with which I/O is to be performed, and the B register contains a functions code which determines the type of I/O to take place:

Code	I/O Type
1	@GET
2	@PUT
4	@CTL

ENTRY: A = 20 Dec/14 Hex
B = Function code {1,2,4}
C = Output character (If @PUT or @CTL)
IX => DCB or FCB

EXIT: (Exit conditions are device-driver dependent. The following are standard conventions followed by DOSPLUS IV drivers.)

If @GET performed:
If ZF, A=Character
If NZ: If A=0, no character available
If A≠0, A=Error code

If @PUT or @CTL performed:
If ZF, function successful
If NZ, A=Error code

@MSG
SVC: 13 Dec/0D Hex

This SVC will output a block of text to any character-oriented device or file. The text block must be terminated with either a carriage return, 0DH, or an ETX, 03H. If the block is terminated with a carriage return, the carriage return is output as part of the block. If an ETX is used as the terminator, is is not output to the device. The DE register pair is used to point to the DCB or FCB associated with the device or file to which the text is to be output.

ENTRY: A = 13 Dec/0D Hex
DE => DCB or FCB
HL => Text block, terminated with CR or ETX

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@KEY
SVC: 1 Dec/01 Hex

This SVC will scan the Model 4's keyboard and wait for a keypress. Note that @KEY does not return to the caller until a keypress is fetched.

ENTRY: A = 1 Dec/01 Hex

EXIT: If ZF, A=Character
If NZ, A=Error code

DE is altered

@KBD
SVC: 8 Dec/08 Hex

This routine scans the keyboard to determine if a key has been depressed, and if so, it returns the value of that key in the accumulator.

ENTRY: A = 8 Dec/08 Hex

EXIT: If ZF, A=Character
If NZ: If A=0, no character available
If A≠0, A=Error code

DE is altered

@KEYIN
SVC: 9 Dec/09 Hex

This SVC will accept an entire line of input from the keyboard. The calling program specifies the location of the input buffer and the maximum number of characters to accept. @KEYIN is terminated by pressing <ENTER> or <BREAK>. Note that the input buffer must be large enough to accommodate both the input string and a 1-byte terminator (0DH or 80H).

ENTRY: A = 9 Dec/09 Hex
B = Maximum number of characters to accept
HL => Input buffer

EXIT: B = Number of characters accepted (less terminator)
C = Original field length (same as B on entry)
HL => Input data

If ZF, function successful
If CF, <BREAK> key was pressed
If NZ, A=Error code

DE is altered

@DSP
SVC: 2 Dec/02 Hex

@DSP is used to display a character on the video display. The character is placed in the C register upon entry to @DSP.

ENTRY: A = 2 Dec/02 Hex
C = Character to output

EXIT: If ZF, function successful
If NZ, A=Error code

DE is altered

==========

@DSPLY
SVC: 10 Dec/0A Hex

This routine is used to display an entire block of text on the video display. The block of text to be displayed must be terminated with either a carriage return, 0DH, or an ETX, 03H. If the block is terminated with 0DH, the carriage return is displayed, moving the video cursor to the beginning of the next line on the display. If the block is terminated with 03H, the cursor remains on the character position following the last character displayed.

ENTRY: A = 10 Dec/0A Hex
HL => Text block to be displayed

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

==========

@PRT
SVC: 6 Dec/06 Hex

This SVC is used to output a single character to the @PR, or printer device. The character to print is contained in the C register. Note that the built-in printer driver in DOSPLUS IV will wait several seconds for the printer to become ready before returning the 'Device not available' error.

ENTRY: A = 6 Dec/06 Hex
C = Character to output

EXIT: If ZF, function successful
If NZ, A=Error code

DE is altered.

==========

@PRINT
SVC: 14 Dec/0E Hex

This routine is used to output a block of text to the @PR, or printer device. The block of text may be termiated either with a carriage return, 0DH, or with an ETX, 03H. If the 0DH is used, the carriage return is printed as part of the text block, causing the printer to begin a new line. If the 03H is used, the printer carriage is not advanced, and any subsequent characters are printed following the last character in the block.

ENTRY: A = 14 Dec/0E Hex
HL => Text block to output

EXIT: If ZF, function successful
If NZ, A=Error code

File Handler Functions -

These SVCs are used to manipulate disk files under DOSPLUS IV. With them, files may be created, written to, read from, renamed, and deleted. This group of file-handling SVCs is further broken down into three general groups:

A. File control SVCs. These SVCs perform assorted functions related to files, but they do not operate upon the data within a file. They include:

@RENAME	@CLOSE
@REMOV	@FEXT
@INIT	@FNAME
@OPEN	

B. File positioning SVCs. These SVCs allow the user to access any record within a file, as well as providing information on the length of the file and the current position within the file. They include:

@BKSP	@POSN
@CKEOF	@REW
@LOC	@SKIP
@LOF	@WEOF
@PEOF	

C. File I/O SVCs. These are the SVCs which allow the user to read data from or write data to a disk file. They are:

@READ	@VER
@WRITE	

Note that the @GET and @PUT SVCs listed under Device I/O Functions are also useful in file I/O.

@RENAM
SVC: 56 Dec/38 Hex

This routine is used to change the name and/or extension of an existing file. Note that a file password cannot be changed with @RENAM.

ENTRY: A = 56 Dec/38 Hex
DE => FCB containing current filename
HL => FCB containing new filename

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@REMOV
SVC: 57 Dec/39 Hex

@REMOV is used to remove, or kill, a file. The file's disk space is de-allocated and its directory entry is marked as deleted. @REMOV may also be used with devices. In this case, the device is simply closed.

ENTRY: A = 57 Dec/39 Hex
DE => Open FCB or DCB

EXIT: If ZF, function successful
If NZ, A=Error code

@INIT
SVC: 58 Dec/3A Hex

@INIT is used to OPEN exisiting files or to create new files. @INIT may also be used to open devices for I/O.

ENTRY: A = 58 Dec/3A Hex
B = Logical record length with which to open file (0=256)
DE => FCB containing file or device name
HL => 256-byte I/O buffer

EXIT: If ZF: Function successful
If CF, new file created
If NC, existing file/device OPENed
If NZ, A=Error code

AF is altered

EXAMPLE:

```
;
;       INIT A FILE
;
;
IFILE   LD      HL,INBUF        ;HL=>INPUT BUFFER
        LD      B,25            ;B=MAX INPUT FIELD LEN
        LD      A,@KEYIN        ;GET KEYBOARD INPUT
        RST     28H
        RET     C               ;ABORT IF <BREAK> PRESSED
        LD      DE,FCB          ;DE=>FILE CONTROL BLOCK
        LD      A,@FSPEC        ;MOVE FILENAME INTO FCB
        RST     28H
        JR      NZ,IFILE        ;IF ERROR, TRY AGAIN
        LD      HL,FBUFF        ;FILE I/O BUFFER
        LD      B,0             ;LRL=256
        LD      A,@INIT         ;INIT FILE
        RST     28H
        RET     NZ              ;ABORT ON ERROR
        JR      C,NEWFIL        ;IF NEW FILE CREATE
OLDFIL  .                       ;HERE IF OLD FILE OPENED
        .
        .
INBUF   DEFS    26
FBUFF   DEFS    256
FCB     DEFS    32
```

==========

==========

@OPEN
SVC: 59 Dec/3B Hex

This SVC is used to open, or prepare, a disk file for I/O. Before executing @OPEN, the DE register pair must point to a 32-byte FCB containing the name of the file to open, HL must point to a 256-byte file I/O buffer, and the B register must contain the logical record length with which the file is to be opened. If the desired LRL is 256, a value of 0 should be used. Note that @OPEN may be used with devices as well as files.

ENTRY: A = 59 Dec/3B Hex
B = Logical record length (0=256)
DE => FCB or DCB containing file/device name
HL => 256-byte I/O buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

@CLOSE
SVC: 60 Dec/3C Hex

This SVC is used to close an open file, writing any residual data from the file's RAM buffer to disk and updating the file's directory entry. All files, once OPENed, must be CLOSEd. @CLOSE may also be used to CLOSE a device, in which case the @CTL SVC outputs an 03H (flush buffers) to the device.

ENTRY: A = 60 Dec/3C Hex
DE => Open FCB or DCB

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@FEXT
SVC: 79 Dec/4F Hex

@FEXT is used to append an extension to te filename in an unopen FCB. If the filename already contains an extension, @FEXT has no effect.

ENTRY: A = 79 Dec/4F Hex
DE => FCB containing filename
HL => 3-character extension (padded on the left with spaces if necessary)

EXIT: If ZF, extension not added
If NZ, extension added

AF, BC, & HL are altered

EXAMPLE:

```
START   LD    HL,INBUF      ;INPUT BUFFER
        LD    B,25          ;INPUT FIELD LEN
        LD    A,@KEYIN      ;GET FILENAME FROM KBD
        RST   28H
        LD    DE,FCB        ;DE=>FILE CONTROL BLOCK
        LD    A,@FSPEC      ;MOVE FILENAME INTO FCB
        RST   28H
        LD    HL,EXT        ;HL=>DEFAULT EXTENSION
        LD    A,@FEXT       ;APPEND EXT
        RST   28H
        .
        .
        .
EXT     DEFM  'DAT'         ;DEFAULT EXTENSION
INBUF   DEFS  26            ;INPUT BUFFER
FCB     DEFS  32            ;FILE CONTROL BLOCK
```

@FNAME
SVC: 80 Dec/50 Hex

@FNAME fetches a diskette filename into a user-specified RAM buffer, given a logical drive number and a logical file number (LFN). Note that @CKDRV should be used before executing @FNAME to insure that the desired drive is ready for I/O.

ENTRY: A = 80 Dec/50 Hex
B = LFN
C = Logical drive number
DE => 16-byte buffer to receive filename

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@BKSP
SVC: 61 Dec/3D Hex

This SVC is used to backspace a file one logical record. For example, if a file is currently positioned to record 52, executing @BKSP would position the file to record 51. If an attempt is made to backspace past the first record of the file, an error 1DH will result.

ENTRY: A = 61 Dec/3D Hex
DE => FCB

EXIT: If ZF, function successful
If NZ, A=Error code

@CKEOF
SVC: 62 Dec/3E Hex

The @CKEOF SVC is used to determine if a file is currently positioned to the end of the file.

ENTRY: A = 62 Dec/3E Hex
DE => FCB

EXIT: If ZF, file is not positioned to EOF
If NZ: If A=1CH, file is positioned to EOF
If A=1DH, file is positioned past EOF
If A≠1CH or 1DH, A=Error code

EXAMPLE:

```
;
;       . READ FILE INTO RAM BUFFER
;
;         ENTRY:
;         DE=>DCB CONTAINING FILENAME
          HL=>BUFFER
;
RDFILE  PUSH  HL              ;SAVE BUFFER ADDR
        LD    B,0             ;LRL=256
        LD    HL,FBUFF        ;HL=>FILE I/O BUFFER
        LD    A,@OPEN         ;OPEN FILE
        RST   28H
        POP   HL              ;RESTORE BUFFER
        RET   NZ              ;IF ERROR, ABORT
RDFIL0  LD    A,@CKEOF        ;AT END OF FILE?
        RST   28H
        JR    NZ,RDFIL1       ;MAYBE - CHECK IT OUT
        LD    A,@GET          ;OTHERWISE, READ A BYTE
        RST   28H             ;FROM FILE
        LD    (HL),A          ;STORE IN BUFFER
        INC   HL
        JR    RDFIL0
RDFIL1  CP    1CH             ;EOF?
        RET                   ;ZF STAT IF YES & RETURN
```

@LOC
SVC: 63 Dec/3F Hex

@LOC may be used to determine the current logical record number to which a file is positioned.

ENTRY: A = 63 Dec/3F Hex
DE => FCB

EXIT: A = MSB of logical record number
BC = LSBs of logical record number

@LOF
SVC: 64 Dec/40 Hex

This SVC provides the logical record number of the last record in a file.

ENTRY: A = 64 Dec/40 Hex
DE => FCB

EXIT: A = MSB of ending logical record number
BC = LSBs of ending logical record number
If NZ, A=Error code

@PEOF
SVC: 65 Dec/41 Hex

@PEOF positions a file to end-of-file; that is, the position in which the file should be in order to append new data. If @PEOF is successful, the error code 1CH ('Attempted to read past EOF') is returned.

ENTRY: A = 65 Dec/41 Hex
DE => FCB

EXIT: If A=1CH, function successful
If A≠1CH, A=Error code

EXAMPLE:

```
;
;       APPEND A CONTROL-Z TO FILE
;
;       ENTRY:
;       DE=>OPEN FCB
;
FTERM   LD      A,@PEOF         ;GO TO EOF
        RST     28H
        CP      1CH             ;EOF ERROR?
        RET     NZ              ;IF NOT - ABORT
        LD      C,26            ;CONTROL-Z
        LD      A,@PUT          ;WRITE TO FILE
        RST     28H
        RET
```

@POSN
SVC: 66 Dec/42 Hex

The @POSN SVC allows the user to position a file to any logical record number desired. Use of @POSN automatically sets bit 6 of FCB+1 to indicate that the file is being accessed in a random fashion; this prevents the operating system from shrinking the file if the NRN is less than the ERN.

ENTRY: A = 66 Dec/42 Hex
BC => Logical record number
DE => FCB

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@REW
SVC: 68 Dec/44 Hex

This SVC positions a file to its beginning record.

ENTRY: A = 68 Dec/44 Hex
DE => FCB

EXIT: If ZF, function successful
If NZ, A=Error code

AF is destroyed

@SKIP
SVC: 72 Dec/48 Hex

@SKIP is used to position the file to the logical record following the current logical record. No data transfer takes place. Note that it is possible to @SKIP outside the bounds of the file, resulting in a 1CH or 1DH error code.

ENTRY: A = 72/48 Hex
DE => FCB

EXIT: If ZF, function successful
If NZ, A=Error code

@WEOF
SVC: 74 Dec/4A Hex

The @WEOF SVC causes the operating system to write the current end-of-file to a file's directory entry. @WEOF also forces any residual data in the file's I/O buffer to be written to disk.

ENTRY: A = 74 Dec/4A Hex
DE => FCB

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@READ
SVC: 67 Dec/43 Hex

@READ transfers one logical record from a disk file into RAM. If the file was OPENed with a logical record length of 256, the data is placed into the 256-byte file I/O buffer specified at time of @OPEN or @INIT. If the LRL≠256, the user must specify a logical record buffer of length=LRL, pointed to by the HL register pair.

ENTRY: A = 67 Dec/43 Hex
DE => FCB
HL => Logical record buffer (if LRL≠256)

EXIT: If ZF, function successful
If NZ, A=Error code

@VER
SVC: 73 Dec/49 Hex

@VER writes a logical record to a disk file. If the file has been OPENed or INITed with a logical record length of 256, the data written is taken from the 256-byte file I/O buffer specified at time of @OPEN or @INIT. If the LRL≠256, the user must point the HL register pair to a buffer containing the logical record to be written to disk. This routine does not perform a read-after-write as was done in earlier operating systems designed for use with the less reliable hardware of yesteryear.

ENTRY: A = 73 Dec/49 Hex
DE => FCB
HL => Logical record buffer (if LRL≠256)

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@WRITE
SVC: 75 Dec/4B Hex

The @WRITE SVC writes a logical record to a disk file. If the file has been OPENed or INITed with LRL=256, the data written to disk is taken from the 256-byte file I/O buffer specified at time of @OPEN or @INIT. If the LRL≠256, the data is taken from a buffer pointed to by the HL register pair.

ENTRY: A = 75 Dec/4B Hex
DE =>FCB
HL =>Logical record buffer (if LRL≠256)

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

System Control & Information Functions -

These twenty-three SVCs are divided into two general groups:

A. System control functions. These SVCs perform functions such as passing library commands to the DOS for execution, reading data from diskette directories, displaying system error messages and entering the DOS command level. These SVCs are:

@IPL	@DODIR
@ABORT	@RAMDIR
@EXIT	@LOAD
@CMNDI	@RUN
@CMNDR	@BANK
@ERROR	@BREAK
@DEBUG	

B. System information functions. These SVCs return information about the system, such as the current time and date, the location of DCBs and DCTs, the current value of HIGH$, and the location of various system flags. They include:

@DATE	@HIGH$
@TIME	@FLAGS
@DCSTAT	@LOCDCT
@GTDCT	@LOCDCB
@GTDCB	@LOCDEV

@IPL
SVC: 0 Dec/00 Hex

This SVC, which stands for Initial Program Loader, is used to cause the system to reset and load the bootstrap loader program.

ENTRY: A = 0 Dec/00 Hex

EXIT: None

@ABORT
SVC: 21 Dec/15 Hex

@ABORT is similar in function to @EXIT, below. @ABORT returns control to the operating system command level. The DEBUG monitor is invoked if active.

ENTRY: A = 21 Dec/15 Hex

EXIT: None

@EXIT
SVC: 22 Dec/16 Hex

The @EXIT SVC is used to return to the DOS command level from another program. Any program returning to the DOS command level via this SVC should return error status in the HL register pair. If an error has occured, the L register should hold the error code, and the H register should contain a 00H. If no error has occured, HL should contain a 0000H.

ENTRY: A = 22 Dec/16 Hex

EXIT: None

@CMNDI
SVC: 24 Dec/18 Hex

The @CMNDI SVC is used to execute a DOSPLUS library command or executable program. The user passes a valid DOSPLUS command line to the operating system, pointed to by the HL register pair and terminated with a carriage return, 0DH, ETX, 03H, oe semicolon. @CMNDI returns to the DOSPLUS command level after execution. Any program executed by this SVC should return error status in the HL register pair. If an error has occured, the L register should hold the error code, and the H register should contain a 00H. If no error has occured, HL should contain a 0000H.

ENTRY: A = 24 Dec/18 Hex
HL =>Command line terminated with 0DH or 03H

EXIT: None

@CMNDR
SVC: 25 Dec/19 Hex

@CMNDR is similar to @CMNDI above, with the exception that @CMNDR returns to the caller after execution rather than aborting to the DOSPLUS command level. @CMNDR accepts a valid DOSPLUS command line, terminated by a carriage return, 0DH, an ETX, 03H, or a semicolon. Upon return to the calling program, the HL register pair contains an error code generated by the called library command or program. An error code of 0000H indicates no error, and FFFFH indicates an undefined error. Otherwise, the L register contains a DOSPLUS IV error code.

ENTRY: A = 25 Dec/19 Hex
HL => Command line terminated with 0DH or 03H

EXIT: HL = Return code

AF, BC, DE, IX, & IY are altered

@ERROR
SVC: 26 Dec/1A Hex

@ERROR will display an error message on the video display, given a DOSPLUS error code. Two types of error messages are possible:

A. The normal error message
and
B. The extended error message

The normal error message is displayed if bit 6 of the error code is set. If bit 6 is reset, the normal error message will be displayed in addition to information regarding the address at which the error occued and the precise error code involved. For instance, calling @ERROR with the error code 48H (error code 8, bit 6 set) will result in the error message:

Device not available

however, calling @ERROR with the error code 08H (error code 8, bit 6 reset) will result in the message:

Device not available - Referenced at xxxx on yy

where 'xxxx' is the hexadecimal address at which @ERROR was called and 'yy' is the hex error code passed to the routine.

Bit 7 of the error code controls whether @ERROR will exit to the caller or to the DOSPLUS @ABORT routine. If bit 7 is set, @ERROR will exit back to the calling program. If bit 7 is reset, @ERROR will return to the DOSPLUS command level by way of the @ABORT routine.

ENTRY: A = 26 Dec/1A Hex
C = Error code

EXIT: No exit if register C, bit 7 reset

@DEBUG
SVC: 27 Dec/1B Hex

This SVC invokes the DOSPLUS DEBUG monitor. Upon entry to DEBUG, the monitor's PC register is set to the address following the RST28H which entered DEBUG.

ENTRY: A = 27 Dec/1B Hex

EXIT: None

@DODIR
SVC: 34 Dec/22 Hex

This SVC provides several functions relating to reading/displaying a diskette directory. Upon entry to the @DODIR routine, the B register acts as a function switch to determine the type of action the routine will take. The C register indicates the logical drive number whose directory is to be read, and the HL register pair is used to point to various data.

Two function codes, 2 & 3, utilize a partspec. A partspec is a 3-character mask used to select only those files whose extension match the mask. Dollar sign symbols may be placed in the partspec as 'don't care' characters. For example, the partspec 'DAT' would match any file with the extension /DAT. A partspec of 'IN$' would match any file whose extension begins with the characters /IN, such as /INX or /IND.

Display catalog listing

These two function codes produce a catalog listing of a diskettes contents on the video display. Function code 2 allows the use of a partspec.

ENTRY: A = 34 Dec/22 Hex
B = 0 or 2
C = Logical drive number (00H-0FH)
HL => 3-character partspec (If B=2)

EXIT: If ZF, function successful
If NZ, A=Error code

Fetch directory entries

These two function codes allow the user to read 18-byte records concerning each visible user file on a diskette into a RAM buffer. These records consist of the first 16 bytes of the a file's directory entry (see section III for information on the structure of directory entries) followed by two bytes containing the ending record number of the file. The end of the list of records is signified by a an FFH byte. Note that your buffer must be large enough to accomodate all of the visible user file records read from the disk. Function code 3 allows the use of a partspec to restrict the file records read into RAM.

ENTRY: A = 34 Dec/22 Hex
B = 1 or 3
C = Logical drive number (00H-0FH)
HL => RAM buffer to receive file records
If B=3, the buffer pointed to by HL must contain the partspec to be used upon entry to @DODIR. Note that this partspec is destroyed by @DODIR when the buffer is filled with file records.

EXIT: If ZF, function successful
If NZ, A=Error code

Fetch diskette information

@DODIR function code 4 will fetch the diskette name, date, free and allocated space information into a 20-byte RAM buffer. The data is arranged as follows:

HL+00H=>	Diskette name (8 bytes)
HL+08H=>	Diskette date (8 bytes)
HL+10H=>	Number of kilobytes of disk space currently allocated to files (2 bytes)
HL+12H=>	Number of kilobytes of disk space free (2 bytes)

ENTRY: A = 34 Dec/22 Hex
B = 4
C = Logical driver number {00H-0FH}
HL => RAM buffer to receive data

EXIT: If ZF, function successful
If NZ, A=Error code

==========

==========

@RAMDIR
SVC: 35 Dec/23 Hex

The @RAMDIR SVC reads file directory information for single or multiple files, or alternatively, it may read free space information from the diskette.

Read file directory information

In this mode, @RAMDIR may read information from file directory entries. It may read either a single entry or all the entries of visible user files on the disk. The information is read into RAM in the format:

ENTRY+00H =>	Filename in 'filename/ext:d' form, padded with spaces on the right (15 bytes)
ENTRY+0FH =>	File protection level {0-6} (1 byte)
ENTRY+10H =>	End of file byte (1 byte)
ENTRY+11H =>	Logical record length (1 byte)
ENTRY+12H =>	Ending record number (2 bytes)
ENTRY+14H =>	File length in K (2 bytes)

If single or multiple entries are read into RAM, the end of the list is marked by a plus sign, '+' in the ENTRY+00H position following the last entry.

ENTRY: A = 35 Dec/23 Hex
B = Logical drive number {00H-0FH}
C = Function switch
If C=0, read all visible user file directory info into RAM
If C≠0 and C≠255, C=Logical file number-1 of directory record to fetch
HL => Buffer to receive directory data

EXIT: If ZF, function successful
If NZ, A=Error code

Read allocated and free space information

If the function switch contained in the C register has the value 255, @RAMDIR returns information regarding allocated and free space on a diskette. The information is deposited into a user-specified region of RAM and is arranged as shown:

HL+00=> Space allocated in K (2 byte)
HL+02=> Space free in K (2 bytes)

ENTRY: A = 35 Dec/23 hex
B = 255
C = Logical drive number {00H-0FH}
HL => RAM buffer to accept free space info

EXIT: If ZF, function successful
If NZ, A=Error code

==========

==========

@LOAD
SVC: 76 Dec/4C Hex

This SVC will load an object file into RAM. This file must be saved in load module format or an error will occur.

ENTRY: A = 76 Dec/4C Hex
DE => FCB containing filename

EXIT: If ZF, function successful
HL = Program transfer address
If NZ, A=Error code

EXAMPLE:

```
;
;       LOAD PROGRAM MODULE FROM DISK
;
;       ENTRY:
;       A=MODULE NUMBER
;
START   LD    (FNAME+3),A     ;PLACE MOD # IN FILENAME
        LD    HL,FNAME        ;HL=>FILENAME
        LD    DE,FCB          ;DE=>FCB
        LD    A,78            ;DO @FSPEC
        RST   28H
        LD    A,76            ;LOAD FILE INTO RAM
        RST   28H
        RET   NZ              ;RETURN IF ERROR
        LD    (EXADD),HL      ;SAVE ENTRY ADDRESS
        RET
FNAME   DEFM  'MODX/CMD'      ;FILENAME
        DEFB  13
FCB     DEFS  32              ;FILE CONTROL BLOCK
```

@RUN
SVC: 77 Dec/4D Hex

The @RUN SVC loads and transfers control to a program file stored on disk. The file must be saved in load module format.

ENTRY: A = 77 Dec/4D Hex
DE => FCB containing filename
HL => Optional command line passed to program

EXIT: If NZ, A=Error code
If no error, @RUN transfers control to the loaded program. Upon entry to the program, the following Z-80 registers contain the data shown below:

BC => Start of DOS input buffer
DE => Top of user RAM
HL => Next character in command line

@BANK
SVC: 102 Dec/66 Hex

@BANK performs five functions related to the management of the Model 4's bank-switched RAM.

The first function of @BANK controls the Model 4's one primary 32K and two auxilliary 32K RAM banks. The Model 4's memory is divided into four 32K regions. The first section, which occupies the address space from 0000H to 7FFFH, is always available and @BANK does not affect it. Any one of three banks of RAM may be assigned to the upper 32K of RAM, from 8000H to FFFFH. The primary bank, named Bank 0, normally occupies this region. @BANK may replace the bank currently assigned to the 8000H-FFFFH address space with either of the two other banks.

@BANK also allows the user to 'mark' a bank as available or as in-use, as well as providing a means of testing the available/in-use status of a bank and fetching the number of the currently active bank.

Select a bank

This function selects any one of three RAM banks to occupy the address space from 8000H to FFFFH. The Z-80 stack pointer, SP, must point to an address below 8000H when calling @BANK or an error code 2BH, 'SVC parameter error', will be returned.

ENTRY: A = 102 Dec/66 Hex
B = 0
C = Bank number {0-2}
HL = Optional transfer address within selected bank
(If bit 7 of register C is set)

EXIT: If ZF, function successful
C = Previously selected bank. If bit 7 of register C had been set upon entry to @BANK, it will remain set upon exit.

If @BANK had been entered with bit 7 of register C set, the routine does not exit to the caller but to the address specified by HL in the bank selected by C. Upon entry to the specified routine, HL contains the return address of the caller.

If NZ, A=Error code

Flag a bank as available

This function will flag any of the three switchable banks as available for use.

ENTRY: A = 102 Dec/66 Hex
B = 1
C = Bank number {0-2}

EXIT: If ZF, function successful
If NZ, bank not available

Get bank available status

This function is used to ascertain if a bank has been flagged as available or as in-use.

ENTRY: A = 102 Dec/66 Hex
B = 2
C = Bank number {0-2}

EXIT: If ZF, bank available
If NZ, A=Error code
If A=2BH, bank is in-use or non-existent
If A≠2BH, an invalid bank number was specified on entry

Flag bank as in-use

A bank may be marked as 'in-use' with this function.

ENTRY: A = 102 Dec/66 Hex
B = 3
C = Bank number {0-2}

EXIT: If ZF, function successful
If NZ, A=Error code
If A=2BH, bank is already in-use or does not exist
If A≠2BH, an invalid bank number was specified on entry

Fetch current bank number

This function of the @BANK SVC is used to determine which bank is currently selected.

ENTRY: A = 102 Dec/66 Hex
B = 4

EXIT: A = Current bank number

@BREAK
SVC: 103 Dec/67 Hex

The @BREAK SVC is used to set the <BREAK> key vector. Each time a real-time clock (RTC) interrupt is generated, the interrupt processing routine checks to see if the <BREAK> key has been depressed. If so, program execution is transferred to the address specified by the <BREAK> vector. Note that program execution will not be transferred to the <BREAK> vector routine if the interrupted routine lies below 2400H.

If the <BREAK> key is to be active, bit 4 of SFLAG$ must be reset after calling @BREAK, as the SVC sets this flag.

ENTRY: A = 103 Dec/67 Hex
HL = <BREAK> vector (if HL=0, <BREAK> vector is disabled)

EXIT: HL = Old <BREAK> vector

@DATE
SVC: 18 Dec/12 Hex

This SVC performs two functions:

(A) It returns the current system date into a user-specified buffer in MM/DD/YY format

(B) It returns a pointer to the area of RAM in which the system date is stored. This is useful if a user program must set the system date.

ENTRY: A = 18 Dec/12 Hex
HL => 8-byte buffer to receive system date in MM/DD/YY format

EXIT: DE => System date storage (3-bytes, stored in day,month,year order)
HL => Byte following end of user buffer

AF and BC are altered

@TIME
SVC: 19 Dec/13 Hex

This SVC provides two items of information:

(A) It returns the current system time into a user-specified buffer in HH:MM:SS format

(B) It returns a pointer to the area of RAM in which the system time is stored. This is useful if a user program must set the system time.

ENTRY: A = 19 Dec/13 Hex
HL => 8-byte buffer to receive system time in HH:MM:SS format

EXIT: DE => System time storage (3-bytes, stored in second,minute,hour order)
HL => Byte following end of user buffer

AF and BC are altered

@DCSTAT
SVC: 40 Dec/28 Hex

@DCSTAT may be used to check if a disk drive currently has an active DCT assigned to it.

ENTRY: A = 40 Dec/28 Hex
C = Logical drive number {00H-0FH}

EXIT: If ZF, drive has an active DCT assigned to it
If NZ, drive has no DCT assigned to it or has a NIL DCT assigned to it

@GTDCT
SVC: 81 Dec/51 Hex

This SVC fetches the address of the drive control table for any given logical drive number.

ENTRY: A = 81 Dec/51 Hex
C = Logical drive number {00H-0FH}

EXIT: IY => Drive control table

AF is altered

@GTDCB
SVC: 82 Dec/52 Hex

This SVC locates the DCB for any character-oriented system device. The DE register pair must contain the 2-character name of the device; The E register contains the first character of the device name, and the D register contains the second character of the name.

ENTRY: A = 82 Dec/52 Hex
DE = Device name

EXIT: If ZF, function successful
HL => DCB
If NZ, A=Error code

@HIGH$
SVC: 100 Dec/64 Hex

The HIGH$ SVC allows the user to set or read the value of the HIGH$ (top of available RAM) and LOW$ (bottom of available RAM) pointers. The B register is used to determine whether HIGH$ or LOW$ is to be operated upon. If B=0, HIGH$ is selected. If B≠0, LOW$ is indicated. The HL register determines whether the HIGH$ or LOW$ value is to be set or read. If HL=0000H, then the value is to be read. If HL≠0000H, the value is to be set.

ENTRY: A = 100 Dec/64 Hex
B = HIGH$/LOW$ switch (HIGH$: B=0, LOW$: B≠0
HL = New HIGH$ or LOW$ value, or 0000H to read value

EXIT: If ZF, function successful
If HL=0000H on entry, HL=Current HIGH$ or LOW$
If NZ, A=Error code

@FLAGS
SVC: 101 Dec/65 Hex

The @FLAGS SVC returns a pointer to a block of system flags and other data.

ENTRY: A = 101 Dec/65 Hex

EXIT: IY => System flags/data area

The system flags and data block contains the following information:

Location	Data	
IY+1	Current floppy drive number	
IY+2	CFLAG$:	Bit 7: When set, all calls to @ERROR write the error message string into the buffer specified by the DE register pair rather than to the @DO device
		Bit 6: When set, error codes 0-62 Dec/00-3E Hex will not display an error message through the @ERROR SVC
		Bit 5: Reserved
		Bit 4: Reserved
		Bit 3: Reserved
		Bit 2: Reserved
		Bit 1: Reserved
		Bit 0: Reserved
IY+3	DFLAG$	Bit 7: Reserved
		Bit 6: Internal use
		Bit 5: Reserved
		Bit 4: Reserved
		Bit 3: Reserved
		Bit 2: When set, VERIFY option is in effect
		Bit 1: Reserved
		Bit 0: Reserved
IY+5	FEMSK$	Port FEH image
IY+6	NMIMSK$	Non-maskable interrupt port (E4H) image
IY+8	OVLYL$	Current low overlay number-1
IY+9	OVLYH$	Current high overlay number-1

IY+10 KFLAG$ Bit 7: Reserved
Bit 6: Reserved
Bit 5: Reserved
Bit 4: Reserved
Bit 3: Reserved
Bit 2: When set, <ENTER> has been depressed
Bit 1: When set, <SHIFT>-<@> has been depressed
Bit 0: When set, <BREAK> has been depressed

Note that bits 0,1, & 2 of KFLAG$ are set in backround (during the interrupt routine servicing the 30 Hz RTC interrupt), and they are not automatically reset. Therefore, in order to detect the keys each bit represents, the appropriate bit should first be reset and then tested to determine if it has become set.

IY+11 Last error code posted by @ERROR
IY+12 MODOUT Port ECH image
IY+14 OPREG$ Option port (84H) image
IY+17 XFLAG$ Bit 7: When set, @ERROR produces a tone
Bit 6: When set, indicates that the system has previously allocated a 32-byte DCB and 256-byte I/O buffer for use by the DO library command
Bit 5: Reserved
Bit 4: Reserved
Bit 3: Reserved
Bit 2: Reserved
Bit 1: Reserved
Bit 0: Reserved
IY+18 SFLAG$ Bit 7: When set, the DEBUG monitor is active
Bit 6: When set, forces @ERROR to produce extended error messages
Bit 5: When set, the DO processor is active
Bit 4: Reserved
Bit 3: When set, indicates that the system clock is running at 4 MHz
Bit 2: Reserved
Bit 1: When set, indicates that a file with level 6 protection is being loaded
Bit 0: When set, indicates that the next file to be OPENed is to be OPENed for READ only. That is, that no WRITE operations shall be performed. This bit is automatically reset after a call to @OPEN

IY+21	VFLAG$	Bit 7: Reserved Bit 6: When set, a steady cursor is displayed. When reset, a flashing cursor is displayed Bit 5: Internal use Bit 4: When set, the system time is displayed on the video display Bit 3: Reserved Bit 2: Reserved Bit 1: Reserved Bit 0: Reserved
IY+22	WRINTMSK$	Maskable interrupt port (E0H) image
IY+26	SVCTABPTR$	High order byte of SVC table (Low order byte=00H)
IY+27	OSVER$	Operating system version number (BCD representation. Version 6.0=60H)
IY+28	@ICNFG vector	Program execution passes through this vector upon execution of a /CFG file
IY+31	@KITSK vector	Keyboard driver execution passes through this vector
IY+34	@TRAP vector	Disk I/O handler execution passes through this vector
IY+51	LLOW$	Pointer to lowest available byte of system RAM (2-bytes)
IY+53	LHIGH$	Pointer to highest available byte of system RAM (2-bytes)

==========

==========

@LOCDCT
SVC: 122 Dec/7A Hex

@LOCDCT allows the used to locate the drive control table for any of the 16 logical drives supported by DOSPLUS IV. If @LOCDCT is passed a logical drive number greater than 127 (bit 7 set), the SVC returns the address of DOSPLUS IV's DCTTBL$. This is a list of 16 4-byte entries, each entry corresponding to one of DOSPLUS IV's disk drive devices. The first two bytes of each entry point to the address of the drive's DCT and the second two bytes contain the name of the drive device.

ENTRY: A = 122 Dec/7A Hex
C = Logical drive number {00H-0FH}

EXIT: IY => Drive control table (or DCTTBL$ if C > 127 on entry)
==========

@LOCDCB
SVC: 123 Dec/7B Hex

This SVC is used to locate the DCB associated with any of DOSPLUS IV's character-oriented I/O devices. @LOCDCB may also be used to locate DOSPLUS IV's DCBTBL$, if a logical device number greater than 127 (bit 7 set) is passed to it. The DCBTBL$ is a list of 7 4-byte entries, each of which corresponds to one of DOSPLUS IV's character-oriented devices. The first two bytes of each entry point to the device's DCB, and the second two bytes contain the device's name.

ENTRY: A = 123 Dec/7B Hex
C = Logical device number

EXIT: DE => Device control block (or DCBTBL$ if C > 127 on entry)

@LOCDEV
SVC: 126 Dec/7E Hex

The @LOCDEV SVC is used to determine the logical device number associated with any given device or drive specification. The DE register pair is used to point to the specification prefixed with ':', '@', or '*', and the A register contains both a logical device number and a flag which indicates whether the specification belongs to a disk drive device or a character-oriented device.

ENTRY: A = 126 Dec/7E Hex
DE => Drive/device specification

EXIT: If ZF, function successful
Register A: Bit 4: When set, indicates disk drive device specified
Bits 0-3: Logical drive/device number
If NZ, A=Error code

Interrupt Task Functions

DOSPLUS IV responds to the real time clock (RTC) interrupts which are generated every 16.67 mS on the Model 4. Each time the RTC generates an interrupt, one or more tasks on DOSPLUS IV's interrupt chain are executed. DOSPLUS IV provides a chain of 12 interrupt slots, numbered 0-11. The table below shows how often each slot is executed, and what, if any function is assigned to that slot:

Slot #	Exec. period	Function	Priority
0	266.67 mS	Unassigned	5
1	266.67 mS	Unassigned	6
2	266.67 mS	Unassigned	7
3	266.67 mS	Unassigned	8
4	266.67 mS	Unassigned	9
5	266.67 mS	Unassigned	10
6	266.67 mS	Unassigned	11
7	266.67 mS	Cursor Blink	12
8	33.33 mS	Unassigned	4
9	33.33 mS	Clock Display	3
10	33.33 mS	Time/Date update	2
11	16.67 mS	Spooler	1

When an interrupt slot is assigned to a task, a task control block (TCB) is specified. The first two bytes of the TCB contain the address of the task to be executed when the slot is polled. Data used by the interrupt task routine is often placed following the task address, as the IX register points to the TCB upon entry to the task.

DOSPLUS IV provides five SVCs to manage the interrupt chain:

@CKTSK	@RPTSK
@ADTSK	@KLTSK
@RMTSK	

==========

@CKTSK
SVC: 28 Dec/1C Hex

This SVC is used to determine whether a given interrupt slot is currently assigned to a task. If the slot is assigned, the HL register may be used to locate the task block associated with the slot.

ENTRY: A = 28 Dec/1C Hex
C = Slot number {0-11}

EXIT: If ZF, slot is unassigned
If NZ, slot is assigned to a task
HL-1=>2-byte pointer to task block to which slot is assigned

AF and BC are altered

==========

@ADTSK
SVC: 29 Dec/1D Hex

@ADTSK allows the user to assign a task to a given interrupt slot.

ENTRY: A = 29 Dec/1D Hex
C = Slot number (0-11)
DE => Task control block, consisting of at least a 2-byte pointer to an interrupt task routine

EXIT: HL-1=>2-byte pointer to task block to which slot is assigned

AF and BC are altered

EXAMPLE:

```
;
;       ADD AN INTERRUPT TASK
;       TO INTERRUPT CHAIN
;
START   LD      C,8             ;INT SLOT #8 (33 MS)
        LD      A,28            ;@CKTSK
        RST     28H             ;SLOT OCCUPIED?
        RET     NZ              ;IF YES - RET W/ERROR STAT
        LD      A,29            ;@ADTSK
        LD      DE,TSK8         ;DE=>TASK CONTROL BLOCK
        RST     28H             ;INSERT TASK INTO SLOT
        .
        .
        .
TSK8    DEFW    INTTSK          ;POINTER TO INTERRUPT TASK
        DEFB    15              ;COUNTER
        DEFB    15              ;COUNT=15 (1/2 SECOND)
        DEFB    '*'             ;CHARACTER TO DISPLAY
INTTSK  DEC     (IX+2)          ;DECREMENT COUNTER
        RET     NZ              ;COUNT<>0
        LD      A,(IX+3)        ;RESTORE COUNT
        LD      (IX+2),A
        LD      A,(IX+4)        ;GET CHR
        XOR     9               ;FLIP-FLOP CHR
        LD      (IX+4),A        ;PUT IT BACK
        LD      C,A             ;PUT CHR IN C
        LD      H,0             ;ROW 0
        LD      L,79            ;COLUMN
        LD      B,2             ;FUNC CODE
        LD      A,15            ;@VDCTL
        RST     28H             ;PUT CHR ON VIDEO
        RET
```

@RMTSK
SVC: 30 Dec/1E Hex

This SVC will remove any given interrupt task from the interrupt chain.

ENTRY: A = 30 Dec/1E Hex
C = Slot number {0-11}

EXIT: HL-1=>2-byte pointer to task block

BC and DE are altered

@RPTSK
SVC: 31 Dec/1F Hex

The @RPTSK SVC is used within an executing interrupt task to replace the address of the interrupt processing routine (contained in the first two bytes of the task control block) with another address. The new address must immediately follow the call to @RPTSK, as shown below:

```
        LD      A,31            ;@RPTSK
        RST     28H
        DEFW    NEWTSK          ;POINTER TO NEW ROUTINE
```

ENTRY: A = 31 Dec/1F Hex

EXIT: @RPTSK exits to the interrupted foreground program

@KLTSK
SVC: 32 Dec/20 Hex

@KLTSK is used from within an executing interrupt task to remove the task from the interrupt chain. @KLTSK then returns to the interrupted foreground program.

ENTRY: A = 32 Dec/20 Hex

EXIT: @KLTSK returns to the interrupted foreground program

Base Conversion Functions

DOSPLUS IV provides four base conversion SVCs which facilitate conversion from decimal to binary, binary to decimal, and binary to hexadecimal.

@DECBIN
SVC: 96 Dec/60 Hex

This SVC converts an ASCII decimal string into a 2-byte binary value. The decimal string may be terminated by any non-decimal character (any character outside the range '0'-'9'). If the ASCII decimal string represents a value greater than FFFFH (the largest value that can be represented in 2 bytes), the value returned is modulo 10000H.

ENTRY: A = 96 Dec/60 Hex
HL => ASCII decimal string

EXIT: BC = Binary value of string
HL => Terminating character

AF is altered

@BINDEC
SVC: 97 Dec/61 Hex

@BINDEC converts a 16-bit binary value into a 5-character ASCII decimal string representation, padded on the left with spaces.

ENTRY: A = 97 Dec/61 Hex
HL = Binary value to convert
DE => 5-byte buffer to receive ASCII decimal string

EXIT: DE => Byte following last character of ASCII decimal string
HL => First non-blank character of string

AF, BC, & HL are altered

@HEX8
SVC: 98 Dec/62 Hex

The @HEX8 SVC converts a single binary byte into a 2-character ASCII hexadecimal representation.

ENTRY: A = 98 Dec/62 Hex
C = Binary value to convert
HL => 2-byte buffer to receive ASCII hexadecimal string

EXIT: HL => Byte following last character of hex string

AF is altered

@HEX16
SVC: 99 Dec/63 Hex

This SVC converts a 16-bit binary value into a 4-character ASCII hexadecimal string representation.

ENTRY: A = 99 Dec/63 Hex
DE = Binary value to convert
HL => 4-byte buffer to receive ASCII hexadecimal string

EXIT: HL => Byte following last character of hex string

AF is altered

Arithmetic Functions

Four arithmetic routine are supplied by DOSPLUS IV to perform the operations of 8- and 16-bit multiplication and division. These SVC's are:

@MUL8	@DIVD8
@MUL16	@DIVD16

@MUL8
SVC: 90 Dec/5A Hex

This SVC will multiply two unsigned 8-bit quantities. Any overflow out of the 8-bit product is lost.

ENTRY: A = 90 Dec/5A Hex
C = Multiplier
E = Multiplicand

EXIT: A = Product

DE is altered

@MUL16
SVC: 91 Dec/5B Hex

@MUL16 performs an unsigned multiplication of a 16-bit value with an 8-bit value, resulting in a 3-byte product.

ENTRY: A = 91 Dec/5B Hex
HL = Multiplicand
C = Multiplier

EXIT: A = Product LSB
HL = Product MSBs

DE is altered

@DIVD8
SVC: 93 Dec/5D Hex

The @DIVD8 SVC performs an unsigned 8-bit by 8-bit division.

ENTRY: A = 93 Dec/5D Hex
C = Divisor
E = Dividend

EXIT: A = Quotient
E = Remainder

@DIVD16
SVC: 94 Dec/5E Hex

This SVC performs an unsigned 16-bit by 8-bit division.

ENTRY: A = 94 Dec/5E Hex
C = Divisor
HL = Dividend

EXIT: A = Remainder
HL = Quotient

Command Parsing Functions

These three SVCs are useful in command parsing; that is, the process of obtaining parameter values from a command line. The SVCs are:

@PARAM @EVAL
@FSPEC

@PARAM

SVC: 17 Dec/11 Hex

This SVC is used to extract values from a DOSPLUS parameter list. Three types of parameter values are possible:

(A) Numerical values. These are 2-byte integer values and they may be specified in either decimal or hexadecimal format. Hexadecimal values are appended with an 'H' to denote the base.

(B) String values. A string is a group of text characters, such as a filename, a password. Strings are enclosed in either single (') or double (") quotes.

(C) Logical values. A logical value is a true or flase value, which may be represented by the keywords: ON or YES (true values) and OFF or NO (false values). A true value is represented in 2 bytes as FFFFH, and a false value as 0000H. Note that if a parameter name is given in a parameter list without an accompanying value, the parameter is assigned the logical value of true.

The @PARAM SVC requires a set of data, called a parameter block, to supply certain information such as the names of allowable parameters, and the address at which to place their values. DOSPLUS IV supports two type of parameter blocks. The first type is compatible with the parameter block used on older DOSPLUS systems for the TRS-80 Model I and III, and the second type is compatible with the new structure supported by TRSDOS 6.0. We will term the older, Model I/III-compatible parameter block a type-1 parameter block, and the newer TRSDOS 6.0-compatible style a type-2 parameter block. The two tables below outline the structure of each type:

Type-1 Parameter Block Structure

Byte	Contents
BLOCK+0	6-byte parameter name, left-justified and padded with spaces on the right
BLOCK+6	2-byte pointer to address in which to place parameter value
BLOCK+8	00H to end parameter block, or repeat for another parameter as in BLOCK+0 above

Type-2 Parameter Block Structure

Byte	Contents		
BLOCK+0	128 Dec/80 Hex		
BLOCK+1	Parameter type	Bit 7:	If set, numeric values accepted
		Bit 6:	If set, logical values accepted
		Bit 5:	If set, string values accepted
		Bit 4:	Unused
		Bit 0-3:	Length of parameter name
BLOCK+2	Parameter name (N bytes in length, 0 < N < 16)		
BLOCK+N+2	Response flags (Returned upon exit from the @PARAM SVC)	Bit 7:	Numeric value fetched
		Bit 6:	Logical value fetched
		Bit 5:	String value fetched
		Bits 0-4:	Length of value fetched. If length=0, and the parameter value points to a single or double quote, the value is a null string. If not, the parameter exceeded 31 characters in length.
BLOCK+N+3	Parameter value pointer. This 2-byte value points to an address which will receive the value of the parameter. In the case of a string value, the address which this word points to in turn points to the first character of the string.		
BLOCK+N+4	00H to terminate block, or repeat from BLOCK+0, above, for more parameters.		

ENTRY: A = 17 Dec/11 Hex
DE => Parameter block
HL => Parameter field of command line

EXIT: If ZF, function successful
If NZ, an invalid parameter was encountered. Note that no error code is returned.

EXAMPLE:

This is an example of the @PARAM SVC using a 'type-1' parameter block.

```
;
;       FETCH ADDR, FIND, & CHANGE
;       PARAMETERS FROM COMMAND LINE
;
;       ENTRY:
;       HL=>COMMAND LINE
;
GTPRM   LD      DE,PBLK         ;DE=>PARAMETER BLOCK
        LD      A,17            ;DO @PARAM
        RST     28H
        RET     Z               ;DONE IF NO ERROR
        LD      HL,ERMES        ;DISPLAY AN ERROR
        LD      A,10            ;MESSAGE IF @PARAM
        RST     28H             ;FAILED
        OR      -1              ;RETURN NZ STATUS
        RET
ERMES   DEFM    'PARAMETER ERROR'
        DEFB    13
PBLK    DEFM    'ADDR           ;ADDR PARAMETER
        DEFW    AVAL            ;POINTER TO ADDR VALUE
        DEFM    'FIND           ;FIND PARAMETER
        DEFW    FVAL            ;POINTER TO FIND VALUE
        DEFM    'CHANGE'        ;CHANGE PARAMETER
        DEFW    CVAL            ;POINTER TO CHANGE VALUE
        DEFB    0               ;END OF BLOCK
AVAL    DEFW    0               ;ADDR VALUE GOES HERE
FVAL    DEFW    0               ;FIND VALUE GOES HERE
CVAL    DEFW    0               ;CHANGE VALUE GOES HERE
```

This is an example of the @PARAM SVC using the 'type-2' parameter block:

```
;
;       FETCH ADDR, FIND, & CHANGE
;       PARAMETERS FROM COMMAND LINE
;
;       ENTRY:
;       HL=>COMMAND LINE
;
GTPRM   LD    DE,PBLK         ;DE=>PARAMETER BLOCK
        LD    A,17            ;DO @PARAM
        RST   28H
        RET   Z               ;DONE IF NO ERROR
        LD    HL,ERMES        ;DISPLAY AN ERROR
        LD    A,10            ;MESSAGE IF @PARAM
        RST   28H             ;FAILED
        OR    -1              ;RETURN NZ STATUS
        RET
ERMES   DEFM  'PARAMETER ERROR'
        DEFB  13
PBLK    DEFB  80H             ;FLAG TYPE-2 PARAM BLOCK
        DEFB  84H             ;NUMERIC VALUE, 4-CHR NAME
        DEFM  'ADDR'          ;PARAMETER NAME
        DEFB  0               ;RESPONSE FLAGS
        DEFW  AVAL            ;POINTER TO ADDR VALUE
        DEFB  24H             ;STRING VALUE, 4-CHR NAME
        DEFM  'FIND'          ;PARAMETER NAME
        DEFB  0               ;RESPONSE FLAGS
        DEFW  FVAL            ;POINTER TO FIND VALUE
        DEFB  26H             ;STRING VALUE, 6-CHR NAME
        DEFM  'CHANGE'        ;PARAMETER NAME
        DEFB  0               ;RESPONSE FLAGS
        DEFW  CVAL            ;POINTER TO CHANGE VALUE
        DEFB  0               ;END OF BLOCK
AVAL    DEFW  0               ;PLACE ADDR VALUE HERE
FVAL    DEFW  0               ;PLACE FIND VALUE HERE
CVAL    DEFW  0               ;PLACE CHANGE VALUE HERE
```

@FSPEC
SVC: 78 Dec/4E Hex

The @FSPEC SVC is used to move a file or device specification from one area of RAM (typically an input buffer) into an FCB. The file or device specification is automatically converted into upper case during the transfer. @FSPEC moves the specification character by character until either a terminating character (a space, comma, semicolon, or control character) is found, or an invalid file/device specification character is encountered. If an invalid character is found, @FSPEC will terminate with NZ status, indicating an error. Note that @FSPEC does not return an error code. @FSPEC will parse over, or ignore, leading spaces and commas as well as the keywords FROM, TO, & USING.

ENTRY: A = 78 Dec/4E Hex
DE => 32-byte FCB
HL => Buffer containing file or device specification

EXIT: If ZF, function successful
HL => Terminating character
If NZ, specification contains an invalid character
A = Invalid character
HL => Invalid character

AF & BC are altered

@EVAL
SVC: 124 Dec/7C Hex

This SVC is the DOSPLUS command evaluator. This routine scans a command line for the source (FROM), destination (TO), wildmask (USING), and parameter fields, placing the value of each in user-specified regions of RAM.

Normally, @EVAL assigns the fields in the order FROM, TO, USING as it scans the command line from left to right. Therefore, the line:

```
COPY :1 :0 /TXT (ECHO)
```

would be evaluated with ":1" as the source field, ":0" as the destination, and "/TXT" as the wildmask. The parameter field is always signalled by a comma or a left paranthesis. The order in which the fields are placed on the command line may be modified by the use of the FROM, TO, or USING delimiters, or by the use of wildcard characters. For instance, the line:

```
COPY TO :0 USING /TXT FROM :1 (ECHO)
```

is evaluated identically to the first example, since the FROM, TO, and USING delimiters instructed @EVAL which fields were which. Likewise, any field containing a wildcard character is assumed to be the wildmask, or USING, field. The line:

```
COPY !/TXT:1 :0 (ECHO)
```

is evaluated the same as the first two examples. When @EVAL encounters the "!/TXT", it immediately places it into the wildmask field, since it contains a wildcard character. @EVAL then continues with its normal order, placing ":1" into the source field and ":0" into the destination field.

In order to perform its function, @EVAL requires a block of data that instructs it where to place the data from the various fields. This is called the evaluation block, and it is a 9-byte area of RAM containing a 1-byte flag, and four 2-byte pointers arranged as follows:

Byte	Contents	
EVBLK+00H	Flag byte	
	Bit 3:	Parameter field filled
	Bit 2:	Wildmask field filled (USING)
	Bit 1:	Destination field filled (TO)
	Bit 0:	Source field filled (FROM)
EVBLK+01H	Source DCB pointer	
EVBLK+03H	Destination DCB pointer	
EVBLK+05H	Wildmask DCB pointer	
EVBLK+07H	Parameter block pointer	

After executing @EVAL, the flag byte contains four flags that indicate which fields were detected on the command line and moved into the appropriate DCB, or parameter value address in the case of parameters.

The source, destination, and wildmask DCBs are 33-byte regions of RAM which consist of a 1-byte flag followed by a 32-byte DCB. The flag byte contains the following information:

Bit 7:	Devicespec in field
Bit 6:	Filespec in field
Bit 5:	Filespec contains wildcard characters
Bit 4:	Device field contains drivespec
Bits 0-3:	Device number

After executing @EVAL, the contents of the fields are placed in their respective DCBs, and the flag byte can be used to detect what type of information is in each DCB. Bit 7, when set, indicates that the DCB contains a device specification. The device specification may be the name of a character-oriented device, such as @PR, or a disk drive device, such as :1. If it is a disk drive device name, it may be contained with a filespec, such as FILE/DAT:2.

Bit 6 indicates that the DCB contains a file specification, and bit 5 is set is the filespec contains wildcard characters.

Bit 4 is set if the device specification flagged with bit 7 belongs to a disk drive device.

Bits 0-3 contain the logical device number of any device contained in the DCB. This device number may be used in conjunction with the @LOCDCB and @LOCDCT SVCs detailed elsewhere in this manual.

The parameter block used for @EVAL is similar to the 'type-1' parameter block used by @PARAM. The @EVAL parameter block consists of one or more parameter entries, each of which are nine bytes in length, as illustrated below:

Byte	Contents
PBLK+00H	6-byte parameter name
PBLK+06H	2-byte pointer
PBLK+08H	1-byte type specifier

The last entry in the parameter block is followed by a 00H byte to signify the end of the block. The first eight bytes are identical to those used by @PARAM (type-1 parameter block), the first six bytes containing a left-justified parameter name, padded with blanks, and the next two bytes containing a pointer to a location in RAM in which to place the value of the parameter. The @EVAL parameter block has an additional byte, after the parameter value address pointer, which specifies to @EVAL what type of values are acceptable for each parameter.

This byte is called the type specifier, and it contains three flags as shown below:

Bit 7:	String value
Bit 6:	Numeric value
Bit 5:	Logical value

String values must be enclosed in either single or double quotation marks, and the beginning address of the string value is placed in the parameter value address specified in the parameter block. Numeric values may be given in decimal or hexadecimal and may cover the range 0-65535 decimal or 0000H-FFFFH. The value is placed in the parameter value address. Logical values may be specified as "YES or ON" (TRUE) or as "NO or OFF" (FALSE). If a logical parameter is given without a logical value, as in "DIR :1 (INV)", the parameter value is assumed to be TRUE. A TRUE value is represented in the parameter value address a an FFFFH, and a FALSE value as 0000H.

If a command line attempts to set a parameter to a type not specified in the type specification byte, @EVAL will return an error.

Note that if it is not desired to use a parameter block in conjunction with @EVAL, the parameter block pointer in the evaluation block must point to a 00H byte.

ENTRY: A = 124 Dec/7C Hex
HL => Command line text, terminated with ASCII 03H, 0DH, or semicolon.
IX => Evaluation block

EXIT: If ZF, function successful
If NZ, A=Error code
HL => Invalid character

Disk I/O Functions

This group of thirteen SVCs gives the user the ability to perform direct disk I/O; that is, any given cylinder and sector may be written, read, or formatted. The SVCs are:

@CKDRV	@WRSSC
@SEEK	@WRTRK
@RDHDR	@RDSSC
@RDSEC	@DIRRD
@VRSEC	@DIRWR
@RDTRK	@DISKIO
@WRSEC	

@CKDRV

SVC: 33 Dec/21 Hex

This SVC is used to determine if a disk drive is ready to perform I/O. It will also return the write-protect status of the drive.

ENTRY: A = 33 Dec/21 Hex
C = Logical drive number {00-0FH}

EXIT: If ZF, drive is ready for I/O
If CF, drive is write-protected
If NZ, drive is not ready

@SEEK

SVC: 46 Dec/2E Hex

@SEEK is used to position the disk drive's read/write head over a specific cylinder on a formatted diskette. Note that the @SEEK routine does not return a specific error code. @CKDRV should be used prior to @SEEK to insure that the drive is ready to perform a seek.

ENTRY: A = 46 Dec/2E Hex
C = Logical drive number {00-0FH}
D = Cylinder number
E = Sector number

EXIT: If ZF, function successful
If NZ, seek error has occured

AF is altered

@RDHDR
SVC: 48 Dec/30 Hex

This SVC is included for purposes of compatibility with TRSDOS 6.0 As in TRSDOS 6.0, it merely performs a @RDSEC, reading a sector of data from disk into RAM.

ENTRY: A = 48 Dec/30 Hex
C = Logical drive number (00-0FH)
D = Cylinder number
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@RDSEC
SVC: 49 Dec/31 Hex

The @RDSEC SVC is used to read a single sector of data from disk and place it in a user-specified RAM buffer.

ENTRY: A = 49 Dec/31 Hex
C = Logical drive number (00-0FH)
D = Cylinder number
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

@VRSEC
SVC: 50 Dec/32 Hex

This SVC verifies the integrity of the data stored on a specified cylinder and sector; in effect, it performs a read without transferring data into a user buffer.

ENTRY: A = 50 Dec/32 Hex
C = Logical drive number {00-0FH}
D = Cylinder number
E = Sector number

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

==========

@RDTRK
SVC: 51 Dec/33 Hex

This SVC is provided for compatibility with TRSDOS 6.0. As in TRSDOS 6.0, @RDTRK simply performs a @RDSEC.

ENTRY: A = 51 Dec/33 Hex
C = Logical drive number {00-0FH}
D = Cylinder number
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

==========

@WRSEC
SVC: 53 Dec/35 Hex

This SVC writes the contents of a 256-byte RAM buffer to a disk sector.

ENTRY: A = 53 Dec/35 Hex
C = Logical Drive number {00-0FH}
D = Cylinder number
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

==========

@WRSSC
SVC: 54 Dec/36 Hex

@WRSSC is used to write the contents of a 256-byte buffer to a specified sector on the system, or directory, cylinder. No cylinder number need be specified for this SVC, as the directory cylinder is automatically obtained by @WRSSC.

ENTRY: A = 54 Dec/36 Hex
C = Logical drive number {00-0FH}
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@WRTRK
SVC: 55 Dec/37 Hex

This SVC is used to format a single track on a diskette. The disk driver program must support the DFORMT function code described under @DISKIO in order for this SVC to function.

ENTRY: A = 55 Dec/37 Hex
C = Logical drive number {00-0FH}
D = Cylinder number
HL => Format data buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@RDSSC
SVC: 85 Dec/55 Hex

This SVC is used to read a system, or directory, sector into RAM. It is not necessary to specify a cylinder number upon entry to @RDSSC, as the SVC automatically obtains the cylinder number upon execution.

ENTRY: A = 85 Dec/55 Hex
C = Logical drive number {00-0FH}
E = Sector number
HL => 256-byte sector buffer

EXIT: If ZF, function successful
If NZ, A=Error code

AF is altered

@DIRRD
SVC: 87 Dec/57 Hex

The @DIRRD SVC will read the directory sector containing a given directory entry, specified by its logical file number, into the system buffer and position the HL register pair to the first byte of the requested directory entry.

ENTRY: A = 87 Dec/57 Hex
B = Logical file number of directoy entry to read
C = Logical drive number {00-0FH}

EXIT: If ZF, function successful
HL => Directory entry
If NZ, A=Error code

AF is altered

==========

@DIRWR
SVC: 88 Dec/58 Hex

@DIRWR is used to write the contents of the system buffer to a diskette directory sector determined by a user-specified logical file number. Note that since this SVC writes an entire directory sector to disk, rather than a single directory entry, it is good practice to perform a @DIRRD before calling @DIRWR in order to insure that the proper directory sector resides in the system buffer.

ENTRY: A = 88 Dec/58 Hex
B = Logical file number
C = Logical drive number {00-0FH}

EXIT: If ZF, function successful
HL => Directory entry
If NZ, A=Error code

AF is altered

==========

==========

@DISKIO
SVC: 121 Dec/79 Hex

The @DISKIO SVC provides ten basic disk I/O functions, listed below:

Function Code	Function Name	Function Description
0	DCHECK	Check for drive ready
1	DHOME	Home & initialize drive
2	DSEEK	Position read/write head over cylinder
3	DREAD	Read a sector
4	DVERF	Verify a sector
5	DWRITE	Write a sector
6	SREAD	Read a directory sector
7	SWRITE	Write a directory sector
8	DWRITA	Write a system sector
9	DFORMT	Format a track/cylinder

All @DISKIO functions require the same entry information and provide the same exit conditions, given below.

ENTRY: A = 121 Dec/79 Hex
B = Function code {0-9}
C = Logical drive number {00-0FH}
D = Cylinder number
E = Sector number
HL => 256-byte buffer

EXIT: If ZF, function successful
If NZ, A=Error code

With all of the ten @DISKIO functions, the proper function code, taken from the table above, is loaded into the B register.

The C register should contain the disk drive device number (the number returned by the @LOCDEV and @EVAL routines)

The D register contains the cylinder number. In the case of partitioned rigid drives, this cylinder number is the offset from the beginning cylinder of the volume - it is not the actual physical cylinder number.

The sector number is contained in the E register. The sector number is an offset from the beginning sector number on the cylinder. The disk device driver must add the sector offset stored in the drive's DCT to obtain the actual physical sector number.

The HL register pair must point to a 256-byte disk I/O buffer. Any data to be written to disk must be placed in this buffer, and all data read from the disk will appear here.

DCHECK

The DCHECK function is used to test the disk drive to determine its state of readiness. For this function, only the function code and drive device number need be specified. Upon return from DCHECK, ZF status indicates that the drive is ready for I/O, and NZ if the drive is not ready. CF status means that the drive is write-protected.

DHOME

The DHOME function causes the drive to home itself, or bring the drive's read/write head into position over logical cylinder 0. Like DCHECK, DHOME requires only the function code and drive device number to be specified upon entry to the routine.

DSEEK

DSEEK is used to position a drive's read/write head over a specified logical cylinder number. The function code, device number, and of course cylinder number must be provided upon entry to DSEEK.

DREAD

This function is used to read a specified sector from disk. The sector data will be placed in the disk I/O buffer pointed to by the HL register pair.

DVERF

This function is similar to DREAD, above, in that it will read a sector from disk into a 256-byte buffer specified by HL. The difference lies in the fact that if an error is encountered during DVERF, the I/O driver does not -re-try-, or re-read the sector. Rather, it immediately aborts and reports an error.

DWRITE

DWRITE is used to write a sector to diskette. The data is taken from the 256-byte disk I/O buffer pointed to be the HL register pair upon entry to DWRITE.

SREAD

The SREAD function will read a single sector from a diskette's directory cylinder. Only the function code, device number, and sector number need be specified for SREAD, as it will automatically locate the directory cylinder. The data read from the directory sector will be placed in the disk I/O buffer indicated by the HL register pair.

SWRITE

The SWRITE function is used to write a sector to the diskette's directory. Only the function code, device number, and sector number need be specified upon entry to SWRITE, as the function will locate the diskette's directory automatically. The data written to the directory sector is taken from the disk I/O buffer pointed to by HL.

DWRTEA

The DWRTEA function is used to write a sector to any specified cylinder and sector address, and in this fashion it is similar to the DWRITE function described above. The difference between the two functions lies in the fact the DWRTEA writes a protected or locked sector reserved for use by the directory. This is typically used by disk formatter program when creating a diskette directory.

DFORMT

The DFORMT function is used to format any specified track or cylinder on a drive. Whether a single track or an entire cylinder (in the case of multi-surface diskettes/rigid drives) is formatted is a function of the disk drive device driver. The standard floppy driver supplied with DOSPLUS IV formats a single track.

Miscellaneous Functions

Eight SVCs remain that do not fit neatly into any category save 'miscellaneous'. They are:

@WHERE	@PAUSE
@LOGER	@SOUND
@LOGOT	@WILD
@SORT	@VDCTL

@WHERE

SVC: 7 Dec/07 Hex

This SVC simply returns the address of the byte following the call to @WHERE. It is useful in relocatable modules which need to determine where they reside in RAM.

ENTRY: A = 7 Dec/07 Hex

EXIT: HL => Instruction following the RST 28H which invoked @WHERE

@LOGER

SVC: 11 Dec/0B Hex

This routine is provided for purposes of compatibility with TRSDOS 6.0. On TRSDOS, its purpose is to send a message to a special device called *JL, or 'job log'. This device does not exist on the current release of DOSPLUS IV. The @LOGER SVC therefore returns with ZF status, indicating no error; however, no action is taken.

ENTRY: A = 11 Dec/0B Hex

EXIT: ZF indicating no error

@LOGOT

SVC: 12 Dec/0C Hex

This routine is provided for purposes of compatibility with TRSDOS 6.0. On TRSDOS, its purpose is to send a message both to the display and to a special device called *JL, or 'job log'. This device does not exist on the current release of DOSPLUS IV. The @LOGOT SVC therefore displays a message on the CRT and returns. Since no *JL device exists to output a message to, @LOGOT is equivalent to @DSPLY.

ENTRY: A = 11 Dec/0B Hex
HL => Message text, terminated with a carriage return, 0DH

EXIT: If ZF, function successful
If NZ, A=Error code

@VDCTL
SVC: 15 Dec/0F Hex

The @VDCTL SVC provides eight functions related to the video display. The function desired is passed to @VDCTL as a function code contained in the B register. The following table lists each function code and its action:

Function code	Action taken
1	Returns the character at specified row, column position on display
2	Display character at specified row, column position on display
3	Position cursor to specified row, column position on display
4	Return the current row, column position of the cursor
5	Move a 2K block of RAM onto the video display
6	Move 2K of data from the video display to a RAM buffer
7	Scroll protect n lines at top of display (0 < n < 8)
8	Set cursor character

As the table suggests, many of @VDCTL's functions involve a screen position. Positions on the video display are specified in a (row,column) format. The Model 4's video display is 80 characters wide and 24 lines long. Therefore, the rows are numbered from 0-23 (from top to bottom) and the columns from 0-79 (from left to right).

Get character from specified position

ENTRY: A = 15 Dec/0F Hex
B = 1
H = Row position {0-23}
L = Column position {0-79}

EXIT: If ZF, function successful
A = Character at screen position
Note that characters displayed in reverse video have bit 7 set
If NZ, A=Error code

Display character at specified position

ENTRY: A = 15 Dec/0F Hex
B = 2
C = Character to display
H = Row position {0-23}
L = Column position {0-79}

EXIT: If ZF, function successful
If NZ, A=Error code

Position cursor to specified position

ENTRY: A = 15 Dec/0F Hex
B = 3
H = Row position {0-23}
L = Column position {0-79}

EXIT: If ZF, function successful
If NZ, A=Error code

Fetch current cursor position

ENTRY: A = 15 Dec/0F Hex
B = 4

EXIT: H = Row position {0-23}
L = Column position {0-79}

Move contents of RAM buffer to video display

ENTRY: A = 15 Dec/0F Hex
B = 5
HL => 2048-byte buffer containing data to be placed in video memory
Note that this buffer must reside entirely below F400H

EXIT: HL => Byte following last byte of buffer

Move contents of video display to RAM buffer

ENTRY: A = 15 Dec/0F Hex
B = 6
HL => 2048-byte buffer to receive data from video display.
Note that although only 1920 characters are displayed on the CRT, 2048 bytes of data are returned by this SVC. This buffer must reside entirely below F400H.

EXIT: Buffer contains video data. Note that characters displayed in reverse video will have bit 7 set

Scroll protect video display

ENTRY: A = 15 Dec/0F Hex
B = 7
C = Number of lines at top of display to scroll protect, or 0 to disable scroll protection. From 1 to 7 lines may be scroll protected.

EXIT: n lines at top of video are scroll-protected

Set cursor character

ENTRY: A = 15 Dec/0F Hex
B = 8
C = Cursor character

EXIT: DE is altered

@PAUSE
SVC: 16 Dec/10 Hex

This SVC is used to produce a delay of user-specified length. The length is detemined by the value in the BC register pair. The routine will stay in a loop consuming time until the desired delay has been acheived. The value needed to produce the desired delay may be calculated from:

DELAY COUNT = (S/1.48) * 100,000

where 'S' is the desired delay in seconds. Each delay count produces a pause of about 14.8 microseconds duration.

ENTRY: A = 16 Dec/10 Hex
BC = Delay count

EXIT: AF is altered

@SOUND
SVC: 104 Dec/68 Hex

This SVC produces a tone using the built-in 'beeper' in the Model 4. The user may specify both tone and duration. The duration may take on a value of 0 through 31, with 0 being the shortest duration and 31 the longest. The tone may have a value of 0 thorgh 7, with 0 producing the highest tone and 7 the lowest.

ENTRY: A = 104 Dec/68 Hex
B = Tone & duration
Bits 3-7: Duration {0-31}
Bits 0-2: Tone {0-7}

EXIT: AF is altered

@WILD
SVC: 125 Dec/7D Hex

@WILD is used to compare a file specification to a wildmask and return a status flag which indicates whether the filespec matches the wildmask. @WILD actually performs two functions: The first function sets the wildmask to be used in any following comparisons. The wildmask should be terminated with an ETX or carriage return. The second function compares the stored wildmask with a filename and extension. The filename to which the wildmask is compared must be 8 characters in length, padded with spaces on the right if necessary, and followed immediately by a 3 character extension, also padded with spaces if necessary.

Set a wildmask

ENTRY: A = 125 Dec/7D Hex
B = 1
HL => Wildmask to set, terminated with CR or ETX

EXIT: If ZF, function successful
If NZ, invalid wildmask

Compare filespec to wildmask

ENTRY: A = 125 Dec/7D Hex
B = 0
HL => Filename and extension for comparison with wildmask

EXIT: If ZF, filespec matches wildmask
If NZ, filespec does not match wildmask

SORT@
SVC: 127 Dec/7F Hex

This SVC will sort a block of memory composed of any number of entries of user-defined length. All entries in the list to be sorted must be of the same length. The key upon which the sort is performed may be in any position within the entry, and may be of any length up to and including the entire length of the entry.

ENTRY:
B = Offset from beginning of entry to sort key
C = Length of list entries
DE => 1st byte of last entry in list
H = Sort switch.
If H=0, Ascending sort
If H≠0, Descending sort
L = Length of sort key
IX => 1st byte of first entry in list

EXIT:
If ZF, function successful
If NZ, A=Error code
@SORT Error codes:
1: Key offset+key length > entry length
2: Key offset > entry length
3: Entry length = 0
4: Key length = 0
5: Last entry pointer (DE) incorrect

X. - Writing Drivers and Filters for DOSPLUS IV

DOSPLUS IV allows the user to write device drivers, or programs that interface with the operating system to control external I/O equipment, such as disk drives, printers, card readers, etc. This section of the manual contains all of the information a programmer needs to know about interfacing a device driver to DOSPLUS IV.

Writing device drivers for DOSPLUS IV is a simple task for any programmer familiar with Z-80 assembly language programming. DOSPLUS itself performs much of the work for the programmer, providing filter tables, a complete FORCE and JOIN capability within the system's character I/O routines, a flexible and powerful DCT and DCB structure, etc.

All drivers are installed in RAM using the DOSPLUS command ASSIGN. The general form of the command is:

ASSIGN (FROM) @ds/:dr (TO) filespec (param=exp,param=exp, . . .)

where "@ds/:dr" is a device specification either one of the eight character-oriented devices (@KI, @DO, @PR, @RS, @U1, @U2, @SI, @SO) or one of the sixteen disk drive devices (:0 through :15), and "filespec" is the file specification belonging to a driver program for the particular device. The ASSIGN command loads the device driver program into RAM and transfers control to it. Upon entry to the driver program, the following registers contain important and useful information:

ENTRY:
- DE=> Highest available address in user RAM
- HL=> Next field in DOSPLUS command line following driver name
- IX=> DCB or DCT for device
- IY=> DCBTBL or DCTTBL entry for device

It is now the responsibility of the driver program to relocate itself into high memory and to adjust the system's high memory pointer (via @HIGH$) in order to protect itself. The driver should also insert its entry point into the device's DCB or DCT at DCB+01H & DCB+02H or DCT+01H & DCT+02H, as well as initializing any DCT or DCB data that the driver may require.

If a device does not have a DCB or a DCT defined at the time of ASSIGN, DOSPLUS will create a 20-byte DCB/DCT in system (low) RAM for the device. This address will be present in the IX register when ASSIGN passes control to your program.

If a device does have a currently existing DCB or DCT, the driver program is free to relocate that DCB/DCT to any other region of RAM by simply modifying the DCB/DCT pointer address at IY+00 and IY+01 upon entry to the driver program.

The driver program may be passed parameters on the DOSPLUS command line which it may pick up by use of @FSPEC and @PARAM or by @EVAL.

Disk Drivers

DOSPLUS IV disk drivers must support the ten functions explained in section VIII under the SVC @DISKIO. To recap, these functions are:

Function Code	Function Name	Function Description
0	DCHECK	Check for drive ready
1	DHOME	Home & initialize drive
2	DSEEK	Position read/write head over cylinder
3	DREAD	Read a sector
4	DVERF	Verify a sector
5	DWRITE	Write a sector
6	SREAD	Read a directory sector
7	SWRITE	Write a directory sector
8	DWRITA	Write a system sector
9	DFORMT	Format a track/cylinder

When the operating system transfers control to the disk driver, the Z-80 registers contain the following information:

ENTRY:
B = Function code {0-9}
C = Device number {00-0FH}
D = Cylinder number
E = Sector number
HL => 256-byte disk I/O buffer
IY => Drive control table

From this information, the driver must perform the function requested by the calling program (consult section VIII, @DISKIO for descriptions of each function). The driver may make use of any registers necessary, as the operating system saves the contents of all registers before calling the driver and restores them afterward. The driver should return an error code in the A register if an error occurs. The driver must set the Z flag if no error occurs, and set NZ status if an error was encountered.

A sample disk drive device driver is reproduced in this manual, and it serves as a good model of driver structure and execution.

Character-oriented Device Drivers

Drivers for the eight character-oriented devices are typically much simpler than disk drive device drivers. Such drivers need only support one, two, or all three of the basic character I/O functions performed by CIO: Input, Output, and Control I/O.

After CIO transfers program control to the driver, the Z-80 registers contain the following data:

ENTRY: B = I/O type
Bit 2: Control I/O
Bit 1: Output
Bit 0: Input
C = Character for output
IX => Device control block
If NC, Output
If CF, Input
If NZ, Control

Drivers may use any registers required, as the CIO system saves all registers before entering the driver and restores them upon return from the driver. The driver should return any characters read from the device in the A register.

Note that in the case of character input from a device, it is good practice to return a status flag in order to inform the calling program whether a character was available at the device. Typically, drivers set NZ status with a 00H in the accumulator if no character was available, and set ZF status if a character was fetched. Many programs make the assumption that the driver does return a useable status flag, and therefore the practice is advisable.

The character I/O system provided by DOSPLUS IV is responsible for performing filtering, or translation of character values. Device drivers therefore need not be concerned with character translation.

A sample RS232 serial driver is included in this manual to provide an example of character-I/O device driver structure and execution.

Filter Programs

A filter program is a program which simply accepts input (from the keyboard, or from @SI during pipe operations), modifies it in some manner, and then outputs the data (to the display, or to @SO during pipe operations).

Writing a filter program for DOSPLUS IV is a straightforward proposition. During pipe operations, the DOS automatically performs special device linking between @KI and @SI, and @DO and @SO. Therefore, the program simply fetches its input from the @KI device (using @KEY or @KBD), and directs output to the @DO device (using @DSP or @DSPLY).

As an example of filter program implementation, a listing of the MORE/CMD filter may be found in the rear of this manual.

XI. - DOSPLUS IV Error Codes

The following is a list of the error codes generated by DOSPLUS IV and the associated error messages.

Dec	Hex	Error Message
0	00	No error found
1	01	CRC error during header read
2	02	Seek error during read
3	03	Lost data during read
4	04	CRC error during read
5	05	Data record not found during read
6	06	Attempted to read system data record
7	07	Attempted to read locked/deleted data record
8	08	Device not available
9	09	CRC error during header write
10	0A	Seek error during write
11	0B	Lost data during write
12	0C	CRC error during write
13	0D	Data record not found during write
14	0E	Write fault on disk drive
15	0F	Write protected disk
16	10	Illegal logical file number
17	11	Directory read error
18	12	Directory write error
19	13	Improper file name
20	14	GAT read error
21	15	GAT write error
22	16	HIT read error
23	17	HIT write error
24	18	File not in directory
25	19	File access DENIED due to password protection
26	1A	FULL or write protected disk
27	1B	Disk space FULL
28	1C	Attempted to read past EOF
29	1D	Attempted to read outside of file limits
30	1E	Directory FULL - can't extend file

Dec	Hex	Error Message
31	1F	Program not found
32	20	Improper drive specification
33	21	No device space available
34	22	Attempted to use non program file as a program
35	23	Memory fault during program load
36	24	Attempted to load read only memory
37	25	Illegal access attempted to protected file
38	26	File not open
39	27	Device in use
40	28	Protected system device
41	29	File already open
42	2A	LRL open fault
43	2B	SVC parameter error
44	2C	No memory space available
45	2D	Illegal device specification
46	2E	Illegal file specification
47	2F	Specification field missing
48	30	Invalid parameter or specification
49	31	Invalid data provided
50	32	File already exists
51	33	Device already exists
52	34	Terminated

XII. - Keyboard Characters and Codes

The DOSPLUS IV keyboard driver allows the user to generate virtually any 8-bit code desired. The table below lists the key combinations necessary to generate each code:

Key	Code	+ SHIFT	+ CTRL	+ CLEAR	+SHIFT CLEAR	+CTRL CLEAR
<SPACE>	20H	20H	20H	A0H	A0H	A0H
!	21H	21H	1CH	A1H	A1H	9CH
"	22H	22H	1BH	A2H	A2H	9BH
#	23H	23H	1EH	A3H	A3H	9EH
$	24H	24H	1FH	A4H	A4H	9FH
%	25H	25H	7FH	A5H	A5H	FFH
&	26H	26H	5EH	A6H	A6H	DEH
'	27H	27H	7EH	A7H	A7H	FEH
(	28H	28H	5BH	A8H	A8H	DBH
)	29H	29H	5DH	A9H	A9H	DDH
:	3AH	—	—	BAH	—	—
*	2AH	2AH	5BH	AAH	AAH	DBH
-	2DH	—	5FH	ADH	—	DFH
=	3DH	3DH	5FH	BDH	BDH	DFH
BREAK	80H	80H	80H	80H	80H	80H
F1	81H	91H	81H	81H	91H	81H
F2	82H	92H	82H	82H	92H	82H
F3	83H	93H	83H	83H	93H	83H
UP ARROW	0BH	1BH	0BH	8BH	9BH	8BH
DOWN ARROW	0AH	1AH	0AH	8AH	9AH	8AH
LEFT ARROW	08H	18H	08H	88H	98H	88H
RIGHT ARROW	09H	19H	09H	89H	99H	89H
ENTER	0DH	0DH	0DH	8DH	8DH	8DH
@	40H	60H	00H	C0H	E0H	00H
;	3BH	—	7CH	BBH	—	FCH
+	2BH	2BH	7CH	ABH	ABH	FCH
,	2CH	—	7BH	ACH	—	FBH
<	3CH	3CH	7BH	BCH	BCH	FBH
.	2EH	—	7DH	AEH	—	FDH
>	3DH	3DH	7DH	BDH	BDH	FDH
/	2FH	—	5CH	AFH	—	DCH
?	3FH	3FH	5CH	BFH	BFH	DCH
A	61H	41H	01H	E1H	C1H	81H
B	62H	42H	02H	E2H	C2H	82H
C	63H	43H	03H	E3H	C3H	83H
D	64H	44H	04H	E4H	C4H	84H
E	65H	45H	05H	E5H	C5H	85H
F	66H	46H	06H	E6H	C6H	86H
G	67H	47H	07H	E7H	C7H	87H
H	68H	48H	08H	E8H	C8H	88H
I	69H	49H	09H	E9H	C9H	89H
J	6AH	4AH	0AH	EAH	CAH	8AH

Key	Code	+ SHIFT	+ CTRL	+ CLEAR	+SHIFT CLEAR	+CTRL CLEAR
<SPACE>	20H	20H	20H	A0H	A0H	A0H
K	6BH	4BH	0BH	EBH	CBH	8BH
L	6CH	4CH	0CH	ECH	CCH	8CH
M	6DH	4DH	0DH	EDH	CDH	8DH
N	6EH	4EH	0EH	EEH	CEH	8EH
O	6FH	4FH	0FH	EFH	CFH	8FH
P	70H	50H	10H	F0H	D0H	90H
Q	71H	51H	11H	F1H	D1H	91H
R	72H	52H	12H	F2H	D2H	92H
S	73H	53H	13H	F3H	D3H	93H
T	74H	54H	14H	F4H	D4H	94H
U	75H	55H	15H	F5H	D5H	95H
V	76H	56H	16H	F6H	D6H	96H
W	77H	57H	17H	F7H	D7H	97H
X	78H	58H	18H	F8H	D8H	98H
Y	79H	59H	19H	F9H	D9H	99H
Z	7AH	5AH	1AH	FAH	DAH	9AH

Video Display codes

Code	Character
00H	Display next character. Used to display special characters codes 01H-1FH.
07H	Sound a short tone (BEL)
08H	Backspace. Move cursor to previous video position and erase character.
09H	Horizontal tab. Move cursor to next 8-character boundary.
0AH	Linefeed. Move cursor to first character of next video line.
0DH	Carriage Return. Move cursor to start of next video line.
0EH	Turn cursor on.
0FH	Turn cursor off.
10H	Begin inverse video mode.
11H	End inverse video mode.
15H	Toggle space compression/special character code sets.
16H	Toggle special character/alternate character sets.
17H	Set 40 chr/line mode.
18H	Move cursor to previous video position without erasing character.
19H	Move cursor to next video position without erasing character.
1AH	Move cursor to next video row, same column, without erasing character.
1BH	Move cursor to previous video row, same column, do not erase character.
1CH	Home cursor, set 80-column, normal video mode.
1DH	Clear current video line.
1EH	Clear from current cursor position to end of line.
1FH	Clear from current cursor position to end of screen.

XII. - Technical Glossary

Alternate System Driver

The alternate system driver is a disk drive or device driver program which is stored as part of the system disk's bootstrap program, and it is stored beginning on sector 3 of the BOOT/SYS program. It is the responsibility of the bootstrap program to load the alternate system driver into RAM. After initialization, DOSPLUS will transfer control to the driver, which may then install itself into DOSPLUS IV.

Boot

(1) To reset, or restart the computer, resulting in the operating system being re-loaded from diskette. (2) Abbreviation for bootstrap; refers to the file BOOT/SYS.

Blocked Records

Logical records whose length is less than the length of a physical sector. Under DOSPLUS IV, two or more such records are placed into a single physical record on disk.

Blocking Buffer

A 256-byte area of RAM which is used by DOSPLUS to manipulate the data within a physical record during reads and writes from and to a disk file.

Buffer

A broad term which refers to any area of memory used to hold meaningful data.

Cylinder

An artificial diskette structure used by DOSPLUS IV to describe, access, and partition disk drives. A cylinder consists of one or more diskette tracks on a single disk drive over which the drive's read/write heads may be simultaneously positioned.

Data Disk

A diskette formatted by DOSPLUS which does not contain the DOSPLUS operating system, suitable for program and data storage but unable to act as a system diskette.

DCB

Abbreviation for Device Control Block. An area of RAM whose purpose is to store important information concerning a DOSPLUS character-oriented device, including I/O type and driver location.

DCT

Abbreviation for Drive Control Table. An area of RAM whose purpose is to store important information concerning DOSPLUS disk drive devices, including driver address.

Device

A broad term which usually refers to some external peripheral I/O unit, such as a lineprinter or a disk drive. Under DOSPLUS IV, two general classifications of devices exist: (1) Disk drive devices, and (2) Character-oriented devices. Disk drive devices are concerned with reading and writing physical records (length>1 byte) from and to some peripheral. Character-oriented devices may accept or provide a single byte of data at a time.

EOF

Abbreviation for End-Of-File. (1) Refers to the byte in a file's primary directory entry or FCB which specifies how many bytes a file extends into its final physical record. (2) The last byte in a file; The PEOF@ routine positions to EOF.

ERN

Abbreviation for Ending Record Number. This number, found in a file's primary directory entry or FCB, is the number of the final physical record in the file. Records are numbered starting with 0.

FCB

Abbreviation for File Control Block. An area of memory which contains important information for file I/O. Before OPENing a file, the FCB contains the file specification. After the OPEN and before CLOSEing the file, the FCB contains information concerning the file's current record position and other data.

GAT

Abbreviation for Granule Allocation Table. The first sector of the file DIR/SYS, which contains information about the used and unused areas of a diskette, and other miscellaneous data.

Granule

An artificial unit of storage, some multiple of 1 physical record in length. A granule is the smallest unit of diskette space which the DOS may allocate to a file. Often abbreviated to gran.

Hash Code

A 1-byte value calculated from a file specification by a hashing algorithm. Used by the operating system to quickly locate files in a diskette directory.

Hashing

The process of converting a key field (such as a disk file specification) into a numeric value by performing a series of operations, known as a hashing algorithm, upon the key.

HIT

Abbreviation for Hash Index Table. Contained in the second sector of the file DIR/SYS, the HIT is used to store the hash codes calculated for each filename present in the directory. The position of the hash code within the HIT corresponds to the location of the file's primary directory entry.

LFN

Abbreviation for Logical File Number. The LFN is a single-byte value which indicates the directory sector number and offset from the beginning of the sector in which a file directory entry may be found.

Load Module Format

A special file format created by the DOSPLUS DUMP command and most TRS-80 assembler programs. Load module format files contain information instructing the operating system where in RAM file data should be placed.

Library

In reference to DOSPLUS, the set of 39 commands intrinsic to the DOSPLUS operating system; Library commands as opposed to utility programs.

Lock-out Table

This table, located on the first sector of the DIR/SYS file, contains information concerning which granules on a diskette are useable and which are unusable, or locked-out.

Log

To "log in a diskette"; refers to the process in which DOSPLUS IV determines the location of the directory cylinder, density, surface count, and other information pertaining to a diskette.

Logical Record

A contiguous block of data read from or written to a file, usually representing some meaningful information. Under DOSPLUS IV, a logical record may be of a different length than a physical record.

Logical Record Length

(1) Refers to the number of bytes contained in a logical record. (2) Refers to the byte in a file's primary directory entry or FCB which specifies the logical record length with which the file was originally created or OPENed, respectively. Often abbreviated LRL.

LSB

Abbreviation for Least Significant Byte. In a 16-bit value, the LSB is the byte which occupies the rightmost 8 bits.

Master Password

A 1-8 character string which may be used to access any file upon a diskette.

MSB

Abbreviation for Most Significant Byte. In a 16-bit value, the MSB is the byte which occupies the leftmost 8 bits.

NIL

An inactive state which disk drive and character-oriented devices may assume before being ASSIGNed to a driver or after being KILLed. In the case of a disk driver device, the NIL condition causes the drive to respond as "not ready", and in the case of character-oriented devices, the device ignores output data and provides no input data.

NRN

Abbreviation for Next Record Number. A 2-byte value contained in the FCB which indicates the number of the next physical record in the file.

Overlay

A program module designed to occupy the same area of memory as other programs. Only one overlay program, such as the DOSPLUS IV low and high overlay groups, may occupy any area of RAM at any given time, and it is the responsibility of an overlay loader program to supervise the loading or overlays as they are needed.

Password

A 1-8 character field which is used to obtain access to protected files.

Physical Record

The smallest unit of data which may be read from or written to a disk drive device. This is typically 256 bytes, although it may vary depending on the type of hardware employed.

Platter

On a rigid drive, a flat, cylindrical disk which is rotated at high speed and contains two recording surfaces; one on the top surface and one on the bottom surface.

Pointer

A general term for any value which is used to reference another value, especially a 2-byte word which contains the address of some other data.

Random Access

A mode in which file data may be read or written in any order desired, as opposed to sequential access.

Read/Write Head

A component of a disk drive which may be positioned over any concentric track on a diskette. The read/write head is responsible for picking up the magnetic information stored on the diskette, and for recording new information on the diskette.

Re-try

To repeat a disk I/O operation that resulted in an error. DOSPLUS IV automatically re-tries when it encounters an error when attempting to read a physical record from disk.

Sector

The smallest unit of data which may be read from or written to a disk drive device. Also referred to as a physical record.

Segment

A portion of a disk file, described by an entry in the segment descriptor list in a file's directory entry. Segments are contiguous blocks of granlues that do not exceed 32 granules in length.

Sequential Access

A mode in which data may only be read from or written to a file in linear order; the first record must be read before the second, the second before the third, etc.

Software Write Protect

A flag which may be set with the DOSPLUS library command CONFIG, or thorough user software, which informs a disk drive device driver program that the diskette should not be written to.

System Files

Files which have the system attribute set as part of the file protection status. Normally, users may not create files with the system attribute.

System RAM

An area of RAM located in low memory used to hold data such as DCBs and DCTs, or small programs. This area of RAM is valuable because it always remains accessible regardless of which RAM bank is currently resident in the upper 32K of memory. The pointers $LLOW and $LHIGH indicate where the free portion of system RAM currently resides.

Track

A single circular magnetic recording area on a diskette. Diskettes generally contain many circular tracks.

Trapdoor Code

Under DOSPLUS IV, the trapdoor code is a two-byte value generated from the 8 character file access and update passwords.

User Files

Any file which does not have the system attribute set.

User Record

A 1-255 byte area of RAM used to contain a logical record to be read from or written to a disk file. The DOS uses this user record in conjunction with the blocking buffer in order to block and deblock records in blocked files.

Vector

A small portion of memory, usually on the order a few bytes, which contains machine instructions or data used to divert program flow to another area of RAM.

Volume

A logical disk drive; on a rigid drive, a single partition of the physical drive.

Disk driver example

```
;
;       THIS PACKAGE CONSISTS OF TEN ROUTINES:
;
;       0 -     DCHECK  - CHECK DRIVE READY
;       1 -     DHOME   - HOME/INITIALIZE DRIVE
;       2 -     DSEEK   - SEEK SPECIFIED ADDRESS
;       3 -     DREAD   - READ SECTOR W/SEEK
;       4 -     DVERF   - VERIFY SECTOR W/SEEK
;       5 -     DWRITE  - WRITE SECTOR W/SEEK
;       6 -     SREAD   - READ SYSTEM SECTOR W/SEEK
;       7 -     SWRITE  - WRITE SYSTEM SECTOR W/SEEK
;       9 -     DWRTE1  - WRITE SECTOR W/AM
;       8 -     DFORMT  - FORMAT TRACK W/SEEK
;
;       HARDWARE ADDRESSING
;
BASE    EQU     50H             ;HDC BASE ADDRESS
HD#D    EQU     BASE            ;HDC DATA REGISTER
HD#E    EQU     BASE+1          ;HDC ERROR REGISTER
HD#W    EQU     BASE+1          ;HDC WRITE PRECOMP
HD#X    EQU     BASE+2          ;HDC SECTOR COUNT
HD#S    EQU     BASE+3          ;HDC SECTOR REGISTER
HD#L    EQU     BASE+4          ;HDC CYLINDER LOW
HD#H    EQU     BASE+5          ;HDC CYLINDER HIGH
HD#Q    EQU     BASE+6          ;HDC SIZE/HEAD/DRIVE
HD#C    EQU     BASE+7          ;HDC COMMAND/STATUS
;
;       WD-1000 COMMANDS
;
HREST   EQU     00010000B       ;RESTORE & INIT
HREAD   EQU     00100000B       ;READ SECTOR
HWRTE   EQU     00110000B       ;WRITE SECTOR
HFRMT   EQU     01010000B       ;FORMAT TRACK
HSEEK   EQU     01110000B       ;SEEK ADDRESS
;
;       INITIALIZATION CODE SEQUENCE
;
;       ENT     BC =>   INPBUF$
;               DE =>   HIGH$
;               HL =>   COMMAND LINE
;               IX =>   DCT
;               IY =>   DCT DESCRIPTOR
;
        ORG     HI$ORG
;
START   PUSH    DE              ;SAVE HIGH
        LD      HL,TITLE        ;'DRIVER TITLE'
        LD      A,DSPLY@
        RST     SVC             ;DISPLAY MESSAGE
        POP     HL              ;HL => HIGH$
```

```
;
;       FETCH MEMORY FROM DOS
;
        LD      BC,ENDDVR-BEGDVR
        OR      A                   ;CLR CRY
        SBC     HL,BC               ;HIGH$-LEN DRIVER
        LD      B,0
        LD      A,HIGH@
        RST     SVC                 ;SET HIGH$
        INC     HL                  ;HL => BLOCK
;
        PUSH    HL                  ;SAVE => START FREE
        LD      BC,BEGDVR
        OR      A
        SBC     HL,BC               ;HL = OFFSET
        LD      C,L
        LD      B,H                 ;BC = OFFSET
;
;       ADJUST DRIVER ADDRESSES
;
        LD      IY,TABLE            ;IY => ADD TABLE
        LD      A,27                ;ENTRY COUNT
ADJ1    LD      L,(IY+0)
        LD      H,(IY+1)            ;HL => ADD LOCATION
        LD      E,(HL)
        INC     HL
        LD      D,(HL)              ;DE = ADDRESS
        EX      DE,HL               ;HL = ADDRESS
        ADD     HL,BC               ;HL = NEW ADDRESS
        EX      DE,HL               ;DE = NEW ADDRESS
        LD      (HL),D
        DEC     HL
        LD      (HL),E              ;SET NEW ADD
        LD      DE,2                ;OFFSEY
        ADD     IY,DE               ;IY => NEXT ADD
        DEC     A                   ;DONE?
        JR      NZ,ADJ1             ;IF NOT
        POP     DE                  ;DE => START FREE
;
;       INSTALL DRIVER HERE
;
        LD      (IX+1),E            ;ADD TO DCT
        LD      (IX+2),D            ;ADD TO DCT
        LD      HL,BEGDVR           ;HL => DRIVER
        LD      BC,ENDDVR-BEGDVR
        LDIR                        ;MOVE!
        RES     7,(IX+3) ;5"
        SET     5,(IX+3)            ;HD
        RES     1,(IX+3)            ;FIXED
        LD      (IX+4),6            ;STEP 6 (3MS)
        LD      (IX+12),1           ;SURFACE COUNT
        LD      (IX+13),32          ;SEC/TRACK
        SET     0,(IX+3)            ;LOG IT!
        RES     3,(IX+0)            ;DEVICE ACTIVE
```

```
        LD      E,(IX+1)
        LD      D,(IX+2)        ;=> DRIVER
        LD      HL,MES1+20      ;'0000H'
        LD      A,HEX16@
        RST     SVC             ;BIN/HEX CONVERSON
        LD      HL,MES1         ;'DRIVER INSTALLED AT'
        LD      A,DSPLY@
        RST     SVC
        LD      HL,0            ;NO ERRORS
        LD      A,EXIT@         ;NORMAL EXIT
        RST     SVC
;
;       TABLE OF NON-RELOCATABLE ADDRESSES
;
TABLE   DEFW    RL1
        DEFW    RL2
        DEFW    RL3
        DEFW    RL4
        DEFW    RL5
        DEFW    RL6
        DEFW    RL7
        DEFW    RL8
        DEFW    RL9
        DEFW    RL10
        DEFW    RL11
        DEFW    RL12
        DEFW    RL13
        DEFW    RL14
        DEFW    RL15
        DEFW    RL16
        DEFW    RL17
        DEFW    RL18
        DEFW    RL19
        DEFW    RL20
        DEFW    RL21
        DEFW    RL22
        DEFW    RL23
        DEFW    RL24
        DEFW    RL25
        DEFW    RL26
        DEFW    RL27
;
;       MESSAGES AND TEXT STRINGS
;
TITLE   DEFM    'WD/DVR - DOSPLUS Rigid disk driver - 1.00'
        DEFB    LF
        DEFM    '(c) Copyright 1983, Micro-Systems Software Inc.'
        DEFB    LF
        DEFB    CR
;
MES1    DEFM    'Driver installed at 0000H'
        DEFB    CR
```

```
;
;       FLOPPY/HARD DRIVER ENTRY
;
BEGDVR  EQU     $
HCODE   BIT     5,(IY+3)        ;HARD?
        JR      NZ,HCODE1       ;IF YES
        LD      A,8             ;DRIVE NOT AVAILABLE
        OR      A               ;NZ STATUS
        RET
;
HCODE1  PUSH    HL              ;SAVE BUFFER
        LD      HL,DTABLE       ;ROUTINE TABLE
RL17    EQU     $-2
        ADD     A,A             ;* 2
        ADD     A,L
        LD      L,A             ;HL => ENTRY
        JR      NC,$+3
        INC     H
        LD      A,(HL)          ;LSB
        INC     HL              ;NEXT
        LD      H,(HL)          ;MSB
        LD      L,A             ;HL => ROUTINE
        EX      (SP),HL         ;RESTORE BUFFER
        RET                     ;GO!
;
DTABLE  DEFW    DCHECK          ;CHECK DRIVE READY
RL18    EQU     $-2
        DEFW    DHOME           ;HOME/INIT DRIVE
RL19    EQU     $-2
        DEFW    DSEEK           ;SEEK ADDRESS
RL20    EQU     $-2
        DEFW    DREAD           ;READ SECTOR
RL21    EQU     $-2
        DEFW    DVERF           ;VERIFY SECTOR
RL22    EQU     $-2
        DEFW    DWRITE          ;WRITE SECTOR
RL23    EQU     $-2
        DEFW    SREAD           ;READ SYSTEM SECTOR
RL24    EQU     $-2
        DEFW    SWRITE          ;WRITE SYSTEM SECTOR
RL25    EQU     $-2
        DEFW    DWRIT1          ;WRITE ALT SECTOT
RL26    EQU     $-2
        DEFW    DFORM           ;FORMAT TRACK
RL27    EQU     $-2
```

```
;
;       $DCHECK - CHECK DRIVE READY
;
;       ENT     IY =>   DCT
;
;       EXT     Z SET IF DRIVE READY
;               C SET IF WRITE PROT
;
DCHECK  LD      E,0             ;E = SECTOR
        CALL    UPTASK          ;UPDATE TASK FILE
RL1     EQU     $-2
        IN      A,(HD#C)        ;GET STATUS
        CPL                     ;INVERT
        AND     40H             ;READY?
        RET     NZ              ;NOT AVAIL
        LD      A,(IY+3)        ;GET SOFT WP
        RLCA                    ;WP TO 7
        AND     80H             ;WP ONLY
        ADD     A,A             ;WP TO CRY
        RET
;
;       $SREAD - READ SYSTEM SECTOR
;
;       ENT     E =     SECTOR (0-3 IF DCT ASSUMED)
;               IY =>   DCT (ASSUMED)
;               HL =>   I/O BUFFER
;
;       EXT     IY =>    CORRECT DCT
;
SREAD   BIT     0,(IY+3)        ;LOG DISK?
        CALL    NZ,SREAD3       ;IF YES
RL2     EQU     $-2
        RET     NZ              ;IF ERROR
        LD      D,(IY+18)       ;DIR CYLINDER
        LD      A,DREAD#        ;DREAD
        CALL    DISKIO%         ;DO IT!
        RET
;
SREAD3  CALL    REGSAV%         ;SAVE REGISTERS
        LD      DE,0<8+2        ;CYL,SEC
        CALL    DHOME           ;HOME DRIVE
RL3     EQU     $-2
        LD      A,DVERF#        ;DVERF
        CALL    DISKIO%         ;DO IT!
        RET     NZ              ;IF ERROR
```

```
        LD      A,(HL)          ;GET 1ST CHAR
        CP      'D'             ;'DCT' ?
        LD      A,17            ;DIR READ ERROR
        RET     NZ              ;IF ERROR
        RES     0,(IY+3)        ;DISK LOGGED
        INC     HL              ;NEXT
        INC     HL
        INC     HL              ;HL => DATA
        LD      DE,11           ;OFFSET
        ADD     IY,DE           ;IY => DCT PERISH
        PUSH    IY
        POP     DE              ;DE => DCT PERISH
        LD      BC,9            ;COUNT
        LDIR                    ;MOVE!
        XOR     A               ;NO ERROR
        RET
;
;       $DHOME - SEEK TRACK 0
;
;       ENT     IY =>   DCT
;
DHOME   LD      A,HREST+6       ;STEP
        CALL    HFCNW           ;HOME
RL4     EQU     $-2
        LD      A,(IY+4)        ;GET STEP
        OR      HREST           ;RESTORE CMD
        CALL    HFCNW           ;ISSUE CMD
RL5     EQU     $-2
        LD      (IY+9),0        ;HEAD AT 0
        RET
;
;       $DSEEK - DISK SEEK FUNCTION
;
;       ENT     DE =    CYL,SEC
;               IY =>   DCT
;
DSEEK   LD      A,(IY+4)        ;STEP RATE
        OR      HSEEK           ;SEEK COMMAND
        CALL    HFCNW           ;ISSUE CMD
RL6     EQU     $-2
        LD      (IY+9),D        ;SET CYL
        RET                     ;SEEK COMMAND
;
;       READ SECTOR ROUTINES
;
;       ENT      E =    LOGICAL SECTOR
;                D =    CYLINDER
;               HL =>   I/O BUFFER
;
DREAD   CALL    DIOP
RL7     EQU     $-2
        DEFB    HREAD           ;HDC READ
        DEFB    5               ;RETRY COUNT
```

```
        DEFB    60              ;ERROR OFFSET
        DEFB    1               ;I/O TYPE
;
DVERF   CALL    DIOP
RL8     EQU     $-2
        DEFB    HREAD           ;HDC READ
        DEFB    2               ;RETRY COUNT
        DEFB    60              ;ERROR OFFSET
        DEFB    1               ;I/O TYPE
;
;       WRITE SECTOR ROUTINES
;
;       ENT      E =    LOGICAL SECTOR
;                D =    CYLINDER
;               HL =>   I/O BUFFER
;
DWRITE  CALL    DIOP
RL9     EQU     $-2
        DEFB    HWRTE           ;HDC WRITE
        DEFB    5               ;RETRY COUNT
        DEFB    60              ;ERROR OFFSET
        DEFB    2               ;I/O TYPE
;
SWRITE  LD      D,(IY+18)       ;DIR TRACK
DWRIT1  CALL    DIOP
RL10    EQU     $-2
        DEFB    HWRTE           ;HDC WRITE
        DEFB    5               ;RETRY COUNT
        DEFB    60              ;ERROR OFFSET
        DEFB    2               ;I/O TYPE
;
;       FORMAT TRACK ROUTINE
;
;       ENT      E =    LOGICAL SECTOR
;                D =    CYLINDER
;               HL =>   I/O BUFFER
;
DFORM   CALL    DIOP
RL11    EQU     $-2
        DEFB    HFRMT           ;HDC FORMAT
        DEFB    1               ;RETRY COUNT
        DEFB    60              ;ERROR OFFSET
        DEFB    4               ;I/O TYPE
```

```
;
;       DISK I/O OPERATION
;
;       ENT      E =     LOGICAL SECTOR
;                D =     CYLINDER
;               HL =>    I/O BUFFER
;               IY =>    DRIVE CONTROL TABLE
;               SP =>    FDC FUNCTION
;              SP+1 =>   RETRY COUNT
;              SP+2 =>   ERROR CODE OFFSET
;              SP+3 =>   XFER OPCODE
;
DIOP    POP     IX              ;IX => INFO
        BIT     0,(IX+3)        ;INPUT?
        JR      NZ,DIOP1        ;IF YES
        LD      A,(IY+3)        ;GET FLAGS
        AND     40H             ;WP?
        LD      A,19            ;WRITE PROT!
        RET     NZ              ;IF SOFT WP
;
;       WINCHESTER I/O ROUTINE
;
DIOP1   CALL    UPTASK          ;UPDATE TASK FILE
RL12    EQU     $-2
        LD      BC,0<8+HD#D     ;COUNT & DATA REG
        LD      A,(IX+0)        ;HDC COMMAND
        OUT     (HD#C),A        ;ISSUE CMD
        BIT     0,(IX+3)        ;INPUT?
        JR      NZ,DIOP2        ;IF YES
;
;       HD OUTPUT OPERATION
;
        OTIR                    ;WRITE DATA
        CALL    HBUSY           ;WAIT TIL READY
RL13    EQU     $-2
        JR      DIOP3           ;ALL DONE!
;
;       HD INPUT OPERATION
;
DIOP2   CALL    HBUSY           ;WAIT TIL READY
RL14    EQU     $-2
        INIR                    ;READ DATA
```

```
;
;       GET HD ERROR STATUS
;
DIOP3   IN      A,(HD#E)        ;GET ERROR CODE
        LD      B,A             ;B = CODE
        IN      A,(HD#C)        ;GET STATUS
        AND     1               ;ERROR?
        RET     Z               ;IF NOT
;
DIOERR  LD      A,(IX+2)        ;ERROR OFFSET
DIOER1  RRC     B               ;BIT TO CRY
        RET     C               ;ANY?
        INC     A               ;ERROR CODE
        JR      DIOER1          ;TIL FOUND
;
;       UPDATE HARD DISK TASK FILE
;
;       ENT      E =    LOGICAL SECTOR
;                D =    CYLINDER
;               IY =>   DCT
;
UPTASK  PUSH    BC              ;SAVE REGISTERS
        PUSH    HL
;
        PUSH    DE              ;SAVE CYL
        LD      A,(IY+13)       ;SEC/TRACK
        CALL    SDIVD%          ;GET HEAD,SEC
        LD      D,A             ;D = HEAD
        LD      A,(IY+8)        ;SECTOR OFFSET
        ADD     A,E
        OUT     (HD#S),A        ;SET SECTOR
        LD      E,D             ;E = HEAD
;
        LD      A,(IY+12)       ;SURFACE COUNT
        BIT     2,(IY+3)        ;SKIP?
        JR      Z,UPTSK1        ;IF NO
        SRL     A               ;/2
UPTSK1  CALL    SDIVD%          ;GET CYL,HEAD
        LD      C,A             ;C = CYL OFFSET
        LD      D,E             ;D = HEAD
;
        LD      A,(IY+10)       ;BINARY DRIVE
        RLCA
        RLCA
        RLCA                    ;TO BITS 3-5
        ADD     A,(IY+5)        ;HEAD OFFSET
        ADD     A,D             ;+ HEAD
        OUT     (HD#Q),A        ;SET SIZE/DRIVE/HEAD
        POP     DE              ;GET CYL
```

```
;
        LD      A,(IY+6)        ;CYL OFFSET
        ADD     A,D             ;CYLINDER
        LD      L,A  •
        LD      A,(IY+7)        ;CYL OFFSET
        ADC     A,0             ;MSB
        LD      H,A             ;HL = CYL
;
        BIT     2,(IY+3)        ;SKIP?
        JR      Z,UPTSK2        ;IF NOT
        ADD     HL,HL           ;CYL * 2
        LD      B,0             ;BC = CYL OFFSET
        ADD     HL,BC           ;HL = CYL
;
UPTSK2  LD      A,L
        OUT     (HD#L),A        ;SET CYL LOW
        LD      A,H
        OUT     (HD#H),A        ;SET CYL HIGH
        POP     HL              ;RESTORE REG
        POP     BC
        RET
;
;       ISSUE HD FUNCTION & WAIT
;
;       ENT      A =    DISK FUNCTION
;                IY =>  DCT
;
HFCNW   PUSH    AF              ;SAVE CMD
        CALL    UPTASK          ;UPDATE TASK FILE
RL15    EQU     $-2
        POP     AF              ;GET CMD
        OUT     (HD#C),A        ;ISSUE CMD
        CALL    HBUSY           ;WAIT TIL READY
RL16    EQU     $-2
        IN      A,(HD#C)        ;GET STATUS
        AND     1               ;ERROR?
        RET     Z               ;IF NOT
        IN      A,(HD#E)        ;A = ERROR
        RET
```

```
;
;       WAIT FOR HDC READY
;
HBUSY   IN      A,(HD#C)        ;HDC STATUS
        RLCA                    ;BUSY?
        JR      C,HBUSY         ;WAIT
        RET
ENDDVR  EQU     $
;
        END     START
```

Character-oriented device driver example

```
;
;       INITIALIZATION CODE SEQUENCE
;
;       ENT     BC =>   KEYBUF$
;               DE =>   HIGH$
;               HL =>   COMMAND LINE
;               IX =>   DCB
;               IY =>   DCB DESCRIPTOR
;
        ORG     HI$ORG
;
START   PUSH    HL              ;SAVE COMMAND LINE
        LD      HL,TITLE        ;'DRIVER TITLE'
        LD      A,DSPLY@
        RST     SVC             ;DISPLAY MESSAGE
;
        LD      C,KI#           ;KEYBOARD DEVICE
        LD      A,LOCDCB@       ;LOCATE DCB
        RST     SVC
        PUSH    IX
        POP     HL              ;CURRENT DCB
        OR      A               ;CLR CF
        SBC     HL,DE           ;@KI DEVICE?
        JP      NZ,POST2        ;ILLEGAL DEIVCE!
        POP     HL              ;GET COMMAND LINE
;
;       FETCH/OPEN MKEY FILE
;
        LD      DE,DCB#1        ;FILE DCB
        LD      A,FSPEC@        ;FETCH MKEY FILE
        RST     SVC
        JP      NZ,POST1        ;IF ERROR
        LD      A,(DE)
        CP      '@'             ;DEVICE SPEC?
        JP      Z,POST1
        CP      '*'             ;DEIVCE/WILD?
        JP      Z,POST1
        LD      HL,DEXT         ;'TXT'
        LD      A,FEXT@         ;OPTIONAL EXTENSION
        RST     SVC
;
        LD      HL,BLKBUF       ;BLOCKING BUFFER
        LD      B,0             ;LRL
        LD      A,OPEN@         ;OPEN MKEY FILE
        RST     SVC
        JP      NZ,POSTE        ;IF ERROR
```

```
        LD      A,FLAGS@        ;FETCH SYSTEM FLAGS
        RST     SVC
        BIT     6,(IY+3)        ;KMEY LOADED?
        JR      NZ,INFILE       ;IF YES
;
;       INSTALL MACRO-KEY DRIVER
;
        LD      B,0             ;HIGH$
        LD      L,B
        LD      H,B             ;FETCH HIGH$
        LD      A,HIGH@
        RST     SVC
        LD      DE,ENDDVR-BEGDVR
        OR      A               ;CLR CRY
        SBC     HL,DE           ;HIGH$-LEN DRIVER
        LD      A,HIGH@
        RST     SVC             ;SET HIGH$
        INC     HL              ;HL => BLOCK
;
        PUSH    HL              ;SAVE => DRIVER
        LD      BC,BEGDVR
        OR      A               ;CLR CRY
        SBC     HL,BC
        LD      C,L
        LD      B,H             ;BC = OFFSET
        POP     DE              ;DE => DRIVER
;
        LD      L,(IX+1)
        LD      H,(IX+2)        ;FETCH KI DRIVER
        LD      (KDVR),HL       ;SAVE DRIVER
        LD      (IX+1),E
        LD      (IX+2),D        ;NEW DRIVER FOR @KI
        LD      (IX+0),I        ;INPUT DEIVCE
        LD      HL,(RL1)        ;DRIVER ADDRESS
        ADD     HL,BC           ;CORRECT
        LD      (RL1),HL
        LD      HL,BEGDVR       ;HL => DRIVER
        LD      BC,ENDDVR-BEGDVR
        LDIR                    ;MOVE!
;
        LD      A,FLAGS@        ;SYSTEM FLAGS
        RST     SVC
        SET     6,(IY+3)        ;MKEY LOADED
```

```
;
;       POST DRIVER INSTALLATION
;
        LD      F,(IX+1)
        LD      D,(IX+2)        ;=> DRIVER
        LD      HL,MESI+20      ;'0000H'
        LD      A,HEX16@
        RST     SVC             ;BIN/HEX CONVERSON
        LD      HL,MESI         ;'DRIVER INSTALLED AT'
        LD      A,DSPLY@        ;OUTPUT
        RST     SVC
;
;       INSTALL MACRO-KEY FILE
;
;       ENT     IX =>   DCB
;
IFILE   LD      L,(IX+1)
        LD      H,(IX+2)        ;HL => DRIVER
        PUSH    HL
        POP     IX              ;IX => MKEY DCB
;
        LD      L,(IX+3)
        LD      H,(IX+4)        ;HL => MKEY TABLE
        LD      C,(IX+5)
        LD      B,(IX+6)        ;BC = LENGTH TABLE
        LD      A,B
        OR      C               ;NULL LEN?
        OR      Z,IFILE0        ;NO TABLE
        CALL    RECHI           ;RECLAIM MEMORY
;
IFILE0  LD      BC,0            ;NULL
        LD      (IX+5),C        ;SET LENGTH TABLE
        LD      (IX+6),B
        INC     BC              ;1 BYTE
        CALL    FETHI           ;FETCH MEMORY
        LD      (IX+3),L        ;SET START TABLE
        LD      (IX+4),H
        LD      (HL),-1         ;END TABLE
        LD      (IX+5),C        ;SET LENGTH TABLE
        LD      (IX+6),B
```

```
;
;       READ MACRO-KEY DEFINITION
;
IFILE1  LD      HL,LBUFF        ;LINE BUFFER
        LD      BC,80<8+0       ;COUNT/FLAGS
        CALL    GET             ;FETCH BYTE
        CP      ' '             ;BLANK?
        JR      Z,$-5           ;IGNORE
        CP      '.'             ;COMMENT?
        JR      Z,IFILE5        ;SKIP
        INC     C               ;NZ
        CP      ''''            ;LITERAL?
        JR      Z,IFILE8        ;SKIP
        CP      '"'             ;LITERAL?
        JR      Z,IFILE8        ;SKIP
        CALL    HEXCP           ;HEX?
        RLCA
        RLCA                    ;LOW
        RLCA                    ;TO
        RLCA                    ;HIGH
        LD      E,A             ;SAVE MSB
        CALL    GET             ;GET LSB
        CALL    HEXCP           ;HEX?
        OR      E               ;A = BYTE
        JR      IFILE4          ;SKIP
;
IFILE8  CALL    GET             ;GET BYTE
        LD      E,A             ;SAVE BYTE
        CALL    GET
        CP      ''''            ;END LITERAL?
        JR      Z,IFILE9
        CP      '"'             ;END LITERAL?
        JP      NZ,POST3        ;INVALID DATA
;
IFILE9  LD      A,E             ;GET BYTE
        CP      'a'             ;LOWER CASE?
        JR      C,IFILE4
        CP      'z'+1           ;LOWER CASE?
        JR      NC,IFILE4
        AND     5FH             ;FOLD TO UPPER
;
IFILE4  AND     7FH             ;IGNORE HIGH BIT
        LD      (HL),A          ;TO LINE BUFFER
        INC     HL              ;BUMP
        CALL    GET             ;FETCH BYTE
        CP      ' '             ;BLANK?
        JR      Z,$-5           ;IGNORE
        CP      '='             ;ASSIGNMENT?
        JP      NZ,POST3        ;INVALID DATA
```

```
;
;       FETCH MACRO-KEYSTROKES
;
IFILE5  CALL    GET             ;GET KEYSTROKES
        INC     C
        DEC     C               ;COMMENT?
        JR      NZ,IFILE6       ;IF NOT
        CP      CR              ;END OF LINE?
        JR      NZ,IFILE5       ;TIL END OF LINE
        JR      IFILE1          ;FETCH NEXT LINE
;
IFILE6  LD      (HL),A          ;TO LINE BUFFER
        INC     HL              ;BUMP
        CP      CR              ;END KEYSTROKES?
        JR      Z,IFILE7        ;IF YES
        DJNZ    IFILE5          ;TIL END LINE
        JP      POST3           ;LINE TOO LONG!
;
;       MOVE TO USER MEMORY
;
IFILE7  LD      A,80+1+1        ;DEF+LEN+CR
        SUB     B
        LD      C,A
        LD      B,0             ;BC = LENGTH MKEY
        CALL    FETHI           ;FETCH MEMORY
        PUSH    HL              ;SAVE => MEMORY
        LD      L,(IX+5)        ;FETCH LENGTH
        LD      H,(IX+6)
        ADD     HL,BC           ;ADD LINE
        LD      (IX+5),L        ;NEW LENGTH
        LD      (IX+6),H
        POP     DE              ;GET => MEMORY
        LD      (IX+3),E
        LD      (IX+4),D
        LD      HL,LBUFF        ;MOVE LINE
        LDIR
;
;       NEXT DEFINITION IF ANY
;
        LD      DE,DCB#1        ;MKEY DCB
        LD      A,CKEOF@        ;END FILE?
        RST     $28
        JP      Z,IFILE1        ;TIL END
```

```
;
;       POST TABLE INSTALLATION
;
        LD      E,(IX+3)        ;DE => MKEY TABLE
        LD      D,(IX+4)
        LD      HL,MES2+24      ;'0000H'
        LD      A,HEX16@
        RST     SVC             ;BIN/HEX CONVERSON
        LD      HL,MES2         ;'MKEY TABLE INSTALLED AT'
        LD      A,DSPLY@        ;OUTPUT
        RST     SVC
;
;       ALL DONE, LET'S GO HOME!
;
        LD      HL,0            ;NO ERRORS
        LD      A,EXIT@         ;NORMAL PROGRAM EXIT
        RST     SVC
;
;       FETHI - FETCH HIGH MEMORY BLOCK
;
;       ENT     BC = LENGTH OF BLOCK
;
;       EXT     HL => START ADDRESS OF BLOCK
;               (HIGH$) IS ADJUSTED ACCORDINGLY
;
FETHI   PUSH    DE              ;SAVE REGISTERS
        PUSH    BC
        LD      B,0             ;HIGH$
        LD      L,B
        LD      H,B
        LD      A,HIGH@         ;FETCH HIGH$
        RST     SVC
        POP     DE
        PUSH    DE              ;GET LEN
        OR      A
        SBC     HL,DE           ;NEW HIGH$
        LD      A,HIGH@         ;SET HIGH$
        RST     SVC
        INC     HL              ;HL => USER AREA
        POP     BC              ;RESTORE REG
        POP     DE
        RET
```

```
;
;       RECHI - RECLAIM HIGH MEMORY BLOCK
;
;       ENT     HL => ADDRESS OF MEMORY BLOCK
;               BC = LENGTH OF BLOCK
;
;       EXT     MEMORY RECLAIMED IF (HIGH$) = HL-1
;
RECHI   PUSH    DE              ;SAVE REGISTERS
        PUSH    BC
        PUSH    HL
;
        LD      B,0             ;HIGH$
        LD      L,B
        LD      H,B
        LD      A,HIGH@         ;FETCH HIGH$
        RST     SVC
        PUSH    HL
        POP     BC              ;BC => HIGH$
;
        POP     HL              ;GET ADDRESS
        POP     DE              ;GET LENGTH
;
        PUSH    HL              ;SAVE ADDRESS
        PUSH    DE              ;SAVE LEN
        SCF                     ;CLR CRY
        SBC     HL,BC           ;ADDRESS -1 @ HIGH$?
        JR      NZ,RECHI1       ;YEP
        EX      DE,HL           ;HL = LEN
        ADD     HL,BC           ;HL = NEW HIGH$
        LD      B,0             ;HIGH$
        LD      A,HIGH@         ;SET HIGH$
        RST     SVC
;
RECHI1  POP     BC              ;RESTORE REG
        POP     HL
        POP     DE
        RET
;
;       GET BYTE FROM FILE
;
GET     PUSH    DE              ;SAVE
        LD      DE,DCB#1        ;SOURCE DCB
        LD      A,GET@
        RST     SVC             ;GET BYTE
        POP     DE              ;RESTORE
        JP      NZ,POSTE        ;IF ERROR
        RET
```

```
;
;       CHECK/CONVERT HEX DIGIT
;
HEXCP   SUB     '0'             ;IGNORE ASCII
        JP      C,POST3         ;INVALID!
        CP      10
        RET     C               ;IF 0-9
        CP      17
        JP      C,POST3         ;INVALID!
        SUB     16
        JP      NC,POST3        ;INVALID!
        RET
;
;       POST ERROR MESSAGE AND EXIT
;
POST1   LD      A,46            ;ILLEGAL FILE SPEC
        DEFB    1
POST2   LD      A,45            ;ILLEGAL DEVICE SPEC
        DEFB    1
POST3   LD      A,49            ;INVALID DATA PROVIDED
;
POSTE   OR      40H             ;NO DETAIL/ABORT
        LD      C,A             ;MOVE CODE
        LD      A,ERROR@        ;POST ERROR
        RST     SVC
;
;       MESSAGES AND TEXT STRINGS
;
TITLE   DEFM    'MKEY - DOSPLUS Macro-key driver - 1.00'
        DEFB    LF
        DEFM    '(c) Copyright 1983, Micro-Systems Software Inc.'
        DEFB    LF
        DEFB    CR
;
MES1    DEFM    'Driver installed at 0000H'
        DEFB    CR
;
MES2    DEFM    'MKEY table installed at 0000H'
        DEFB    CR
;
DEXT    DEFM    'TXT'
;
;       VARIABLES AND DATA AREAS
;
DCB#1   DEFS    32              ;MKEY FILE DCB
LBUFF   DEFS    82              ;LINE BUFFER
BLKBUF  DEFS    256             ;BLOCKING BUFFER
```

```
;
;       MKEY@@ - MACRO-KEY DRIVER ROUTINE
;
;       ENT       C =     CHARACTER
;                 B =     I/O OPERATION
;                IX =>    DEVICE CONTROL BLOCK
;                CF =     SET IF INPUT
;                ZF =     SET IF OUTPUT
;
;       EXTENDED DEVICE CONTROL BLOCK:
;
;       +0,1     DRIVER BRENCH
;       +2       DRIVER FLAGS
;       +3,4     START MKEY TABLE
;       +5,6     MKEY TABLE LENGTH
;       +7,8     PENDING MKEY POSITION
;
;       DRIVER FLAGS:
;
;       BIT:     7 - PENDING MKEY OPERATION
;                6 - PENDING MKEY LINK
;
BEGDVR  EQU      $
;
;       DEVICE CONTROL BLOCK
;
MKEY@@  JR       MKEY@              ;TO DRIVER
        DEFB     0                  ;DRIVER FLAGS
        DEFW     0                  ;START MKEY TABLE
        DEFW     0                  ;MKEY TABLE LENGTH
        DEFW     0                  ;PENDING MKEY POSITION
;
;       CHECK FOR PENDING MKEY
;
MKEY@   LD       IY,MKEY@@          ;IY => MKEY DCB
RL1     EQU      $-2
        BIT      7,(IY+2)           ;PENDING MKEY?
        JR       Z,KSCAN            ;IF NOT
        LD       L,(IY+7)
        LD       H,(IY+8)           ;HL => POSITION
        LD       A,(HL)             ;FETCH MKEY
        INC      HL                 ;BUMP MKEY POS
        CP       CR                 ;EOL?
        JR       NZ,MKEY@1          ;IF NOT
        RES      7,(IY+2)           ;NO PENDING MKEY
        JR       MKEY@              ;SCAN KEYBOARD
```

```
;
;       CHECK FOR PENDING LINK
;
MKEY@1  LD      (IY+7),L
        LD      (IY+8),H        ;SAVE MKEY POS
        BIT     6,(IY+2)        ;PENDING LINK?
        JR      NZ,MKEY@7       ;IF YES
        CP      '\'             ;IMPLIED C/R?
        JR      NZ,$+4          ;IF NOT
        LD      A,CR            ;C/R!
        CP      '&'             ;MKEY LINK?
        JR      NZ,MKEY@2       ;IF NOT
        SET     6,(IY+2)        ;PENDING LINK
        JR      MKEY@           ;START OVER
;
MKEY@2  CP      A               ;HAVE KEY
        RET
;
;       SCAN KEYBOARD FOR A KEY
;
KSCAN   PUSH    IY              ;SAVE
        CALL    0               ;EXECUTE DRIVER
KDVR    EQU     $-2
        POP     IY              ;RESTORE
        RET     NZ              ;IF ERROR
        BIT     7,A             ;MKEY GENERATION?
        RET     Z               ;IF NOT
        AND     7FH             ;ASCII KEY
;
MKEY@7  CP      'a'             ;LOWER CASE?
        JR      C,MKEY@3
        CP      'z'+1           ;LOWER CASE?
        JR      NC,MKEY@3
        AND     5FH             ;FOLD TO UPPER
;
;       SEARCH TABLE FOR MACRO-KEY
;
MKEY@3  RES     7,(IY+2)        ;NO PENDING MKEY
        RES     6,(IY+2)        ;NO PENDING LINK
        LD      L,(IY+3)
        LD      H,(IY+4)        ;HL => MKEY TABLE
```

```
;
MKEY@4  LD      F,(HL)          ;FETCH CHAR
        INC     F               ;END TABLE?
        JR      Z,MKEY@6        ;NO KEY FOUND
        CP      (HL)            ;FOUND?
        INC     HL              ;BUMP
        JR      Z,MKEY@5        ;IF FOUND
        PUSH    AF              ;SAVE KEY
        LD      A,CR            ;SEARCH CHAR
        PUSH    BC
        LD      BC,0            ;MAX LENGTH
        CPIR                    ;FIND C/R
        POP     BC
        POP     AF              ;GET KEY
        JR      MKEY@4          ;CHECK AGAIN
;
;       SET-UP PENDING MACRO-KEY
;
MKEY@5  SET     7,(IY+2)        ;PENDING MKEY
        LD      (IY+7),L
        LD      (IY+8),H        ;KEY POSITION
        JR      MKEY@           ;START OVER
;
;       UNDEFINED MACRO-KEY
;
MKEY@6  OR      80H             ;CORRECT KEY
        CP      A               ;Z STATUS
        RET
;
ENDDVR  EQU     $
;
        END     START
```

Filter program example - MORE/CMD

```
;
          ORG       HI$ORG
;
;         READ INPUT FROM KEYBOARD
;
START     LD        HL,BUFFER          ;START TEXT BUFFER
GTKEY     PUSH      DE
          LD        A,KEY@             ;FETCH KEY
          RST       SVC
          POP       DE
          JR        NZ,CKEND
          PUSH      HL                 ;SAVE BUFFER
          SCF
          SBC       HL,DE
          POP       HL                 ;GET BUFFER
          JP        NC,POST1           ;OUT OF MEMORY!
          LD        (HL),A             ;TO BUFFER
          INC       HL                 ;BUMP
          JR        GTKEY              ;TIL END OF FILE
;
CKEND     CP        28                 ;EOF?
          JP        NZ,POSTE           ;I/O ERROR!
          EX        DE,HL              ;DE => END BUFFER
          LD        HL,BUFFER          ;START TEXT BUFFER
;
;         WRITE OUTPUT TO DISPLAY
;
WRDSP1    LD        B,22               ;ROWS
WRDSP2    PUSH      HL                 ;SAVE BUFFER
          OR        A                  ;CLR CRY
          SBC       HL,DE
          POP       HL                 ;GET BUFFER
          JR        Z,DONE             ;ALL DONE
          LD        C,(HL)             ;GET CHARACTER
          INC       HL                 ;BUMP
          PUSH      DE
          LD        A,DSP@
          RST       SVC
          POP       DE
          JP        NZ,POSTE           ;IF ERROR
          LD        A,C                ;GET CHARACTER
          CP        ' '                ;CTL?
          JR        NC,WRDSP2          ;IF NOT
          DJNZ      WRDSP2             ;TIL FULL SCREEN
```

```
;
        PUSH    DE
        PUSH    HL              ;SAVE
        LD      HL,MES1         ;'--- MORE ---'
        LD      A,DSPLY@        ;OUTPUT
        RST     SVC
        JP      NZ,POSTE        ;IF ERROR
WRDSP3  LD      A,KEY@          ;WAIT FOR KEY
        RST     SVC
        JR      NZ,WRDSP3       ;IF EOF WAIT FOR KBD
        POP     HL              ;RESTORE
        POP     DE
        JR      WRDSP1          ;TIL DONE
;
DONE    LD      HL,0            ;NO ERRORS
        LD      A,EXIT@         ;NORMAL EXIT
        RST     SVC
;
;       ERROR HANDLER
;
POST1   LD      A,44            ;OUT OF MEMORY!
POSTE   OR      40H             ;POST/@ABORT
        LD      C,A             ;MOVE CODE
        LD      A,ERROR@
        RST     SVC
;
;       MESSAGES AND TEXT STRINGS
;
MES1    DEFM    '--- MORE ---'
        DEFB    CR
;
BUFFER  EQU     $
;
        END     START
```

A

Allocate disk space.....2-55
APPEND.....2-2
ASCII files.....2-21
ASSIGN.....2-6
  Device drivers.....2-6
  File drivers.....2-6,6-1
  Mem disk drivers.....6-5
  Rigid drivers.....2-6
ATTRIB.....2-10
AUTO.....2-15

B

Background printing(SPOOL).....2-7
BACKUP.....3-1
Bank.....2-130,6-5
BASIC.....5-1
  Creating enhanced BASIC.....5-1
  Enhanced BASIC.....5-1
    BE1.....5-1
    BE2.....5-1
    DI.....5-4
    DR.....5-5
    DU.....5-4
    INPUT@.....5-13
    Label addressing.....5-16
    OPTION.....5-18
    REF.....5-6
    RESOLVE.....5-12
    Shorthand.....5-3
    SORT.....5-9
    SR.....5-8
  Entering BASIC.....5-2
  Error flags.....5-20
  Model III programs.....3-6,5-18
BAUD rate 2-125
Beep.....2-133
BOOT.....2-19
BREAK.....2-20
  Abort.....2-73
  Enable/Disable.....2-20
BUILD.....2-21

C

Calendar.....2-60
Capital letters.....2-131
CAT.....2-24
Catalog of disk.....2-24
CLEAR.....2-29
  Clear files.....2-31
  Clear memory.....2-29
  Clear screen.....2-33
Click - keyboard.....2-133
CLOCK.....2-32
CLS.....2-33
Command syntax.....1-16
CONFIG.....2-34
  Cylinder offset.....2-44
  Hard disk.....2-41
  Head load delay.....2-39
  Head offset.....2-45
  Motor delay.....2-39
  Physical drive numbers.....2-38
  Step rate.....2-37,2-129
  Size.....2-36
  Track size.....2-45
  Write protect drives.....2-38
Configure.....2-34
  Drives.....2-34
    Eight inch drives.....2-36
    Number of sides.....2-37
    Skip (80 track drives)...2-40
    Step rate.....2-37,2-129
CONVERT.....3-6
COPY.....2-48
  Devices.....2-49
  Files.....2-50
CREATE.....2-55
Cursor
  Blink status.....2-131
  Character.....2-131

D

Data pattern.....3-1,3-28
DATE.....2-60
  Calendar.....2-60
  Disable.....2-116
  Display.....2-60,2-116
  Set.....2-60
DEBUG.....2-61
Definitions.....1-19
Delete files.....2-100,2-116
  Directory of.....2-65
  Remove from directory.....2-114
  Restoring after deleting....3-39
Device
  Cancel FORCE/JOIN.....2-121
  File disk.....6-1
  Filter characters to/from...2-78
  Join two devices.2-49,2-97,2-105
  Kill a device.....2-102,2-116
  PIPES.....1-21
  Reroute a device.....2-48,2-85
    2-97,2-105,2-122
DIR.....2-64
DIRCHECK.....3-11
Directory
  Additional entries.....6-1
  Alphabetize.....2-64
  Delete entries.....2-96
  Description.....2-65
  MEDIC.....3-43
  Remove deleted entries.....2-106
  Rename files.....2-109

Repair directory.....3-11
Technical description.....T-7
To disk file.....2-70
To display.....2-64
To printer.....2-70
Wildcard masks.....2-70,1-16
Disk
Available entries.....2-64
Configuration.....2-34
Directory.....2-64
File disk.....6-1
Free space.....2-64,2-88
I/O error trapping.....3-42
Names.....2-106,3-28
Passwords.....2-106,3-28
DISKDUMP.....3-13
DISKZAP.....3-16
Display
Characters and codes.....T-101
Clearing.....2-33
Files and programs.....2-108
DO.....2-73
Auto execute.....2-15
Creating file.....2-21
DOS commands.....2-104,3-33
Drives
Configuration.....2-34
File disk.....6-1
Memdisk.....2-6,2-130,6-5
Naming.....2-109
Step rate.....2-40,2-44,2-132
Write protect.....2-38
Drivers
File disk.....2-6,6-1
Installing.....2-6
Macro keys.....2-6,6-9
Mem disk.....2-6,2-130,6-5
Spooler.....2-6,6-12
Writing.....T-95
DUMP.....2-77

E

EBEEP.....2-133
ERROR.....2-80
Error messages (DOS).....2-80,T-99
Error trapping disk I/O.....3-42

F

File disk.....2-6,6-1
Files
Clearing.....2-29
Closing.....2-121
Copying.....2-48
Creating.....2-21,2-55
Extensions.....1-10
Location on disk.....3-34
Passwords.....2-10,2-106
Record length.....2-9
Renaming.....2-1
Routing to a device...2-48,2-1
Space used on disk.....2-64,3-
Visible status.....2-6,2-9,2-
Filespecs
Allowable characters.....1-
Extensions.....1-
Passwords.....1-
FILTER.....2-
Creating filter file...2-21,2-
Filter programs.....T-1
Installing filters.....2-
MORE/CMD.....1-
Printer filters
DVORAK keyboard.....6-
EPSON printer graphics...6-
FORCE.....2-
FORMAT.....3-
FORMS.....2-
FREE.....2-

G

Granules.....T

H

HELP.....3-
High memory.....2-74,2-1

I

I.....2-
Initialize drives.....2-
Invisible files
Creating.....2-1
Displaying.....2-24,2-6

J

Job Control Language (JCL).....4-
Commands.....4-
Invoking JCL.....4-
Keyboard queue.....4-2,4-1
Labels.....4-
Remarks.....4-
Variables.....4-
JOIN.....2-9

K

Keyboard
Characters and codes.....T-10
Click.....2-13
Macro keys.....2-6
Filtering keys.....2-85
KILL.....2-
Drives/devices.....2-10
Files.....

L

LIB.....2-104
LINK.....2-105
LIST.....2-108
LOAD.....2-110
Lock (disks).....2-113
Logo (Enable/Disable).....2-129
LRL(logical record length)2-55,2-10

M

Macro keys.....2-6,6-9
MAP.....3-34
Masks.....1-16
Master password.....2-113,3-28
MEDIC.....3-43
Memdisk.....6-5
Memory
  Clearing.....2-29
  Protecting.....2-73,2-132
  Saving to disk.....2-77
Mod flags.....2-9,2-24,2-52,2-65
MORE/CMD.....1-22

N

Names
  Devices.....2-120
  Disks.....2-113,3-28
  Drives.....2-120
  File.....1-9

O

Output
  Filtering.....2-81
  Piping.....1-21
  Ports.....2-133
  Routing.....2-85,2-122
Overlays
  Contents.....T-5
  Deleting.....2-100,2-116,3-40

P

Passwords
  Change.....2-10,2-113
  Disk.....2-113,3-29
  File.....2-10
  Remove.....2-10,2-113
PATCH.....3-36
PAUSE.....2-112
Pause scrolling.....2-25,2-66
Pdrive.....2-38
PIPES.....1-21
Port output.....2-133
Printer
  Control codes.....2-89
  Filtering.....2-81,6-15
  Forms control.....2-86
  Indenting.....2-86
  Paging.....2-86
  Spooling.....2-7,6-12
PROT.....2-113
Protection(disk or file).2-10,2-113
Purging files.....2-100,2-116

R

Ram disk.....6-12
Record length.....2-9,2-55
REMOVE.....2-116
RENAME.....2-120
Repair directory
  DIRCHECK.....3-11
  DISKZAP.....3-16
Reserve memory.....2-73,2-130
RESET.....2-121
RESTORE.....3-39
Rigid drives
  Configuring.....2-41
  Moving system to.....3-40
  Partitioning.....T-2
ROUTE.....2-122
Routing devices
  COPY.....2-48
  FORCE.....2-85
  JOIN.....2-97
  LINK.....2-105
  PIPES.....1-21
  ROUTE.....2-122
RS232.....2-124

S

SCREEN.....2-128
Scrolling (pausing).....2-25,2-66
Single drive
  BACKUP.....1-5,3-1
  COPY (prompt).....2-48
Sound
  Bell.....2-133
  Error messages.....2-133
  Keyboard click.....2-133
Step rate.....2-37
Supervisor calls.....T-25
SVC.....T-25
SYSGEN.....3-40
SYSTEM.....2-129
System disks
  Creating
    BACKUP.....3-1
    Double sided disks.....2-37
    SYSGEN.....3-40
  System files
    Contents.....T-5
    Deleting.....2-100,2-116

T

Technical information.....T-1
BOOT/SYS.....T-14
DCB table & organization....T-15
DCT organization.....T-22
Directory structure.....T-7
File entries.....T-11
GAT organization.....T-7
HIT organization.....T-10
Error codes.....T-99
FCB structure.....T-20
Rigid drive partitioning.....T-2
Supervisor calls.....T-25
Arithmetic functions.....T-72
Base conversion.....T-70
Device I/O.....T-32
Disk I/O.....T-81
File handling.....T-41
Interrupt routines.....T-67
Miscellaneous routines...T-89
System control routines..T-51
Technical glossary.....T-103
Writing drivers.....T-95
TIME.....2-137
Disable.....2-137
Display.....2-131,2-137
Set.....2-137
Tone.....2-133
TRAP.....3-42

U

Unlock.....2-113

V

VERIFY.....2-138
Video characters and codes....T-101
Volume.....T-2

W

Wildcard masks.....1-16,1-20
Write protect drives.....2-38