DOSPLUS

USER'S MANUAL

VER. A.3

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

DOSPLUS 3.5 – User's manual addendum – Misc. information

<u>A note to all previous owners of DOSPLUS</u>

If you are a previous DOSPLUS owner (versions 3.3, 3.4, or 4.0), there are a couple of areas that have changed with this new version that deserve a bit of documenting. Specifically, they are:

(1) Methods of configuring the system.
(2) Creating a double density Model I master disk.

One very important note to all owners of version 3.5, whether you are a first time DOSPLUS owner or not. Please read carefully the library command SYSTEM. There is a great deal of information there on "customizing" your DOSPLUS. Used in conjunction with the information in this addendum, you should be able to configure the new system in any manner you desire.

Methods of configuring the system

<u>Using the external device structure with alternate drivers</u>

In earlier versions, DOSPLUS used what is referred to as an "internal" device structure. This means that all drivers needed to operate whatever devices are attached to the computer were <u>internal</u> to the DOSPLUS operating system. This in turn meant that every time we moved to support a new device (a different hard drive, a new printer, etc.) we were forced to create a "new" version of the system.

This has become impractical. Therefore, DOSPLUS 3.5 adopts a new procedure referred to as an "external" device structure. This means that all drivers needed for the operation of that various peripherals are no longer built in to the system but exist as options outside the system. The system will use the drivers provided by the TRS-80's ROMs by default and only load in the new drivers if you indicate that you want them.

This is a great deal more flexible. It allows us to provide support for a <u>much</u> wider range of add on peripherals by simply altering a driver without having to make major changes to the system itself. It also removes many of the memory restrictions associated with maintaining drivers internally in the "system's memory area".

The other thing that the new external device structure means is that certain advanced functions that used to be automatic are not automatic now until you install one of the alternate drivers (usually with the ASSIGN command). Specific examples are the printer driver and, on the Model I, the keyboard and display drivers.

The printer driver. On both the Model I and the Model III, DOSPLUS will now use the ROM printer driver unless you instruct the system to install the PR/DVR driver supplied on the disk for you. This means that automatic pagination, serial printer support, spooled output, etc., are not available until this driver is installed with ASSIGN. This creates an extra step initially, but allows us to supply other alternate drivers for non-standard printers. Later, we will cover how to make the installation of our DOSPLUS printer driver (and any of our drivers) a permanent part of your system.

The Model I keyboard and display drivers. Another area that will surprise some of you will be the lack of repeating keys and lower case support on the Model I when you first boot up the new system. These items are still present, but you must install the KI/DVR keyboard driver and the DO/DVR display driver first. Because the system uses the ROM drivers unless otherwise instructed, the "standard" features are those provided by these drivers until such time as you install an alternate driver and make that the "standard". What is important to remember, is that we have taken away nothing from the system. We may have required you to ask for it before we give it to you, but all the functions of previous systems exist in DOSPLUS 3.3.

What does all this have to do with methods of configuring the system? A great deal. Many users of previous DOSPLUS systems will find that they cannot configure the system to suit their desires until alternate drivers are installed. This is why it is important to understand that the "standard" features are those provided by the ROM drivers. If these features are not sufficient for your needs, then we have provided for your convenience many alternate device drivers that should provide you with a wealth of options. All that is required of you is to install any needed drivers before attempting to configure that area of the system.

This is not limited to just the drivers discussed. Although the printer, keyboard, and display drivers were the ones selected as examples, these are not the only times you would need to use the alternate drivers. For example, if you wanted to install a filter on the display device (@DO), the ROM drivers would not support this. Before this could be done, the alternate display driver DO/DVR would have to be installed with the ASSIGN command.

Also, it is not limited to just the device drivers, although these are the most common. You will also need to ASSIGN alternate disk drivers for some special instances. Examples would include operating a hard disk drive or an eight inch floppy disk drive. These drivers are not provided with the standard 3.5 system and must be requested separately. They may require an extra charge, as in the cases of the hard disk drivers, to name one.

How to permanently configure DOSPLUS

Now that we have addressed the basics of customizing your DOSPLUS 3.5 and how that it may require the installation of alternate drivers, we will move on to the subject of, once that your system is configured the way that you want it with all drivers installed, how do you make this a permanent change.

In the older systems, you had several commands that were used to configure the DOSPLUS system. Among them were CONFIG, RS232, and FORMS. Each of these had some manner (usually a "SAVE" parameter) of making the configurations in that command permanent. This has changed with version 3.5.

In the new version, we are no longer just concerned with saving some settings for the internal drivers, but also with saving the drivers themselves. If we did not save the drivers also, it would mean that each time you booted the system, you would have to re-install all the alternate drivers needed to support the functions desired.

For example, we will assume that you have installed the alternate printer driver in order to drive a serial printer. You have used the ASSIGN command to install the PR/DVR driver and specified all needed parameters. You have also adjusted the FORMS command to provide pagination of printer output. Everything functions fine. Then you reboot the system. If all that we saved was the settings on FORMS, before you could resume printer output, you would have to install the driver again and reset all needed parameters. This would quickly become <u>very</u> tedious.

Therefore, we now provide you with the ability to create <u>configuration files</u>. Please don't confuse these with the CONFIG command. They will save the CONFIG command's current settings, but they do a great deal more as well. When you create a configuration file, the system is saved as it is <u>at that moment</u>. This includes not only the settings of all the commands such as FORMS, CONFIG, RS232, and SYSTEM; it also includes any drivers that may have been installed.

All of this information, drivers and settings, is stored on the disk in a file. All you must do to regain the configuration and driver scheme as it was when you created this file is to execute the file. Execute it exactly as you would any other machine language program.

To summarize all of this, in the old system you would use various commands to customize the system and then you would permanently save these changes with the command. In DOSPLUS 3.5, this is not possible for two reasons. First, the drivers need to be saved also. Second, there are now more areas in which you can configure the system and to have them all save independently of each other would get confusing.

Therefore, picture a funnel. That is the manner in which the new system functions. All of the commands and all of the drivers funnel down to one act: creating the configuration file. When the system is set up just the way you want it, you will create the configuration file to be able to recall this later in a moment.

You may then, if you wish, put the name of this file on an AUTO statement so that your system configures itself automatically upon powerup. By using the invisible AUTO, you can even do this without having to observe its occurrence. Also, because you may have more than one command on an AUTO statement now, you can place the configuration file on an AUTO statement and still be able to execute other commands.

One final note. This technique will also be used to operate the hard disk drives. Once you have initialized the drive and, if desired, transferred system control to the hard disk, you will then create a configuration file containing this. After booting from your floppy disk, execute that configuration file and you will immediately be re-configured for the hard disk. You may also create as many of these files as you like, each with a slightly different configuration. You can change configurations quickly and simply by executing these various files. Although it is not a requirement, it may be useful to have at least one file on the disk with a "bare bones" configuration. That is, one file with only those items you consider essential to the operation of the system. This allows you to load this file and proceed from that point on in further configuration when creating new files. It prevents you from having to start completely over when designing a configuration file for a different function.

Please note that if items such as TRAP, JCL, or filters are in effect, they will also be saved in the configuration file. Also saved will be the current drive CONFIGurations and device names. In short, all areas of the system are saved with this file.

Creating a double density Model I master disk

All of the following assumes that you have hardware capable of operating in double density. This would include some form of double density adapter for the expansion interface unit.

Creating the actual disk to be used

The first and most important item to remember is to backup both of the Model I master disks you received before attempting ANY of the following. Only work with duplicate copies.

Version 3.5 of DOSPLUS is supplied on two single density system disks. This is so that anyone may boot and read them. It is no longer necessary to have a separate double density version. However, there is now a simple procedure that must be followed before a double density master disk can be created.

Note: Owners of the Radio Shack Double Density Disk Kit, catalog number 26-1143, must install a patch before beginning. This is described later.

The first step is to format the disk. Use the FORMAT utility to accomplish this. When FORMAT prompts you for the disk density respond with "D" for double. After you have formatted the disk, use the SYSGEN command to install the operating system on the disk you just created.

Once SYSGEN is finished, use the COPY command to copy all files from the single density master disk to the disk you just formatted and sysgened. Following that step, exchange the single density master disk currently in the system drive with the other master and repeat the copy so that all files on both disks are copied to the double density disk.

Once this is done, the process is complete. File this working master away in a safe place and make all your backups from it.

Example

Place the single density master disk labeled "A" or "SDEN" in the system drive and reset the machine. Place a blank disk in drive 1. Issue the command:

FORMAT :1

When FORMAT prompts you for disk density, respond with "D" for double. Answer all other questions according to your drive hardware (number of cylinders, number of sides, etc.). When format is complete, remove the disk labeled "A" or "SDEN" from drive 0 and replace it with the disk labeled with "B" or lacking the "SDEN" stamp. One that disk is in place, issue the command:

SYSGEN :1

SYSGEN will copy the system modules to the disk in drive 1. Once this is complete that disk will contain only the system files. To move the utilities and other files to this disk, issue the command:

COPY :0 :1,ECHO,SPW="PASSWORD",INV,NEW

This will copy all files from the single density disk to the double density disk just formatted and sysgened. This assumes that the password on the master disk is still "PASSWORD". If you have changed this, alter the statement accordingly. After this step is complete, replace the single density master disk in drive 0 with the other single density master (the one labeled "A" or "SDEN") and issue the command:

COPY :0 :1,ECHO,SPW="PASSWORD",INV,NEW

Note that you did not have to reboot, since both master disks are system disks. Also note that you could have simply typed a slash (/) and pressed ENTER after making the swap.

Once all of these steps are complete, the disk in drive 1 will be a double density master DOSPLUS diskette. You may make all future backups from this disk. This would have also allowed you to create a double sided or 80 track master at the same time, if you so desired.

Whatever manner of disk you formatted (double density, double sided, 80 track, etc.), when you have sysgened it and copied all files to it, that disk is now a DOSPLUS master disk. This very simple sequence of events allows you to create whatever type of master is desired from the two 35 track single density masters we supply.

Please note that if all you wanted to do was create a 40 track single density master (e.g. expand the cylinder count but not alter the density), this may be accomplished through using FORMAT and BACKUP. Format the destination disk to 40 tracks before beginning the backup and when BACKUP told you the disk contained data, reply with a "Y". This tells it to use the existing format. BACKUP will not de-allocate the last five cylinders.

The following notice applies to all of you DOSPLUS owners with the Radio Shack double density adapter. It is in regards to a patch required before DOSPLUS may be used with this kit.

NOTICE TO ALL MODEL I USERS

If you have a TRS-80 Model I with the Radio Shack Double Density Disk Kit, catalog number 26-1143, you must modify your DOSPLUS 3.5 for proper operation with the double-density adapter. To make this modification, type the following command from the DOSPLUS command level:

DO RS

This command will initiate a sequence of actions which will modify the DOSPLUS diskette in drive 0 for use with the Radio Shack double-density adapter.

Please note that these steps are not required for other "doubler" kits, and if the changes are made, other double density adapters may not function properly.

Example

Place a backup of the DOSPLUS Master diskette labeled "A" or "SDEN" in drive 0 and reset the system. From the DOS PLUS prompt, issue the command:

DO RS

This will cause that Master diskette to be patched for Radio Shack doubler operation. Press the reset button to reboot and load in the patched DOSPLUS system. After you have done this, follow the steps outlined above to create a double density Master disk.

If for any reason, you need the "B" disk (the one labeled "B" or lacking the "SDEN" stamp) to operate in double density also, you will have to first copy the file RS/TXT to the "B" disk from the "A" disk and then while the "B" disk is in drive 0, issue the "DO RS" command.

Note: You <u>must</u> reboot after making the patch. This is vital to the correct operation of the system. Then, while creating the double density Master disk, you should not reboot when switching between Masters. This is so that the patched system remains resident in memory until the actual double density Master is created.

Manual omissions

There are several items that were inadvertently left out of the DOSPLUS 3.5 User's manual as it was sent to you. In this part of the addendum, we shall cover some of these.

Pausing DIR, CAT, and FREE display outputs

There is no clear statement in the manual regarding what is used to pause the display output on the library commands DIR, CAT, and FREE. This is accomplished by pressing the SPACE BAR. The listing will pause and wait for you to press any key to continue.

Pressing the SPACE BAR will pause the output on many DOS commands and utilities such as LIST, MAP, etc. This replaces the SHIFT @ used in earlier systems. Please note that BASIC still uses SHIFT @ to pause program listings, however.

Screen printer option

When you have assigned the DOSPLUS alternate keyboard driver KI/DVR, you have the option of obtaining, at any time, a printout of what information is currently on the screen. This is accomplished by typing control-asterisk (*). The key sequence for this is SHIFT DOWN ARROW, and while still holding these two keys, pressing the asterisk (*).

This is the same for both the Model I and Model III editions of the driver.

LABEL/CMD

As discussed in the BASIC section of this manual, DOSPLUS 3.5 Extended Disk BASIC supports labeled line addressing. Briefly, this means that all BASIC commands that normally require line number references may accept label names instead of line numbers. Labels are assigned by placing the BASIC keyword NAME followed by the label, at the beginning of any desired line. For example, this BASIC program:

```
10 PRINT"(1) ENTER NEW ITEM"
20 PRINT"(2) DISPLAY EXISTING ITEM"
30 PRINT"(3) DELETE OLD ITEM"
40 PRINT
50 INPUT"SELECTION";X
60 ON X GOTO 1000,2000,3000
70 GOTO 10
1000
.
.
2000
.
.
3000
.
.
```

could be written, using label addressing, like this:

```
10 NAME MENU:PRINT"(1) ENTER NEW ITEM"
20 PRINT"(2) DISPLAY EXISTING ITEM"
30 PRINT"(3) DELETE OLD ITEM"
40 PRINT
50 INPUT"SELECTION";X
60 ON X GOTO ENTER,DISPLAY,DELET
70 GOTO MENU
1000 NAME ENTER
.
.
2000 NAME DISPLAY
.
.
3000 NAME DELET
.
.
```

This is, of course, a very convenient way to write BASIC programs. Unfortunately, most BASIC interpreters do not understand label addressing, and therefore programs using label addressing written under DOSPLUS 3.5's Extended Disk BASIC often will not run under other versions of BASIC (including TBASIC). The LABEL program is a label resolver; that is, the program will process a program containing labels and label references and create a program without labels. LABEL will replace all label references with line number references, and will delete all label names and associated NAME keywords from the source program.

To create a resolved BASIC program from a program containing labels, type:

LABEL filespec1 filespec2

where "filespec1" is the name of the BASIC program containing label references, and "filespec2" is the name of the file to contain the resolved BASIC program.

If the filenames are not specified on the command line when requesting the LABEL program, LABEL will prompt for the source and destination filenames.

When the LABEL program processes a BASIC program, it does so in two stages. The first stage, or pass, involves scanning the source program for all label definitions; that is, all of the line numbers containing the NAME keyword. LABEL builds a table in the computer's memory of all label names and the associated line number in which they occur. In the second pass, LABEL removes all label names, the NAME keyword, and replaces any references to a label with the line number in which the label was defined. If a reference is found to a label that does not occur in the program, the label is left unchanged in the program text, and the message:

Resolving line #:xxx   Undefined label:xxxxxx

is displayed on the screen, informing you of the line number in which the undefined label was found and the name of the label.

After the LABEL program is finished, the resolved destination BASIC program file may be used normally.

Disk Master password

The password on the DOSPLUS 3.5 Master disks as supplied from Micro-Systems Software is "PASSWORD".

This password may be required when performing certain global functions with library commands such as COPY, ATTRIB, PROT, and KILL. This password may also be substituted at any time for a regular file password.

Note: This should indicate to you the importance of assigning a Disk Master Password when formatting new disks. If the Disk Master Password is null (not set), then by not specifying the password, you have, in effect, given the password. This is because NO password IS the password. If you desire to protect any of the files on that diskette, you must assign a Disk Master Password. If one was not assigned during format, you may use the PROT command to assign a password.

File passwords

Many of the DOSPLUS utility and system files are protected with a password. This is to prevent you from accidentally deleting one of these files. If you wish to copy only a single utility or system file, though, the password will be the same as the file's extension. For example, the password on SYS0/SYS is "SYS" and on BACKUP/CMD, it is "CMD". By using the correct password, you may access any of the files that have protected status.

Standard files

There were many files included with your DOSPLUS 3.5. Some of these are covered in the manual and some are not. Most of the files that are not covered are patch files.

These files will have extensions such as /PPT or /FIX. These files will also usually be simply text files to be interpreted by certain DOSPLUS commands. Therefore, you can usually determine what the files do by using the LIST command to display them and reading any comment lines they may contain.

Files that have the /SYS extension are DOSPLUS system files. Files with the /CMD extension are usually DOSPLUS utilities. Files with extensions of /DVR and /FLT are drivers and filters and will be covered in that section of the manual.

Remember, if you have a question about one of these files (one not covered elsewhere), it is usually advisable to list it and see if it is self explanatory.

Manual errors

PRINT@ routine address incorrect

On pages T/23 and T/38 of the technical manual, it states that the address for the PRINT@ routine is 4462. This is incorrect.

The correct address is: 446A. This applies to both Model I and Model III.

Parameter on FILE/DVR incorrect

On page 6-10 of the DOSPLUS manual, it states that the SIZE parameter of the FILE/DVR program indicates the desired size of the pseudodisk in KILOBYTES. This is not true.

The SIZE parameter of the FILE/DVR program indicates the desired pseudodisk size in 256 byte RECORDS.

FILTER command misleading

On pages 2-79 and 2-80 of the DOSPLUS manual, the statement is made that the FILTER command assumes the extension /FLT. This is incorrect.

The FILTER command assumes no extensions whatsoever. We recommend that you use the /FLT extension to identify the filter files, but you will have to specify this extension when using FILTER.

I command inaccurate

On pages 2-90 and 2-91 of the library commands section, it states that the command I by itself will cause the DOS to recognize a change in the number of sides on a diskette at the next disk access. This is not the case.

In order to have the system recognize a switch in diskettes from single to double sided or vice versa, you <u>must</u> use the MOUNT parameter. The I command by itself will simply initialize the driver. With the standard floppy disk drivers, this will not affect a change in sides recognition.

DOS Operations

Introduction

Welcome to DOSPLUS 3.5! This unique Disk Operating System is based upon our belief that a program does not have to be confusing to be powerful. For those of you are used to our previous systems, DOSPLUS 3.5 will prove to be a bit of a surprise. This is perhaps the most device independent system ever designed for the TRS-80 Models I and III. This manual is designed to help you get acquainted with DOSPLUS 3.5 and is written in what (I hope) is 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 the area 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 utilities may be easily removed by the user (if not needed, of course), thus allowing the user to "customize" their DOSPLUS in the interests of disk space efficiency.

The Disk BASIC manual. DOSPLUS 3.5 has its own Disk BASICs. A Disk BASIC is a program that enhances the capabilities of the BASIC that is included with your TRS-80 in the ROM. DOSPLUS has two : **BASIC** and **TBASIC**. This section will cover each of them and then detail the differences.

The DOSPLUS 3.5 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 3.5 and other systems, though, (even earlier DOSPLUS') so we strongly suggest that you read the manual before beginning to use the system.

General manual syntax

This manual will follow 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. On the Model III, it is the orange button in the upper right hand area of the keyboard. For the Model I, it is located at the left rear of the keyboard next to the expansion bus. 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 3.5 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. After answering the date and time prompts, you will see the prompt "DOS PLUS" and a cursor appear.

This is called the "DOS command mode". Please remember this term, as we will refer to it often in this manual. This is the mode from which you will enter all commands to the system. It is from this mode that we will now issue the backup command.

Backing up with multiple drives

Place whatever diskette you wish to use for BACKUP in drive 1. It does not matter if the diskette is blank or not. From the DOS command mode, type BACKUP and press <ENTER>. The drive will engage, the backup program header will be displayed, 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 <u>N</u> and press <ENTER>. You may also simply press <BREAK> to abort (at <u>any</u> 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 may use BACKUP to backup between two non-system disks if you wish. Consult the utilities manual under **BACKUP** for details.

To continue with the backup and attempt to use the current destination disk format, type <u>Y</u> or <u>U</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 just re-format the entire disk.

To continue with the backup, but force BACKUP to re-format the destination disk first, type <u>F</u> 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 too many flaws such 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>. You should be returned to the DOS command mode. Your backup is complete. File your Master away in a safe location and use the disk you just made as your "working" Master.

<u>Backing up with a single drive</u>

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. From the DOS command mode, type <u>BACKUP</u> and press <ENTER>. 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 <u>FROM</u> 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 <u>TO</u> 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 <u>most</u> 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, type <u>N</u> and press <ENTER>. You may also simply press <BREAK> to abort (at <u>any</u> of the prompts). BACKUP will then flash the message :

Insert System disk (ENTER)

Place your Master diskette in drive 0 and press <ENTER>. You may use BACKUP to backup between two non-system disks if you wish. Consult the utilities manual under **BACKUP** for details.

To continue with the backup and attempt to use the current destination disk format, type <u>Y</u> or <u>U</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 just re-format the entire disk.

To continue with the backup, but force BACKUP to re-format the destination disk first, type <u>F</u> 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 too many flaws such 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>. At that point, you will be returned to the DOS command mode. 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 permanently stored 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 3.5 from earlier versions of DOSPLUS and from all other Model I/III 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. Remember, just because we let you do it, doesn't mean the rest of the systems will, too.

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, though, 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 rewarded with 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 <u>not</u> optional! An extension may be from 1 to 3 to three characters 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.

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 <u>required</u> part of the filespec, it is often used to more completely describe 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 <u>must</u> 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 <u>TEST</u> 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 them 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 <u>only</u> 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 we understand clearly what portions of the filespec contribute to the uniqueness of the filespec. If, for instance, you have written a BASIC program and we wish to store it on the disk, we must assign it a filespec. It is important to us 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 <u>not</u>. 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 :

<u>Filespec 1</u>	<u>Filespec 2</u>	<u>Same?</u>
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 data lost because of carelessly overwriting a previous file.

To recap an important point, remember that in filespecs the filename field is mandatory in <u>all</u> cases. All other fields are optional.

Device and drive specifications

The DOSPLUS system has eight character devices and eight 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
User defined	U1	User defined
User defined	U2	User defined

Drives :

First drive	0	Input or output
Second drive	1	Input or output
Third drive	2	Input or output
Fourth drive	3	Input or output
Fifth drive	4	Input or output
Sixth drive	5	Input or output
Seventh drive	6	Input or output
Eighth drive	7	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 is to say 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 is to say 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. 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, even if the devices are of different types (character/file).

Any time that you refer to a device, no matter what sort of operation you are performing, you will use the devicespec. It is very important that you, if you decide to rename devices, remember what names you have assigned 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 we continue to remember that, we shall have no problems with specifying an illegal I/O path for the data to move on.

When specifying the I/O path, we 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. So you see, 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.

<u>Detailed explanation of the command line</u>

The command line is the means by which you communicate with DOSPLUS. When you are at the DOS command mode (remember that term?), you may enter up to 64 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.

<u>The command</u>. 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, otherwise DOSPLUS will assume that you have entered a program name. 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.

<u>The I/O field</u>. 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 delimiter words FROM, TO, and USING respectively. Each of these portions of the I/O field must be separated from their delimiters and each other by a space. You may omit the delimiter 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 TESTI/CMD:1

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

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

is NOT the same thing as :

    COPY TEST1/CMD:1 TEST/CMD:0

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 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 an entire drive's worth if files. It saves keystrokes and is more convenient. For example :

> T!

is the same as :

> T*/* or 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". A very useful character. DOSPLUS is signaled that a wildmask is present whenever: (1) the USING delimiter precedes the wildmask, (2) the wild mask 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 on including 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. Within the parameter field, though, you may use either a space OR a comma. 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 in any base 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. These are 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, either.

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 please 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.

Definition of terms (cont.) :

Term	Definition
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.
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 specified 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).

Definition of terms (cont.) :

Term	Definition
Separator	Used to separate delimiters and channels, parameters, etc. Within the I/O field, separators MUST be a space. Within the parameter field, they may either be a space or a comma. If you use commas within the I/O field, DOSPLUS will terminate the I/ID 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, TO, 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.

Library Commands

DOSPLUS 3.5 Library of commands

    The following are the library commands for DOSPLUS 3.5. 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)	2-71
DUMP	(Save memory to disk file)	2-75
ERROR	(Display detailed error message)	2-77
FILTER	(Filter I/O to/from specified device)	2-78
FORCE	(Re-direct I/O to device/file)	2-82
FORMS	(Alter printer pagination parameters)	2-84
FREE	(Display free space data)	2-88
I	(Initialize disk drive)	2-91
JOIN	(Link two logical devices)	2-93
KILL	(Kill specified device or file)	2-96
LIB	(Display list of library commands)	2-100
LIST	(List file to device)	2-101
LOAD	(Load disk file into memory)	2-103
PAUSE	(Pause execution)	2-105
PROT	(Alter disk's protection status)	2-106
RENAME	(Rename a device or file)	2-109
RESET	(Restore device to default driver)	2-110
RS232	(Display/alter serial port settings)	2-111
SCREEN	(Send contents of screen to device)	2-115
SYSTEM	(Customize your operating system)	2-116
TIME	(Display time or set system clock)	2-123
VERIFY	(Toggle automatic disk verification)	2-124

APPEND

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

==

The command syntax is :

   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 linking together 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.

   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 created by standard SCRIPSIT 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.

   APPEND can also be used as a sort of dynamic disk merge. 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.

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 <u>it</u> 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.

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 <u>from</u> an <u>input</u> device and to an <u>output</u> device. If you are not certain what a particular device is, use the FORCE or JOIN command 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 1=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 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 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.

Finally:

    Remember, APPEND will <u>never</u> affect the source device or file.

    Also, unless you specify the CMD option, the appendage will always be saved in data file format. Machine language appendages MUST be appended with the CMD option. There simply is no choice.

    Please remember that 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 operations section of this manual. A disk file may function as either input or output.

ASSIGN

This command allows you to install a driver program for any device or drive.

==

The command syntax is

ASSIGN [FROM] devicespec/drivespec [TO] filespec
ASSIGN [FROM] devicespec1/drivespec1 [TO] devicespec2/drivespec2

devicespec/drivespec is the name of the device or drive for which you intend to install the driver (i.e. @KI, @DO, @PR, :0, :1, etc.).

filespec is the name of the driver program.

devicespec1/drivespec1 is the name of any active device or drive whose information will be installed for devicespec2/drivespec2.

==

The ASSIGN command gives you the ability to install non-standard drivers for any device or drive in the system. This allows you to operate with hardware other than that supplied by Radio Shack (i.e. hard disks, eight inch drives, etc.).

A driver is simply a machine language program that controls I/O to or from a particular device. Depending upon the nature of the physical equipment, the driver program may also input certain control signals from the peripheral to govern its operation, as in the case of a serial printer driver. Such a driver may cease output when the peripheral signals "printer not ready" or "out of paper". Other driver programs can be used to control input devices such as the keyboard or the serial interface. These drivers can then process any data coming in from these devices. Therefore, you can accurately refer to a driver program as being an interface between a physical piece of equipment and the DOSPLUS Disk Operating System.

There are two basic forms of the ASSIGN command. In the first, you are actually installing the driver in memory from a file. In the second, you are simply setting two devices to the same driver. The advantage to the second method is that it does not use the additional memory to load the driver in a second time. Obviously, though, the driver must have been loaded in at some point for the device to function at all.

The DOSPLUS system comes standard with default drivers already established for the main system devices (@KI, @DO, @PR, and the disk drives). Generally, these drivers are suitable for a wide variety of situations. However, a situation may arise where the system's driver is inadequate or incompatible with a particular piece of equipment. Such a situation may be when a line printer using non-standard symbols is attached to the system, or when you need to do I/O to the serial port, or when you wish to attach a hard disk drive. In such a case, a new driver must be installed for the affected device. Such driver programs may be provided with the system, or they may be specially written for the user's purposes. Information is provided in the technical manual on how to write driver programs of your own and interface them to DOSPLUS.

Driver programs are stored on the disk just as any other file and are called via the ASSIGN command when they are needed. When you ASSIGN a driver, you must specify both the devicespec and the filespec of the driver program. Wildmasks are not valid. You must ASSIGN each driver independently of any others. No extension will be assumed on the filespec, but we recommend that you use "/DVR".

When you use the first form of the ASSIGN command, loading the driver from the disk, the driver program will load into high memory. It will adjust the high memory pointer downward to protect itself from being overlaid by other routines. This is at least the case with all drivers supplied by us. We cannot attest to how other drivers will be written. The driver programs are also fully relocatable. If another program is already in high memory and has adjusted the high memory pointer accordingly, the driver will load in below it and avoid memory conflicts.

When a driver is installed for a device by means of the ASSIGN command, any previous forcing or joining of the device will be reset. However, any filtering or translating established for the device will remain in effect. Any drivers that are in effect at the time that a configuration file is saved (see SYSTEM) will be saved with the file and restored when that file is executed.

If further devices are going to use the same driver program, to use the first form of the command would waste memory by loading the driver again from the disk. To avoid this, use the second form of the ASSIGN command. When you use this form, the driver is not re-loaded. The second device is simply set up to use the same driver as the first. The only memory used is that which is required to set up the new device (only a few bytes). This represents a great saving of memory.

For example, if you ASSIGNed drive 4 to the driver RIGID/DVR (we are assuming that drive 4 is a hard disk here), and you wished to also use drives 5, 6, and 7 as hard disks, you would merely assign each of those drives to drive 4 (i.e. ASSIGN :5 :4, ASSIGN :6 :4, etc.). This would install each of those drives to the one copy of the driver that was loaded when we called it for drive 4 (i.e. ASSIGN :4 RIGID/DVR).

Please note : After you duplicate a driver for two drives in this manner, the second drive will share the same CONFIG settings as the first. This includes the physical drive number (see CONFIG). You may have to alter this to reflect the proper values.

Whenever you ASSIGN a device or drive to itself (i.e. ASSIGN :4 :4, ASSIGN @PR @PR), you will re-install the current driver defined for that device or drive. In the case of the character devices, this will cause any FORCEs or JOINs to be reset. In the case of both devices and drives, if they have been killed (see KILL) and are in the NIL state, it will cause them to be re-activated. If that device or drive never had a driver ASSIGNed to it, then it will report an error if you attempt to ASSIGN it to itself.

Each driver that Micro-Systems Software provides for DOSPLUS comes with specific instructions as to the installation and operation of the driver. When you receive your DOSPLUS, if you receive any non-standard drivers with it, you should have such documentation. There is a section of this manual called **Standard I/O drivers** that contains information on all drivers sent with your DOSPLUS.

Parameter passing with the ASSIGN command

Certain specialized drivers may require the user to pass it a set of parameters at installation time. These parameters could include, but are not limited to, additional filespecs. What these parameters are will depend on the driver program itself. For instance, the alternate keyboard driver could require a filename indicating where the MacroKey definition file is stored (see Standard I/O drivers : DOSPLUS alternate keyboard driver).

You pass the parameters to the driver just as you would to any library command. After you enter the I/O field (e.g. the FROM and TO fields), you simply enter the parameter list. You must separate this from the I/O field by either a comma, a space, or a left parenthesis. For example, if you chose to use the MacroKey option on the keyboard driver and you had your MacroKey definitions stored in a file called KEYS/DAT, when you ASSIGNed the driver it should look something like this :

ASSIGN @KI KI/DVR KEYS/DAT

If you wanted to install a printer driver that allowed an indent parameter and you needed to pass the driver a value indicating the amount of characters to indent, it might look something like this :

ASSIGN @PR PRTR/DVR (INDENT=20)

The parameter field must be terminated by a carriage return, an implied carriage return (e.g. a semi colon), or a right parenthesis. It will be up to the individual driver program 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 FROM @PR TO PR/DVR
ASSIGN PR PR/DVR

This will install the driver program PR/DVR for the device @PR (the printer). Any previous FORCEs or JOINs would be cancelled. Any needed parameters could have been passed after the PR/DVR filespec.

ASSIGN :4 RIGID/DVR

This will load from the disk and install the driver program RIGID/DVR for drive 4. All I/O to or from drive 4 would then be controlled by that driver.

ASSIGN :5 :4

This would install whatever driver was currently defined for drive 5 to drive 4 also. As discussed earlier, this has the advantage of conserving memory when more than one device will be operating from the same driver.

        ASSIGN FROM :1 TO :1
        ASSIGN :1 :1

        This will re-install the driver currently defined for drive 1. If there never was a driver defined for that drive, an error will result. If that drive had been killed, it will now be restored.

Finally:

        Please note that there is no standard system driver ASSIGNed to the serial interface. Before that can be implemented, the driver (included with your system) must be installed.

        Also please note that none of the standard drivers will support filtering. If you wish to implement a filter, you MUST install one of the non-standard drivers for that device first.

        Once a driver is installed, you may remove it by loading a configuration file (see SYSTEM) that does not have the driver installed or re-booting the system.

ATTRIB

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

===

The command syntax is :

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

<u>filespec</u> specifies the file that you wish to alter the attributes of.

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

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

The parameters for ATTRIB 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.

Abbreviations :

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

===

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. If there will be any protected files included in this, you will have to specify the disk's Disk Master Password using the PW parameter.

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.

<u>Controlling a file's protection level</u>

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. The default password for any file is a series of blanks. This is also the default when no password is specified with the filespec. Therefore, when the user omits the password, they have in actuality SPECIFIED the correct password and they are allowed full access to the file.

Remember also, the ACCESS password controls access to the file per your protection level. In other words, if an access password has been set, the user needs that password to even get at the file. Once they have the password, they may access that file only up to the limit that you have assigned. However, if they know the UPDATE password, it will allow full access to the file even if a protection level has been set. Therefore, keep your update passwords in closer confidence than your access passwords.

Also 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. To alter the Disk Master Password, use the library command PROT (see the library command PROT).

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 :

<u>Number</u>	<u>Protection level</u>
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.

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

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

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

<u>Protection level 4</u>, 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.

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.

Under DOSPLUS 3.5, 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.

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.

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.

The final parameter that you can alter with ATTRIB is the modification flag (MOD FLAG for short). This is one of the most useful items in DOSPLUS' directory. 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.

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

These three commands will have the same effect. 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 or in any way examine the code without using the update password. This example assumes that no previous password existed.

        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.

Finally:

When using ATTRIB, please keep in mind that if a file already has some protection (a password and a protection level set) on it, you MUST use this current password when accessing the file to alter its attributes.

Also bear in mind that when you specify a filespec WITHOUT a drivespec, it will do a global search of all drives in the system looking for that filespec. The first one that it locates will have the prescribed action performed on it, and ATTRIB will stop at that point. On the other hand, if you use a wildmask, the effect will be global on all files matching that wildmask, but ONLY on the drive specified. If a drive is NOT specified, then ATTRIB will assume that the system drive is to be used.

For example :

ATTRIB FILE,I

will search until it finds FILE and make that file invisible. But :

ATTRIB FILE/*,I

will search out all files that have the filename FILE and any extension and make them invisible, but it will only do this for the system drive.

If a file already has a password set for it and you will be operating upon that file in a global ATTRIB, you will need to indicate the Disk Master Password. That was mentioned earlier, but deserves a bit more emphasis.

Any file that has a password assigned to it must have that password specified when you are changing the file's attributes. This is not possible when doing a global ATTRIB, because all the files can have potentially different passwords.

It is because of that we have the PW parameter. DOSPLUS will always allow you to specify the Disk Master Password in place of the file password.

AUTO

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

===

The command syntax is :

    AUTO [drivespec] [command line]

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

        command line is the command line that you wish executed upon power-up. The first character (or two) can be an optional switch to indicate the type of AUTO.

Command switches :

    !   Invisible AUTO.
    *   Non-breakable AUTO.

===

The AUTO command allows you to set a command that will be automatically executed whenever the system is booted (unless the ENTER key is held down as the system is booting). This command 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 32 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, lets 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.

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.

Finally:

Perhaps one of the most useful aspects of AUTO is to execute a "system configuration" file. For a detailed explanation of these files, consult the Section **Customizing your DOSPLUS** in the operators portion of the manual. 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.

BOOT

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

==

The command syntax is :

BOOT

There are no parameters for this command.

==

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

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.

One additional option is to hold down the shift and up arrow keys while the system is booting. This will cause DOSPLUS to initialize with only the unmodified ROM drivers in effect. There will be no modifications made by the operating system.

BREAK

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

===

The command syntax is:

BREAK [param]

param 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) occurs 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.

==

The command syntax is :

    BUILD devicespec/filespec (param=switch...)

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

              <u>(param=switch...)</u> is the optional parameter to modify the command's action.

Your parameter is :

    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.

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 (63 chars/line)

At that point you are free to type up to 63 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. Should you press BREAK without pressing ENTER, the line will be stored in the file without a terminating carriage return.

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.

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.

<u>Sample use</u>

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 (63 chars/line)
FORMS (W=80) <ENTER>
BASIC MENU/BAS-F:1-M:65000 <ENTER>
<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.

Finally:

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 he 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.

CAT

This command will display a disk's file catalog.

===

The command syntax is :

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.

Abbreviations:

SYSTEM	S
INVIS	I
KILL	K
ALPHA	A

===

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.

<u>The file catalog display</u>

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

Drive: 1   DOS 3.5   - Space: 085/128   82.5k

PATCH/CMD	TAPE/CMD	KBD/DVR	CODIR/CMD
RFORMAT/CMD	DIRCHECK/CMD	HELP/CMD	BACKUP/CMD
WD/DVR	ERROR/OVL	DISKDUMP/CMD	JCL/RLD
MAP/CMD	FORMAT/CMD	MENU/JCL	SYSGEN/CMD
DIR/DVR	JCL/CMD	TRAP/CMD	TBASIC/CMD
BASIC/CMD	DISKZAP/CMD	CONVERT/CMD	RESTORE/CMD

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.

<u>Specifying output files and devices</u>

When using CAT, if you wish to specify an output file or device other than the display and you have NOT specified a source channel, 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 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.

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.

Finally:

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.

Although overwriting files by accident was mentioned earlier, it deserves to be strengthened now. The 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.

CLEAR

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

==

The command syntax is :

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

               <u>filespec</u> 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.

    The START 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.

    The END 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.

The DATA 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.

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 5C00H 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=6000H,END=7000H,DATA=6CH)
CLEAR (S=6000H,E=7000H,D=6CH)
CLEAR,S=6000H,E=7000H,D=6CH

These three commands are equivalent. All three of them will fill memory between addresses 6000H (24576 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).

Finally:

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.

As a note to users of DOSPLUS 3.4, CLEAR used to be the command you used to set the high memory address. You will now use SYSTEM.

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 <u>always</u> 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.

CLOCK

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

===

The command syntax is :

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.

===

The command syntax is :

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 32 character per line format (e.g. double wide text), you will be restored to the standard 64 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 for non-standard disk drives.

==

The command syntax is :

    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.
PD=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.
PD=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
PD	P

Rigid Drives :

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

===

   The CONFIG command allows you to configure your DOSPLUS to operate correctly with all manner of non-standard disk drives. CONFIG will allow you to set any parameter for any drive. It is up to the driver 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.

<u>Floppy disk drives</u>

   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,PD=0,MD
        $01 :1 Floppy,Dden,Size=5,Sides=1,Step=3,PD=1,MD
        $02 :2 Floppy,Dden,Size=5,Sides=1,Step=3,PD=2,MD
        $03 :3 Floppy,Dden,Size=5,Sides=1,Step=3,PD=3,MD
        $04 :4 NIL
        $05 :5 NIL
        $06 :6 NIL
        $07 :7 NIL

   Note : The above settings are the standard default settings for DOSPLUS 3.5. 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 <u>drive device number</u>. As you can see from the numbers 0 through 7, there are 8 drive devices in the DOSPLUS 3.5 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.

Second is displayed the <u>drivespec</u>. 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 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 3.5 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 "Single or Double sided?". If you respond with a "D", 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. The first time that DOSPLUS accesses a drive, it will pick up this parameter. However, should you switch disks, the system would have to be informed that a different disk is in the drive. This can be accomplished two ways.

First, you could use the I command to "init" the drive so that the next time DOSPLUS accessed that disk it would know it had to re-read the DCT information because something had been changed. Second, you could use this parameter on CONFIG to inform the system directly that a double sided disk is now resident in that drive.

Therefore, whenever you are switching between single and double sided disks in the same drive, you will need to take some action to inform the system that this has occurred. This parameter is one of the ways of doing this. 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 <u>not</u> double sided. If you are not specifically aware of the fact that you have double sided disk drives, you probably do not.

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

PD

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 PD 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 PD 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 (PD=1)
                                  CONFIG :1 (PD=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 CONFIC 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 <u>still</u> 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.

<u>Caution!</u> 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 excellent 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,PD=0,MD
$01 :1 Floppy,Dden,Size=5,Sides=1,Step=3,PD=1,MD
$02 :2 Floppy,Dden,Size=5,Sides=1,Step=3,PD=2,WP,MD,HL,Skip
$03 :3 Floppy,Dden,Size=5,Sides=1,Step=3,PD=3,MD
$04 :4 NIL
$05 :5 NIL
$06 :6 NIL
$07 :7 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 parameter. For more specific information and exact syntax's 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,PD=0,MD
$01 :1 Floppy,Dden,Size=5,Sides=1,Step=0,PD=1,MD
$02 :2 Floppy,Dden,Size=5,Sides=1,Step=0,PD=2,MD
$03 :3 Floppy,Dden,Size=5,Sides=1,Step=0,PD=3,MD
$04 :4 Hard,Fix,Size=5,Sides=0,Step=6,PD=0,CO=0,HO=0,TS=0
$05 :5 NIL
$06 :6 NIL
$07 :7 NIL
```

Note : The above example is of a standard DOSPLUS 3.5 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 <u>drive device number</u>. As you can see from the numbers 0 through 7, there are 8 drive devices in the DOSPLUS 3.5 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.

Second is displayed the <u>drivespec</u>. 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 :

<u>Rigid disk parameters</u>

Hard

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

PD

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 PD 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 (PD=0)
                            CONFIG :6 (PD=1)
                            CONFIG :5,PD=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, CONFIC 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 PD 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 : <u>None</u> 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,PD=0,CO=0,HO=0,TS=32
$01 :5 Hard,Fix,Size=5,Sides=6,Step=6,PD=0,CO=115,HO=0,TS=32
$02 :6 Hard,Fix,Size=5,Sides=6,Step=6,PD=0,CO=172,HO=0,TS=32
$03 :2 Floppy,Dden,Size=5,Sides=1,Step=0,PD=2,MD
$04 :1 Floppy,Sden,Size=5,Sides=1,Step=0,PD=1,MD
$05 :0 Floppy,Dden,Size=5,Sides=1,Step=0,PD=0,MD
$06 :A NIL
$07 :B 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 <u>back</u> 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-existent 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.

===

The command syntax is:

    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 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 :

<u>Mode 1</u>

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 <u>from</u> an input device <u>to</u> 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 <u>differing</u> 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 <u>same</u> 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).

<div align="center">

Mode 2

</div>

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 <u>not</u> specified, but rather assumed to be unchanged from the first, you may <u>not</u> 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 <u>close</u> attention to the prompts and only insert the proper disks at the proper times.

<u>Mode 3</u>

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 3.5 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 <u>must</u> 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 <u>not</u> 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 <u>never</u> 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 <u>not</u> errors, but rather the command has fairly sophisticated error trapping.

CREATE

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

==

The command syntax is :

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	C
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 come up to the specified amount of total disk space.

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

<u>Verification of data</u>

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 <u>always</u> 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 <u>not</u> 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.

===

The command syntax is :

    DATE
    DATE mm/dd/yy or mm/dd/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 3.5 owners in the year 2000 to change the base year.

Examples:

        DATE 1:27:83
        DATE 01:27:83
        DATE 01:27:1983
        DATE 1-27-83
        DATE 1 27 83
        DATE 1.27.83
        DATE 01/27/83
        DATE 1,27-83

    All of these commands are equivalent and will have the same exact effect. They will set the system date to January 27th, 1983.

        DATE 9

    This would set the month in the system date to September. If you do not specify any other fields, the current values remain unchanged.

DEBUG

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

==

The command syntax is :

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 BREAK at any time (SHIFT-BREAK on the Model III).

(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	Exit to DOSPLUS (DEBUG still engaged)
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):

```
AF  = 0044 -Z---P--
BC  = 0200: 64 79 0D 1E 3D AF C9 3E  0D CD 3B 00 AF C9 7E 23
DE  = 2000: 2D 38 02 1F 26 C3 A2 19  11 0A 00 D5 28 17 CD 4F
HL  = 3FC0: 64 36 30 30 30 20 35 32  33 30 3A 20 20 30 37 20
AF' = D380 S-------
BC' = 0000: F3 AF C3 15 30 C3 00 40  C3 00 40 E1 E9 C3 12 30
DE' = 4314: FE 41 20 13 CD EC 51 7E  FE 20 38 04 FE C0 38 02
HL' = 58C1: 20 5B 45 4E 54 45 52 5D  0D FF FF FF 10 04 04 10
IX  = 4010: 07 73 04 16 3E 20 5F 00  90 44 FC 43 00 00 FF 52
IY  = 446C: 25 40 FF FF FF FF 66 FC  02 50 52 08 45 FF FF FF
SP  = 41D3: 10 40 A6 57 00 43 3F 3F  69 06 41 01 00 43 6A 4E
PC  = 0694: D1 DD E1 E1 C1 C9 AF 32  9F 40 16 FF C3 8D 28 E6
        6000: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        6010: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
        6020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

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) :

```
5B00:   6E 79 74 68 69 6E 67 20   79 6F 75 20 77 69 73 68
5B10:   2E FD 0D 62 0D 03 02 EF   FD 0D 62 0D 03 02 EF F2
5B20:   4E 6F 77 2C 20 61 66 74   65 72 20 72 65 62 6F 6F
5B30:   74 69 6E 67 20 74 68 65   20 73 79 73 74 65 6D 2C
5B40:   20 74 6F 20 6C 6F 61 64   20 74 68 65 20 64 72 69
5B50:   76 65 72 73 20 77 65 20   68 61 64 20 61 73 73 69
5B60:   67 6E 65 64 20 61 6E 64   20 69 6E 73 74 61 6E 74
5B70:   6C 79 F8 72 65 74 75 72   6E 20 61 6C 6C 20 74 68
5B80:   65 20 69 74 65 60 73 20   77 65 20 68 61 64 20 63
5B90:   6F 6E 66 69 67 75 72 65   64 20 74 6F 20 74 68 65
5BA0:   20 76 61 6C 75 65 73 20   77 65 20 73 65 74 20 74
5BB0:   68 65 6D 20 74 6F 2C 20   61 6C 6C 20 77 65 20 68
5BC0:   61 76 65 20 74 6F 20 64   6F F8 69 73 20 65 78 65
5BD0:   63 75 74 65 20 74 68 65   20 66 69 6C 65 20 40 4F
5BE0:   44 31 2F 43 46 47 2E FD   0D 62 0D 03 02 EF FD 0D
5BF0:   62 0D 03 02 EF F2 54 68   69 73 20 63 61 6E 20 62
```

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) :

```
A000:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A010:   .   .   .   .   .   .   .   .   .   _   .   S   C   R   I   P
A020:   S   I   T   /   C   M   D   .   .   .   .   .   b   .   .   .
A030:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A040:   .   .   .   .   .   .   .   .   .   .   .   .   .   M   I   S
A050:   S   P   E   L   L   /   C   T   L   .   .   .   .   .   .   .
A060:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A070:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A080:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A090:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0A0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0B0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0C0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0D0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0E0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
A0F0:   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
```

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.

==

The command syntax is :

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

<u>drivespec</u> is the optional drivespec indicating which disk's file directory to display.

<u>device/file</u> is the optional output device or file.

<u>wildmask</u> is the optional wildmask to restrict DIR to a certain group or class of files.

<u>(param=exp...)</u> 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.

Abbreviations :

```
SYSTEM  S
INVIS   I
KILL    K
ALPHA   A
```

==

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 file directory display

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

```
Drive: 1        DOS:3.50 - Space: 080/128    67.5k
Filespec            Attrib      LRL      #Secs  #Recs  Space  Updated

BACKUP/CMD          6P..I..     256      12     12     3.0k   01/26/83
DIR/SYS             5PS.I..     256      18     18     4.5k   01/26/83
DISKZAP/CMD         6P..I..     256      13     13     4.5k   01/26/83
DO/DVR              6P....+     256      2      2      1.5k   01/26/83
FILE/DVR            6P.K.!.     256      6      6      1.5k   01/26/83
SYS0/SYS            6PS.I..     256      15     15     4.5k   01/26/83
SYS2/SYS            6PS.I..     256      5      5      1.5k   01/26/83
TRAP/CMD            0           256      3      3      1.5k   01/26/83
```

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 directories.

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 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:3.50"). 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 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).
> * 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 <u>always</u> 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. 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
        <u>extremely</u> 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).

The logical record length. For machine language programs and data files that are to be accessed by machine language programs, 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 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 "01/01/80" 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.

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.

Finally:

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.

Although overwriting files by accident was mentioned earlier, it deserves to be strengthened now. The 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.

DO

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

===

The command syntax is :

DO filespec (param=exp...)

> <u>filespec</u> is the standard DOSPLUS file specification that indicates what file the commands to be executed are stored in.
>
> <u>(param=exp...)</u> 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.

The BREAK parameter allows you to set up whether the break key may be used to abort the DO file at PAUSE prompt (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.

The HIGH 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 <u>none</u> of the DOSPLUS drivers exhibit this problem.

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 this 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.

<u>Applications of DO</u>

There are many areas that DO is 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-F:7).

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.

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.

Finally:

  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.

  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.

  When using DO from BASIC, you <u>must</u> 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.

DUMP

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

===

The command syntax is :

    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 TESTEILE (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 hexadecimal.

          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.

==

The command syntax is :

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 TRSDOS compatibility and 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.

FILTER

This command allows you to set up a "filter" on any of the character orientated system devices.

==

The command syntax is :

FILTER [FROM] devicespec [To] filespec (param)

devicespec is the name of the device you wish to filter.

filespec is the name of the file that contains the filter.

param is the optional action switch.

Your switches are:

YES — Engage filter after loading (assumed). Also can be used to engage previously loaded filters.

NO — Load, but do not engage filter. Also can be used to deactivate a previously loaded filter.

MAP — Display filter codes.

==

This command is used to install device filters. These filters are used to modify data as it passes between the device and its driver program. DOSPLUS was designed in such a manner as to make this filtering easy. Filter files are simply stored as ASCII lists on the disk and the system will interpret which codes get translated to what new values.

DOSPLUS comes standard with several filters. These are explained in the section on System and BASIC enhancement files. You will find that most of these filters are used to provide enhancement or alteration of existing functions. Filters are NOT drivers. They simply modify the data as told. Device drivers are another thing altogether and are installed via the ASSIGN command (see ASSIGN).

A good example of using a filter is the DVORAK keyboard. If you were to re-arrange the keys on the keyboard into the DVORAK formula, that would only accomplish half the task. For example, the "Q" key now has a "D" label, but when you press "D", "Q" shows up on the screen. Therefore, you have need of a filter.

The device to filter in this case is the keyboard. We wish to alter the data BEFORE the system evaluates it. We could simply filter any "Q"s that went to the display into "D"s, but then although we would see a "D", DOSPLUS will still regard it as a "Q". Therefore, we must filter the data after it leaves the input device and not before it goes to the output device. In other words, the filter goes on the keyboard.

In this filter file that we install on the keyboard, we would instruct the system to translate any "Q"s into "D"s. The end result is this. You press the key labeled "D" which is really the "Q" key. A "Q" is sent to the filter where it gets translated to a "D" and returned to the keyboard driver which sends it to the system. In effect, the "Q" key has really become the "D".

For your convenience, a DVORAK keyboard filter is included with DOSPLUS. Should you have such a keyboard, all you would have to do is issue the statement :

FILTER @KI DVORAK

and the rest would be taken care of. Whenever you create a configuration file with the SYSTEM command, all current filters are saved with it and re-installed automatically when the configuration file is called.

<u>Writing a filter file</u>

Some other systems may provide you with filter capability, but they almost ALWAYS require some form of at least moderately comprehensive assembly language filter/driver combination. DOSPLUS will not require this of you. Filter files are nothing more than ASCII text files.

The first step is to list the code you wish to translate. This may expressed as a hexadecimal value or it may be a quoted literal. For instance, if you wanted to translate the letter "A", you could search for the codes :

41 hex
"A" quoted literal

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.

Finally:

To LOAD a filter without engaging it, use the switch with the name of the filter. For example :

FILTER @DO DISPLAY (NO)

will load the filter DISPLAY/FLT into memory and install it on the video device, but because of the NO parameter, it will not engage it. You may then turn the filter on later. This "pre-loading" of filters can come in handy as an easy way of getting all needed memory reserved for filters before loading BASIC or your applications program.

Once a filter is installed on a device, you may remove it by loading a configuration file that does not have the filter in it or by rebooting the system. You may turn the filter "Off", but this does not remove it or reclaim the memory.

If another filter is used on the same device, the previous memory will be reclaimed and used, if possible.

FORCE

This command allows you to route the I/O of one DOSPLUS system device to another.

===

The command syntax is:

    FORCE
    FORCE [FROM] devicespec [TO] device/file

           <u>devicespec</u> is the primary device which is to be routed.

           <u>device/file</u> is a device or a file <u>to</u> 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 <-   4DCAH
    $01 @DO <-> 0473H
    $02 @PR   -> 03C2H
    $03 @RS   -   NIL
    $04 @U1   -   NIL
    $05 @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).

<u>Restrictions</u>

(1)   Only devices 0-5 can be the primary device. These are the system devices @KI, @DO, @PR, and @RS, 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 <u>to</u> 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 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.

==

The command syntax is :

    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 <u>not</u> 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.

Abbreviations:

PAGE	P
LINES	L
WIDTH	W
TOP	T
CODE	C

==

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.

Please note that certain functions may not be supported without the installation of the DOSPLUS alternate printer driver supplied on your Master diskette. The resident printer drivers in the Radio Shack ROMs do not support the LINES and WIDTH parameters on the Model I and the LINES parameter on the Model III. For your convenience, we have supplied an alternate printer driver that supports all of these parameters fully as well as adding certain advanced support features such as a spooler, line indenting, serial printer support, and much more. Consult the actual driver documentation for details. You may install this as needed (see Drivers and Filters).

The PAGE 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.

One additional note. In the Model III, if you are using the resident ROM driver, it will require you to store the number of lines per page as number of lines per page plus one. Therefore, if you are using standard 66 line per page paper, you would set the PAGE parameter to 67. If you are using the alternate printer driver, this will not be the case. Lines per page are stored exactly.

The LINES 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. Please note that this option is not supported under either Model I or Model III unless the alternate driver is installed. This option is simply not honored by the resident drivers in the Radio Shack ROMs.

The WIDTH 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 attempt to output more than the limit of characters. Since the Model I resident ROM driver doesn't use this parameter, there will be no conflict there. On the Model III, WIDTH defaults to 255. After that point is reached, text will be wrapped around to the next line. To avoid this problem on the Model III, if you are using the alternate printer driver, you may specify WIDTH=0 and the driver will also never automatically wrap a line around no matter how many codes are output prior to a carriage return.

Please note that this parameter is not recognized at all on the Model I without the installation of our driver and one the Model III, you must store the value here at characters per line plus two. This will, of course, be both active and exact under the alternate driver.

TOP 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.

The CODE 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 <u>any</u> codes you desire, this command is <u>totally</u> 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.

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".

Finally:

    A note to previous owners. The operation of the FORMS command is now slightly different in version 3.50 than in earlier versions. Please look in the "Dos operations" portion of the manual for the section on items that have changed and note the differences.

    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.

FREE

This command will display the free storage space and remaining directory space on all mounted disks.

===

The command syntax is:

    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.

When you request FREE without a drivespec, you should see something similar to this :

```
Drive: 4   Volume 1  - Space: 175/256      3096.0k
Drive: 5   Volume 2  - Space: 238/256      3012.0k
Drive: 6   Volume 3  - Space: 254/256      3440.0k
Drive: 7   Volume 4  - Space: 254/256      3440.0k
Drive: 0   Scripsit   - Space: 071/128       16.5k
```

As covered above, the first item displayed is the drivespec and the disk name. Second is the amount a free directory space remaining. The first number is the amount a 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 <u>with</u> 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.

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.

Finally:

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.

I

This command will instruct the system to log in a disk.

===

The command syntax is:

    I [drivespec] (param=switch)

> <u>drivespec</u> is the specifier of the name of any valid drive in the
> system.
>
> <u>(param=switch)</u> 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.

This is necessary <u>only</u> when you are using double-sided floppy disk drives and wish to swap a single-sided disk for a double-sided one, or vice versa. For all other items about a disk, it is optional. DOSPLUS needs to know whether a disk is double-sided or not. The I command will allow it to determine this information. If all your disks are formatted identically regarding the number of sides, then it is not necessary to use the I command at all.

The MOUNT parameter will cause the system to immediately read the information from the specified drive into memory without waiting for the next disk access.

You may also manually adjust the SIDES parameter on CONFIG if you wish, but use of the I command will accomplish the same results with less user effort. Put simply, all the I command does is "initialize" the disk from the DOS' point of view. When you use the I command, the DOS "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.

The only time this is mandatory is with single and double sided disks. If a disk's track count has changed or the density is altered, DOSPLUS will recognize that automatically on the next access. However, unless the system specifically looks for it, a single sided disk had no immediate difference from a double sided disk and vice versa. This information <u>is</u> stored in the DCT, but we have to be told to look at it.

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 <u>not</u> mount properly.

Examples:

      I

This command will cause DOSPLUS to flag the DCT information in all drives. This is necessary when switching from a double-sided disk to a single-sided disk or vice versa in dual-headed disk drives.

      I :0 (MOUNT)

This form of the I command will cause the system to flag drive 0 for its drive control information. If the disk in drive 0 was a double sided disk and you wish to read a single sided disk in the same drive, mount the single-sided disk and then issue this command.

If the new disk is formatted identically to the one already in drive 0, then this command is unnecessary; however, issuing it will do no harm.

JOIN

This command will link together two devices within the DOSPLUS system.

===

The command syntax is:

    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 <-   4DCAH
    $01 @DO <-> 0473H
    $02 @PR   -> 03C2H
    $03 @RS   -   NIL
    $04 @U1   -   NIL
    $05 @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 0-5 can be the primary device. These are the system devices @KI, @DO, @PR, and @RS, 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. <u>from</u> the device being linked <u>to</u> 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.

Finally:

  Here are some examples of illegal JOINs:

    JOIN FOO/BAR:0 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.

===

The command syntax is:

    KILL filespec (param=exp)
    KILL [FROM] drivespec [USING] wildmask (param=exp)
    KILL devicespec
    KILL drivespec

> <u>filespec</u> is the name of the file you wish to delete.

> <u>wildmask</u> and <u>drivespec</u> 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.

> <u>(param=exp...)</u> is an optional parameter affecting the action of the KILL command. You only have valid parameters when killing files.

> <u>devicespec</u> is the name of a device you wish to deactivate.

> <u>drivespec</u> 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 <u>that</u> file from the disk. The global method is invoked by entering the KILL command followed by a wildmask. This causes KILL to delete <u>all</u> 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" <u>is</u> 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 3.5. Our increased use of that password means that for a disk to be at all protected, a password <u>must</u> be set.

The INV and SYS 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.

The ECHO 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.

The QUERY 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 <u>not</u> 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".

The PW 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 <u>never</u> 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.

==

The command syntax is:

    LIB [TO] device/file

            <u>device/file</u> 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. Library commands -

LIST

This command will list data from a device or disk file to a specified output device or file.

==

The command syntax is:

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

==

The LIST command is normally used for listing a disk file to an output device such as the video display or the lineprinter. However, it may also be used to list data coming from other input devices such as the keyboard or the communications lines. Non-printable characters are listed as periods. However, control codes may be passed to the output channel without being translated to periods by specifying CTL=Y. Note that control codes may affect the action of the output channel. For example, a CTRL-W (17H) may cause the @DO device to switch to reverse video; similarly, a CTRL-L (0CH) may cause your lineprinter to execute an unexpected form feed to the next page.

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 FOO/BAR

    This command will list the disk file called FOO/BAR to the video display (default output device). Control codes will be displayed as periods ".").

            LIST FOO/BAR (CTL=Y)
            LIST FOO/BAR (CTL)
            LIST FOO/BAR,CTL
            LIST FOO/BAR,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 FOO/BAR the display may react unpredictably.

            LIST FOO/BAR:1 TO @PR

    The file called FOO/BAR 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.

===

The command syntax is:

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, that is, it is in executable format and has the /CMD extension, then the address at which it is to be loaded will be taken from the file itself (this address is saved when the program is written to disk). If the file is not in executable format, that is, it is a "core-image" file, and 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, ASCII text, or binary data. It is generally given the extension /CIM. If a file is given without an extension, the LOAD command will assume the extension /CIM if START is specified, or the extension /CMD if not.

The PROMPT 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 in the drive. Pressing ENTER will cause the system to proceed with the load. If the program is to be executed, you will then be prompted again to re-mount the system disk in the drive before the program is executed.

The RUN 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.

The START 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.

The TRA 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 EOOBAR/CMD:0 (PROMPT,RUN)
       LOAD EOOBAR/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.

PROT

This command allows you to change diskette information.

==

The command syntax is:

   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 whose value is 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. This field can be entered in free format.
LOCK=switch	Determines whether the disk master password is to be assigned to, or removed from, all the files in the directory or not.
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).

The PW 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.

The MPW 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.

The NAME parameter assigns a new name to the diskette. The name must be a string of up to eight valid characters enclosed in quotes.

The DATE 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.

The LOCK 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.

The ACC and UPD 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.

The CLEAN 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.

RENAME

This command will permit you to rename devices, disk drives and disk files.

==

The command syntax is:

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 or disk drive to its default driver.

==

The command syntax is:

    RESET
    RESET [FROM] devicespec/drivespec

           devicespec/drivespec is the current logical name of the device or
           disk drive 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 or drive specification will perform a GLOBAL reset of all devices and disk drives. If a devicespec or drivespec is included on the command line, then only that device or drive 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 or drive 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.

RS232

This command allows you to display the current settings of or alter the settings for the serial communications device (RS232).

===

The command syntax is:

    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.

    The Model I includes no standard drivers for this device. The drivers included with the Model III ROMs are so inefficient that DOSPLUS disables them on powerup. Therefore, you can use this command to alter any settings that you desire, but before you implement any use of the RS232 device, you will have to install the drivers we have provided on the disk for your convenience. For specific information on this driver, look in the section **Drivers and Filters**.

To display the current settings, type :

RS232

and press ENTER. The current settings for the serial interface will be displayed.

The BAUD 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.

The WORD 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.

The STOPS 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.

The PARITY 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.

The EVEN 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.

The ODD 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.

The DTR 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.

The RTS 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.

The BREAK 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.

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.

Finally

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.

The default settings are controlled by the ROM. If you wish to alter these, change the settings as desired and save them as part of a configuration file (see SYSTEM).

SCREEN

This command is used to output video data to another device or file.

==

The command syntax is:

　　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 to your requirements.

===

The command syntax is:

    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.

Abbreviations:

    TIME   T
    DATE   D
    LOGO   L
    BLINK   B
    CAPS   C
    CURSOR   CU
    HIGH   H
    STEP   S
    SAVE   SA
    PORT   P
    MODE   M

===

    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$ 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$ 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$ 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.

All of these values 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.

The TIME 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.

The DATE 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.

The LOGO 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.

The BLINK 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.

The CAPS 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.

The CURSOR 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.

The HIGH 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.

The STEP 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 <u>any</u> 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 <u>overall default</u> 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 <u>relative</u> values, not the actual step rate.

The SAVE 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.

The PORT 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.

The MODE 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.

Mode 3

This form of the SYSTEM command allows you to create configuration files. These files are used to permanently store your custom configurations.

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 Model III. 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.

<u>Applications of configuration files</u>

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.

TIME

This command will allow you to set or display the time in the system's real-time clock.

==

The command syntax is:

    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.

===

The command syntax is:

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.

UTILITIES

DOSPLUS 3.5 Utilities Manual Table of Contents

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" is the optional parameter that modifies what action the parameter takes.

The allowable 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 proceed with the BACKUP, or "N" to abort. Use "F" to reformat the diskette.

Abbreviations :

DATE   D
USE   U
==

   The backup utility enables you to backup 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 3.5's BACKUP utility, it is not necessary to pre-format your destination diskettes. If the diskette is blank, DOSPLUS 3.5 will format it automatically. Even if the diskette was previously formatted, DOSPLUS 3.5 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 and whether or not you wish to use the disk if it contains data.

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.

To implement the "Use" parameter, simply type "USE=Y" or "U=Y" in the parameter list. This will inform BACKUP that you do not wish to be prompted before over-writing a disk that already contains data.

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 -

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 proceed 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.

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 have 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 accomplish 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.

  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.

One 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 specified 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.

CONVERT

The CONVERT utility has several uses:

    (1) To allow DOSPLUS 3.5 to copy program and data files from double-density TRSDOS systems onto DOSPLUS 3.5-compatible diskettes.

    (2) To allow DOSPLUS 3.5 to display a file catalog of double-density TRSDOS diskettes.

    (3) To render Model I single-density diskettes readable under Model III DOSPLUS 3.5.

==
CONVERT [FROM] :sd [TO] :dd [USING] wildmask (param=exp)
CONVERT [FROM] :dr (param=exp)

The parameters for CONVERT are:

CAT=switch
: Instructs CONVERT to display a file catalog of a double-density 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

MOD1=switch
: Used to inform CONVERT whether a Model I or Model III double-density diskette is to be used (Model I DOSPLUS only)

Abbreviations:

```
ECHO        E
INVIS       I
SYSTEM      S
OVER        O
QUERY       Q
MOD1        M
```
==

The first application mentioned above, that of copying files from double-density TRSDOS diskettes onto DOSPLUS 3.5 diskettes, can be used on the Model III and double-density Model I DOSPLUS 3.5 systems. Single-density Model I's cannot perform this function, since the diskettes are unreadable to a single-density machine.

Using CONVERT to copy files from double-density 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 double-density TRSDOS diskette, ":dd" is the destination drive containing DOSPLUS-compatible media, and "wildmask" is a valid DOSPLUS wildcard specification.

Examples:

Command	**Action**
CONVERT FROM :1 TO :0 USING */CMD	Copies all /CMD files on the double-density TRSDOS diskette in drive 1 onto drive 0.
CONVERT :1 :0	Copy all files from the double-density TRSDOS diskette in drive 1 to drive 0.
CONVERT */TXT:3 :1	Copies all /TXT files from drive 3 onto drive 1.

Several parameters are valid when using this form of CONVERT. The ECHO parameter is used to instruct the CONVERT utility to <u>echo</u>, or display, the name of 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.

The MOD1 parameter is present only on the Model I DOSPLUS 3.5 CONVERT utility. Model III DOSPLUS users need not concern themselves with this parameter. Model I CONVERT is capable of CONVERTing files on both Model I and III double-density TRSDOS. The Model I CONVERT program will normally assume that the CONVERT is to take place on a Model I double-density TRSDOS instead of a Model III TRSDOS. Therefore, to CONVERT the files on a Model I double-density TRSDOS, one might use the command:

CONVERT :2 :1,E

However, if it were desired to CONVERT the files from a Model III TRSDOS diskette, the command would be modified to read:

CONVERT :2 :1,E,MOD1=N

By specifying "MOD1=N", we inform CONVERT that the diskette to be CONVERTed is <u>not</u> a Model I double-density TRSDOS, and therefore must be a Model III TRSDOS.

The INVIS switch is used to tell the CONVERT utility to CONVERT files that are invisible on double-density TRSDOS as well as files which are visible.

The SYSTEM switch is used to inform CONVERT to copy files which have the system file attribute as well as non-system files.

The OVER 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 0, 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.

The QUERY parameter 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.

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 with the 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

The CONVERT utility may also be used to display a file catalog of a double-density TRSDOS diskette. The form of this command is:

CONVERT :dr (CAT)
        or
CONVERT :dr (DIR)

where ":dr" is the drive specification of the disk drive containing the double-density TRSDOS diskette whose directory is to be displayed. Note that the V12 and V13 parameters are not necessary for this function of CONVERT.

Model I users must use the "MOD1=N" parameter in order to display a file catalog on a Model III TRSDOS diskette.

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 III.

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 <u>will</u> 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 <u>double-density Model I system</u>, 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, and 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 [FROM] :dr [TO] filespec/devicespec (param=exp)

":dr"   is the drive specification of the disk drive containing the diskette whose directory is to be examined

"filespec/devicespec"   is the optional output file or device to which all messages concerning the state of the diskette's directory are routed.

The allowable parameters for DIRCHECK are:

CYL0   Instructs DIRCHECK to ignore a possible "extraneous or unassigned gran" error on Model I double-density diskette bootstrap files

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:

CYL0   C
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.

The list of errors may also be directed to any other DOSPLUS 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 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.

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 to diskette
BOOT/SYS not assigned space	Restore BOOT/SYS file to diskette
DIR/SYS not found	Restore DIR/SYS file to diskette
DIR/SYS not assigned space	Restore DIR/SYS file to diskette

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.

Another parameter is present in DIRCHECK, called CYL0. This parameter is for use on Model I double-density DOSPLUS diskettes which have a single-density cylinder 0. On such diskettes, the file BOOT/SYS does not occupy as many grans as are allocated in the granule allocation table. This will result in an "extraneous or unassigned gran" error, even though the diskette is perfectly normal. The CYL0 parameter instructs DIRCHECK to ignore this special case with cylinder 0, although it does not suppress the same error message should the same error occur on some other cylinder.

NOTE:　　　On floppy diskettes, DIRCHECK will set the surface count for the diskette according to the current CONFIG settings for the appropriate drive if the GAT parameter is used. That is, if drive 1 is CONFIGed for SIDES=2 at the time a DIRCHECK :1,GAT is performed, DIRCHECK will write information to the GAT which indicates that the diskette is a double-sided diskette.

　　　　　　　This feature may be used to good advantage when converting DOSPLUS 3.4/4.0-style double-sided floppy diskettes to DOSPLUS 3.5 format.

DISKDUMP

This is a machine-language disk sector display/modify utility.

===
DISKDUMP filespec <ENTER>

"filespec" is the 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 a 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
@ xx	Fill "xx" bytes with 00 byte, starting at current cursor location
@ xxyy	Fill "yy" bytes with "xx" byte, starting at current cursor location

A sector display will look like this :

```
02      00: FF7B 6F0A 003A 93FB 2020 2020 2050 6174 .{o..:..    Pat
06      10: 6368 2070 726F 6772 616D 2066 6F72 2044 ch program for D
        20: 6F73 706C 7573 204D 6F64 656C 2049 4949 osplus Model III
        30: 009C 6F14 003A 93FB 2020 2020 2050 726F ..o..:..    Pro
        40: 6772 616D 6D65 7220 3A20 4D52 4C2F 4D53 grammer : MRL/MS
        50: 5300 3F70 1E00 3A93 FB20 2020 2020 506C S.?p..:..    Pl
        60: 6561 7365 206D 616B 6520 6365 7274 6169 ease make certai
        70: 6E20 7468 6973 2069 7320 7573 6564 206F n this is used o
        80: 6E20 7468 6520 7072 6F70 6572 206D 6163 n the proper mac
        90: 6869 6E65 0A09 0909 2020 2020 2020 2020 hine....
        A0: 284D 6F64 656C 2049 202D 204D 6F64 656C (Model I - Model
        B0: 2049 4949 292E 2020 416C 736F 206D 6168  III).  Also mak
        C0: 6520 6365 7274 6169 6E20 7468 6174 2074 e certain that t
        D0: 6865 0A20 2020 2020 2020 2076 6572 7369 he.        versi
        E0: 6F6E 206E 756D 6265 7273 206D 6174 6368 on numbers match
        F0: 202E 2E2E 00A3 7028 003A 93FB 2020 2020  .....p(.:..
```

    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 <CLEAR> 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 block graphics 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 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 the DOSPLUS 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:

```
35 (Model I) or 40 (Model III) cylinders
10 (Model I) or 18 (Model III) sectors on track 0
Single (Model I) or double (Model III) density track 0
10 (Model I) or 18 (Model III) sectors on all remaining tracks
All remaining tracks single (Model I) or double density (Model III)
```

Any of these may be altered via the "Set" sub-option. The asterisk that appears to the left of the "Display" 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)

DISKZAP powers up with the control cursor positioned for this sub-option. If the default parameters shown above are proper for the diskette you wish to work with, 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 he 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" single-density diskettes, 10 (on the Model I) or for 5-1/4" double-density diskettes, 18 (on the Model III). 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 "5" for single density or a "0" for double density.

Note that Model I DOSPLUS 3.5 system disks use a single density track zero. This is required by the ROM bootstrap loader. DOSPLUS 3.5 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 he 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 III DOSPLUS 3.5, 40 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. ft 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 <u>data address mark</u>, which is reserved for use by the DOSPLUS 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 <u>not</u> interchangeable with the DOSPLUS 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.

<u>Display (Display or modify diskette sectors)</u>

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 :

```
06   00:  54BE 2320 0100 FC53 0510 F7E1 BFC9 E118  T.# ...S........
04   10:  03CD 1954 10FB 18CC E5B7 ED52 E1C9 CD1B  ...T.......R....
02   20:  00C2 D355 C9CD 1300 C2D3 55C9 DD21 C156  ...U......U..!.V
     30:  DDCB 0046 203F CD19 54FE 02DD CB00 9620  ...F ?..T.......
     40:  04DD CB00 D6FE 20D2 CB55 FE03 DDCB 008E  ...... ..U......
     50:  3004 DDCB 00CE CD19 543D 3DDD 7701 CD19  0.......T== w...
     60:  54DD 7702 CD19 54DD 7703 DDCB 00C6 B7DD  T.w...T.w.......
     70:  CB00 5620 15CD 1954 DD35 0120 04DD CB00  ..V ...T.5. ....
     80:  86DD CB00 4E21 0000 37C8 DD6E 02DD 6603  ....N!..7..n..f.
     90:  DD34 02C0 DD34 03C9 3AA9 57FE 8028 103E  .4...4..: W..(.>
     A0:  2ACD 3300 063F 21C5 56CD 4000 D818 1F11  *.3..?!.V.@.....
     B0:  A957 21C5 5606 40CD 1300 2807 FE1C C2D3  .W!.V.@...(.....
     C0:  5537 C977 23FE 0D28 0510 ECC3 C855 21C5  U7.w#..(.....U!.
     D0:  563E 2EBE 28C2 0641 E5CD F354 CD56 55ED  V>..(..A...T.VU.
     E0:  5305 57E1 E506 46CD F354 1107 57CD 0955  S.W...F..T..W..U
     F0:  E1E5 0643 CDF3 5411 4757 CD09 55E1 C97E  ...C..T.GW..U..~
```

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

<u>Key</u>	<u>Function</u>
;	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 graphic block 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.

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.
CLEAR	Toggles sector display between hexadecimal and ASCII character display mode
@ xx	Fills the entire sector, 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.

The allowable parameters are:

DATE="string"   Allows you to set the format date from the command line. This should be expressed as a quoted literal up to eight characters in length. You are not restricted to numeric input. If this is not given in the command line and the system date is not set, FORMAT will prompt you for it.

PW="string"   Disk Master Password. This parameter allows you to specify the Disk's master password from the command line. This should be expressed as a quoted literal up to eight characters in length.

NAME="string"   Allows you to specify the disk name from the command line. This should again be expressed as a quoted literal and may be up to eight characters in length.

CYLS=value   Number of cylinders. This allows you to specify the number of cylinders to format the disk to. This should be expressed as a numeric value, not a quoted literal.

SIDES=value   Number of sides. This allows you to specify single or double headed format from the command line. This should also be expressed as a value (either 1 or 2), no quotes are needed.

DEN="string"   Track density. This parameter allows you to specify the format density from the command line. It should be expressed as a single character quoted literal. Either an "S" for single density or a "D" for double density.

USE="string"   This allows you to override the prompt "Diskette contains data, Use or not?" that appears when the disk to be formatted is not blank. This is your only warning, so using this parameter can be dangerous if used without caution. This parameter should be expressed as a string (either Y or N).

The default values will be obtained by pressing ENTER when prompted for one of the above. You will be prompted for any fields not filled from the command line. The defaults are :

DATE	01/01/80
PW	No password
NAME	No name
CYLS	40
SIDES	1
DEN	Single Density (Model I) Double Density (Model III)
USE	No

Abbreviations :

DATE	D
PW	P
NAME	N
CYLS	C
SIDES	S
DEN	DE
USE	U

==

   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 3.5 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.

   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 3.5 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 "S" on the Model I and "D" on the Model III. DOSPLUS 3.5 formats 10 sectors per track in single density and 18 in double.

After you have answered all these questions, DOSPLUS 3.5 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 signaled 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 3.5.

Any of those questions that you answered from the command line via the parameters listed would not have been asked. FORMAT only prompts for what it doesn't know. If the system date is set, the "Format date" question will not be asked.

HELP

The HELP utility on DOSPLUS 3.5 is intended to provide a means of quick reference to the proper syntax and valid parameters for DOSPLUS 3.5 library commands and some of the often-used utility programs.

```
=========================================================================
    HELP
    HELP [FROM] command [TO] filespec/@devicespec

               where "command" is the name of a DOSPLUS 3.5 library command or
               utility described in the PATCH 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	LIST	LOAD
PAUSE	PROT	RENAME	RESET	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] filespec/@devicespec [USING] wildmask (param=exp)

":dr"   is a disk drive specification

"filespec/@devicespec"   is a file or character-oriented device to which the output from MAP will be sent

"wildmask"   is a valid DOSPLUS wildcard specification

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:

```
DOS:3.50   01/26/83
RS/DVR       001,006 - 001,011
TRAP/CMD   001,012 - 001,017
FILE/DVR     003,000 - 003,005
DO/DVR       000,006 - 000,011
KI/DVR       000,012 - 000,017
PR/DVR       001,000 - 001,005
```

Each filename is followed by a set of numbers. Take, for example, the case of RS/DVR, in the listing above. The MAP tells us that RS/DVR 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 first example, above, would appear like this in hexadecimal format:

```
DOS:3.50   01/26/83
RS/DVR        01,06 - 01,0B
TRAP/CMD   01,0C - 01,11
FILE/DVR      03,00 - 03,05
DO/DVR        00,06 - 00,0B
KI/DVR        00,0C - 00,11
PR/DVR        01,00 - 01,05
```

The SYSTEM and INVIS parameters may be used to cause the MAP utility to display a MAP of system files and invisible files, respectively.

The MAP utility will accept a wildmask specification. This means that if you are only interested in displaying a MAP a one file, or of a class of files, MAP can accommodate you. For instance, to display a MAP of all files on drive 3 ending with the /CMD extension, issue the command:

MAP */CMD:3

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 (SYS,INV)

PATCH

The PATCH/CMD file provided on DOSPLUS 3.5 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.

The allowable 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 proceed. 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 following 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 3.5, 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.

```
=======================================================================
RESTORE filespec

"filespec"                    is the name of a KILLed file.
=======================================================================
```

Note that RESTORE cannot reclaim KILLed 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.

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 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 <u>first</u> occurrence of the proper filename in a diskette directory. If the same filename has been created and KILLed several times upon a diskette, RESTORE may not recover the same occurrence of the file as was intended, If this is the case, the improper file should be RENAMEd and KILLed. 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

Allowable 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.

Abbreviations:
XFER   X
===

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 3.5 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

The optional parameter, XFER, is used in conjunction with the special bootstrap program. If the bootstrap contains an <u>alternate system driver</u> program, the XFER parameter is used to provide DOSPLUS 3.5 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.

<u>TAPE</u>

This utility will transfer a system tape to disk, transfer a disk object file to system tape, or relocate these files, writing them back to whichever media the operator specifies.

===
TAPE

    There are no parameters for this utility
===

    Once the tape utility is loaded in, the program header will be displayed and TAPE will prompt the operator with an asterisk, "*". Any valid TAPE utility command may now be entered. The commands are as follows :

Command	Function
B	Adjust Baud rate (Model III only)
H	Print help list
I	Re-Initialize program
L	Load a disk file
M	Map out program area
O	Offset program
Q	Exit to DOSPLUS
R	Read a system tape
S	Save a disk file
W	Write a system tape

    "B" allows you to adjust the baud rate for cassette generation. The syntax is simply "B L" to set it to low speed (500 baud) or "B H" to set it to high speed (1500 baud). Remember, only Model III can read a 1500 baud tape. If you are going to take this to a Model I, write it out at 500 baud.

    The "H", or <u>help</u> command prints displays a list of the valid TAPE commands.

    "I" will re-initialize the tape utility. The syntax is simply "I". There are no parameters. If you do not use this option, when you load your next file it will bring it in memory along with your current file. This means that it is possible for you to append two programs together. The are several conditions, though. The programs must be relocatable and written in a fashion that would allow such a joining. The way you would handle it is to load in the first file, offset it (to move it out of the way), load in the second file, offset the entire thing (if needed), and write the entire thing out under one filename.

    "L" will load a disk file into memory. The syntax is "L filespec" (where filespec is a standard DOSPLUS file specification). It does nothing other than load the file into memory. For any information regarding the file's load and execute addresses, you must use the map option.

"M" will map out the load and execute addresses of all files currently in memory. The syntax is simply "M" or "M-P". The optional "-P" indicates that it should output the map to both the screen and the line printer. There are no other parameters. It will display the program name (as you typed it in), and then will display all program and data areas. It uses the format "address-address" and displays each segment separately. Then it will give you the program transfer address (the address the execution begins at) and the current offset. Each segment will be displayed twice. The left-hand column is the original segment and will remain unchanged. The right hand column is the offset segment and will change to reflect the offset currently in effect.

"O" will offset all currently loaded program and data segments to the specified amount. The syntax is "O nnnn" where "nnnn" is the amount you wish to offset the modules. Remember, this is NOT the address you wish it to be offset to, it IS the offset amount. For example, if a program loaded in at hex 8C00 and the O command is used to offset it 1000 hex bytes ("O 1000"), it will now load in at hex 9C00. Please note that tape works only with hexadecimal values. If you type in a value greater than four digits, only the last four will be used. For 99% of standard usage, an offset of hex 1000 (i.e. O 1000) will work very well.

"R" will read a system tape. The syntax is "R filename" (where filename is the tape filename). That is all this command does, to manipulate any of the data read in you must refer to the map and offset commands.

"S" will save the file currently in memory to disk. The syntax is "S filespec" or "S filespec-A". The "filespec" is a standard Dosplus file specification. The "-A" is the optional syntax to indicate that you wish the tape utility to append a module to disable DOS upon program load. If for example, your program HAS to load in at hex 4E00 (below DOS user memory), you would save it with the "-A" parameter. Upon loading, the module would reset the I/O vectors to the ROM and then re-locate the program to the specified load address and transfer control. This is primarily for moving tape-based games or other programs that may not allow you the liberty of relocating them when you move them to disk. For most applications, the appendage (i.e. "-A") will be required.

"W" is the command to write a system tape. The syntax is "W filename" or "W filename-A". The "filename" is a standard system tape filename. The "-A" appendage is the same as described under write (see above). For most applications, the appendage will be needed.

Note that the TAPE utility can be used to relocate programs and then dump them back to the <u>same</u> media from which it was taken (disk-to-disk, tape-to-tape). In this way, the TAPE utility has many more applications than simple tape-to-disk (or vice-versa) transfer.

TAPE can also be used to determine where a program loads in memory. This can be a time-saving function when it is necessary to know where a machine-language program loads into RAM.

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 successfully carried out upon retrying, the TRAP will not intervene and the program will proceed as if no error occurred. 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 occurred. 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.

CO-DIR 2.1 - Cursor Oriented Directory Version 2.1
Copyright (c) 1982 by Hunt K. Brand

Picotrin Technology Inc.
3531 San Castle Blvd.
Lantana, Fl. 33462

TABLE OF CONTENTS

1.1   INTRODUCTION:

CO-DIR 2.1 - Cursor Oriented Directory Version 2.1

CO-DIR 2.1 is a utility program which allows you to perform, with one or two key strokes (no need to type in file names), almost all the DOS commands relating to individual files on a disk. CO-DIR 2.1 allows you to process the files as they are listed on a disk Directory screen, by using a pair of blinking cursors. To use CO-DIR 2.1, simply type CODIR instead of DIR or CAT.

DOS commands and built-in commands which are supported include: Copy, Directory, Free, Help, Do file, Kill, List, Load, Modify, Print, Rename, Search, and Zero. Each of these commands are discussed in detail within the documentation.

CO-DIR 2.1 is not a program which will be used once in a while. It will be used almost every time you boot up your DOS since most of the DOS functions you would use can be done with CO-DIR 2.1 much faster and easier.

Every page you see while the Directory is being listed is saved in memory so you can scroll the Directory pages just like a word processor. Pressing <SHIFT> <UP ARROW> will cause the cursors to move to the top left hand side of the screen and if more than one Directory page is in memory, will move the first page onto the screen.

All options (except <ENTER>) display a prompt (Yes/No questions or Drive #) on the Command line (last line of screen),
i.e. Copy to which drive?

1.2   MANUAL NOTATION:
All capital letters or words enclosed in left and right brackets represent individual keys on the key board. Any two together represent keys that should be pressed together.
i.e. <SHIFT> <UP ARROW>
        Press the shift key and the up arrow key together.
    <ENTER>
        Press the enter key.

1.3   SOUND:
To take advantage of CO-DIR'S optional sound, you must have a Radio Shack "Mini-Amplifier", (Radio Shack Catalog # 277-108) or its equivalent, hooked up to the AUX plug (the large gray one) of the cassette cable. Using the SOUND option causes low tones if function was completed. High tones mean an error in instruction or, CO-DIR was unable to perform function. Warning tones for Kill, Print, and Zero commands are also generated.

NOTES:
Be sure and read the section on BUFFER prior to using CO-DIR 2.1.

The DOS CLOCK should not be active (displayed on the screen) while using CO-DIR.

1.4 PASSWORD:

If a file is password protected you can still access it, just hold down the <SHIFT> key when choosing your option, i.e. if you press <SHIET> <L> ,CO-DIR will ask for a password at the Command line then append the password to the file for you before listing it.

2.1   SETTING UP THE SYSTEM:

2.2   PROGRAMS:
CO-DIR 2.1 contains two files: CODIR/CMD, CODIR2/MOD.

2.3   ONE DRIVE USERS:
If you have only one drive you will need to copy over CODIR/CMD and CODIR2/MOD (CODIR/CMD is 3 grans long, CODIR2/MOD is 2 grans long).

2.4   TWO OR MORE DRIVE USERS:
If you have more than one drive you can leave the programs on any disk while configuring to your system.

2.5   CONFIGURE THE SYSTEM:
Run the BASIC program CODIR2/MOD and answer the questions. This will allow you to activate or de-activate most of the command functions (among other things). You will then have a copy of CODIR/CMD designed for your particular needs and system.

2.6   TRANSFER CO-DIR 2.1 TO YOUR DOS DISK:
Although you can use CO-DIR no matter what disk it is saved on, it is better to copy it over to your DOS disk. This will allow it to load faster when you activate it. If it were saved on a separate drive, the DOS would have to look for it on each drive until it found it; if CO-DIR is on the DOS disk, the DOS will find it as soon as it starts looking.

The only program you need to copy over to your DOS disk is CODIR/CMD. CODIR2/MOD will rarely be used, therefore you can save it on any disk and only insert that disk when you need to modify CO-DIR 2.1.

2.7   RENAME CO-DIR 2.1:
CO-DIR 2.1 is supplied on your disk with the file name of CODIR/CMD. You can rename it to any legal file name of any length (be sure to keep the /CMD extension), i.e. DIRECT/CMD, D1!CMD, HELLO/CMD, etc.

If you do rename the CO-DIR file, try to make the first character of the name a "D" since the <D> Directory command in CO-DIR is activated by pressing a "D". This will prevent you from having to press two different keys for the same function, i.e. If you renamed CODIR/CMD to C/CMD then you might press <C> while CO-DIR is active expecting a Directory command but you would get the Copy command.

> NOTE:
> Do not change the Copyright notice displayed when CO-DIR 2.1 loads itself. If you do modify the title line, the computer will not function properly and you will be warned to re-boot the system - CO-DIR will not allow itself to be used if this line is changed.

At this point you should have modified CO-DIR 2.1 using CODIR2/MOD and transferred it to your DOS disk. You are now ready to begin using CO-DIR 2.1.

3.1   OPERATION:

####################### SPECIAL NOTE ######################

CO-DIR 2.1 is designed to be used from DOS only – not
from BASIC. DO NOT USE CO-DIR 2.1 FROM BASIC.

####################### SPECIAL NOTE ######################

Normally during a CAT command, DOS PROMPT (the message your DOS displays when it is ready for another command) will appear after all the files have been displayed (whether or not you had to press <ENTER> or the <SPACE BAR> to accomplish this for multi-page listings). If you press the <BREAK> key the listing will stop before all the files have been displayed, and DOS Prompt will appear. In any of these examples DOS Prompt appears after the information is listed, or you stop the listing by pressing the <BREAK> key.

3.2   USING CO-DIR 2.1:
To use CO-DIR 2.1 simply type CODIR instead of DIR or CAT. All legal CAT commands are accepted. Here are some examples:

CODIR (i)
CO-DIR (CAT type) Directory of drive 0 with
invisible files.

CODIR :2
CO-DIR (CAT type) Directory of drive 2.

If you type CODIR instead of DIR or CAT, a normal Directory will follow, just as described above except a small graphic block will appear in the upper right hand corner of the screen and stay there until the Directory is stopped manually or normally finishes. Short beeps will be heard through the cassette port as each line is saved. The small graphic block shows that the CO-DIR 2.1 BUFFER is active and saving all information listed in a memory buffer so you can scroll up to files that have scrolled off the screen during the listing of the Directory.

Once the files are listed, instead of the normal DOS prompt appearing like it does after a normal Directory listing, CO-DIR 2.1 will display this message at the bottom of the screen:

CO-DIR 2.1 - Cursor Oriented DIR. (C) 1982 by Hunt K. Brand.

Two cursors will begin to blink over the first file listed on the Directory screen. CO-DIR is now ready to accept commands.

To exit CO-DIR and return to DOS, press the <BREAK> or <CLEAR> key while the cursors are blinking over a file name. If the Directory listed had no files in it, CO-DIR would automatically exit to DOS.

3.3 CURSOR CONTROL:

Once CO-DIR 2.1 is active, two cursors will blink over the first file listed at the top of the screen. You can move the cursor over any file you want to process by pressing the arrow keys in any combination. If you press the <DOWN ARROW> and the cursors move to the bottom of the screen, the files listed on the screen will begin to scroll up (if there are more files in the buffer). The same holds true for moving the cursor up. Press <SHIFT> <UP ARROW> to move the cursors over the first file listed on the first page of the Directory - the top of the Directory. The SEARCH command can help you find (position the cursors over) a particular file (see SEARCH in Chapter 4).

Killed (files marked as killed using the <K>ILL option) may not be processed using CO-DIR 2.1 so the cursors will usually skip over them as it moves.

To force the cursors to stop over a system or killed file that they would normally skip over, hold down the space bar in conjunction with the arrow keys. This is usually used so you can move the cursors over a file which is next to a killed or system file.

3.4 CO-DIR 2.1 DIRECTORY BUFFER:

Whenever you activate CO-DIR, a memory buffer is opened and will remain open until the Directory listing is stopped manually or completed normally. To indicate that the buffer is open, a small graphic block will appear in the upper right hand corner of the screen, and short high tones will be heard as each line of Directory information is listed. Each file name listed on the screen is saved in the buffer so it can be recalled later as you move the cursors, or search for a file name.

At the end of the listing a short low tone will be heard to indicate that the buffer is closed and the graphic block will turn off.

A common procedure is to hold down the <ENTER> key until all pages of a Directory are listed (since you can always scroll up to see files which were listed). The buffer will close and CO-DIR will wait for you to release the <ENTER> key before activating the cursors over the first file. Once the cursors are blinking, the <ENTER> key will function as described in the COMMANDS section of chapter 4).

The buffer can be closed manually by pressing the <BREAK> key. Since the <BREAK> key is also used to stop a Directory listing, the end of the Directory listing (which turns off the buffer) and the pressing of the <BREAK> key (which also turns off the buffer) usually occur simultaneously.

Since the buffer is closed the instant the <BREAK> key is pressed, you should not press the <BREAK> key until the current page of Directory information is listed. If you pressed the <BREAK> key while the files were being listed near the top of the screen or in the middle, the rest of the files would still list normally until the DOS stopped the Directory listing. As soon as CO-DIR activated itself, all the files which were listed after you pressed the <BREAK> key and before the DOS stopped the listing, would be erased. This is because they were not saved in the buffer. This causes no harm to the computer or the remaining files being processed, just bear this in mind when you press the <BREAK> key.

3.5   BUFFER FULL:
   The memory buffer will save the files listed in memory up to the top of memory as marked by the DOS Himem pointers.

   If the buffer is filled before all the files are listed, the buffer will turn itself off and all subsequent files will not be saved in the buffer until the Directory listing stops. Since you were probably looking for a particular file and wanted to continue the search even after CO-DIR turned off the buffer (indicated by the graphic block going off and a short high tone) CO-DIR will perform the following:

   After the Directory listing is stopped, CO-DIR will transfer the last set of files (the ones on the screen after the listing was stopped) into the top of the memory buffer. This will cause some of the files saved previously to be lost but will allow you to process the files on the last page as well as most of the files which were saved in the buffer before the buffer turned off.

3.6   TURN OFF THE CLOCK:
   Since the buffer is constantly updated (saving the screen to the buffer) as you modify files on the screen, the DOS CLOCK should not be on (displaying the time in the upper right hand corner). If it is, CO-DIR will save it in the buffer every time you scroll the screen - erasing over parts of file names!!! - To avoid this problem, make sure the CLOCK is off before calling up CODIR/CMD.

```
     4.1  COMMAND OUTLINE:

COMMAND KEY                    FUNCTION
   ================================================================

<ENTER> ..  EXECUTE file - if no CMD extension is found, assume
            file is BASIC so load BASIC then run file.


<SPACE>&
<ENTER> ..  Force CO-DIR to EXECUTE file as CMD even if no CMD
            extension is found.


<BREAK> or
<CLEAR> ..  EXIT CO-DIR (cursors must be on a file).

<A> ......  Get CO-DIR DIRECTORY of specified drive using A,I
            options.

<C> ......  COPY file to specified drive.

<D> ......  Get CO-DIR DIRECTORY of specified drive.
            Pressing <SHIFT> <D> causes invisible files to be
            listed, also.

<F> ......  FREE, asks for drive number then shows a map of that
            drive.

<H> ......  HELP, Display all command keys on command line,
            i.e. shows ...<ENTER> A C D F C H J K L M O P R S Z.

<J> ......  JOB CONTROL, Activates a DO just like typing "DO
            filename".

<K> ......  KILL file.

<L> ......  LIST file on video (restore screen afterward).

<M> ......  MODIFY file. Uses DISKDUMP. CO-DIR automatically
            adds the name of the file to be processed.

<O> ......  LOAD file. Load /CMD or /CIM type file to memory.

<P> ......  PRINT file to printer.

<R> ......  RENAME file. Type in new name on Command line.

<S> ......  SEARCH for file or extension.

<V> ......  VIEW copyright and license notice (screen restored).

<Z> ......  ZERO file. Performs a CLEAR on specified file.
```

4.2   COMMANDS:

There are 14 command keys displayed by the <H>ELP option (which is one of the commands). The exit and execute commands will be described first then each of the other commands will be described in alphabetical order based on the key which activates it.

4.3   COMMAND DESCRIPTIONS:

The "Required response" line under each command heading in this chapter indicates what CO-DIR expects as a response after you press that command key. The "Exit command" line indicates what keys you may press to cancel the requested command.

All commands except <ENTER> - execute, <J>OB CONTROL, and <L>OAD, allow CO-DIR to remain active in memory so you can process several files on several different drives without having to reload CO-DIR after each function.

4.4   <ENTER> - EXECUTE FILE

Required response : none
Exit command : none

To execute a program (it must be a program and not a data file), move the cursors over the program you would like to execute then press the <ENTER> key. If the program has not been marked as killed (see <K> - KILL) and is not a system file (a file with a ISYS extension), the screen will clear and the program name will be printed at the top of the screen, then CO-DIR will deactivate itself and the program will automatically be loaded and executed.

If the program does not have a CMD extension, CO-DIR assumes it is a Basic program and will automatically load Basic, opening three files (or the number of files that were set using CODIR2/MOD), then it will run the Basic program.

To force CO-DIR to treat a file as a CMD even if it does not have a CMD extension, press the <SPACE BAR> and hold it down while you press the <ENTER> key. If the file needs a password, press <SHIFT> <ENTER> to type in the password (see PASSWORD ACCESS below). Type in the password then hold down the <SPACE BAR> before pressing <ENTER> at the end of the password, this will cause CO-DIR to append the password onto the end of the file then it will execute the file.

4.5   <A> - CO-DIR DIRECTORY WITH A,I OPTIONS

Required response : <ENTER> or drive number
Exit command : any other key (except <SPACE>)

Pressing <A> will cause the following message to appear on the command line:

CO-DIR Directory of which drive (d)?

Press the drive number required, or <ENTER> to use the default drive number (spaces are ignored). Press any other key to cancel the command. (See DEFAULT DRIVE NUMBER under <C> - COPY FILE below for explanation of default drive numbers). The screen will clear and CO-DIR will display a Directory of the specified drive with Invisible files and Allocation (A,I). If there are no files on the specified disk, CO-DIR will exit to DOS.

4.6   <C> - COPY FILE TO SPECIFIED DRIVE

Required response : <ENTER> or drive number other than current drive number (You can't copy to the same drive)
Exit command : any other key (except <SPACE>) or current drive number

To copy a file from one disk to another, move the cursor over the file you want to copy then press the <C> key. A small graphic block will appear to the right of the file which will be copied and the following message will appear on the command line:

Copy to which drive (d)?

Press the number of the drive you want to transfer the file to or <ENTER> to use the default drive number (spaces are ignored). Pressing any other key will cancel the command.

If you press a legal drive number then CO-DIR will begin to copy the file to the specified drive (using the same file name). The small graphic block next to the file will be replaced with the number you pressed and the command line will display the actual command being preformed. If the copy is successful a low tone will sound and the drive number that the file was just copied to will remain just to the right of the file name to show that it was copied to that particular drive.

4.7   DEFAULT DRIVE NUMBER:

When you use an option which requires a drive number as a response, the last part of the prompt line will be:

..... which drive (d)?

Options which require a drive number
are <C>OPY and <D>IRECTORY.

Other options which require a drive number are:
<F>REE option.

The small "d" between the brackets will be a number between 0 and 3 or whatever you set the maximum drive number to be with CODIR2/MOD. Pressing <ENTER> will cause CO-DIR to use the number "d" as your response to the question. If you press a number other than "d" that number will become the new default drive number until you exit CO-DIR or press a new number.

If you press an illegal drive number (one that is too big) CO-DIR will make a short high tone to indicate an error and the command will be cancelled causing the cursors to resume blinking over whatever file they were over before you pressed <C>.

4.8   EXAMPLE OF COPY:

If you are looking at the Directory of drive 0 and you want to copy a file called JULY/DAT, over to drive 1, follow this procedure: Move the cursors over the file JULY/DAT then press <C> and answer the drive question by pressing <1>. The command line will then display the following message:

Copy JULY/DAT:0 :1

The graphic block to the right of JULY/DAT will be replaced by a 1 (to show that the file is being copied to drive 1 and CO-DIR will begin to copy that file over to drive 1. If you had pressed 0 the request would have been canceled

4-2

since you can't copy a file to the same drive you are looking at, and in this example we are looking at drive 0. After the file is copied, the cursors will resume blinking over the file you just copied and the 1 will still be on the screen next to that file to show you that the file was copied over to drive 1.

4.9 <D> - CO-DIR DIRECTORY
      <SHIFT> <D> - CO-DIR DIRECTORY INCLUDING INVISIBLE FILES

     Required response : <ENTER> or drive number
     Exit command : any other key (except <SPACE>)

     Pressing <D> for normal Directory, or <SHIFT> <D> for normal Directory including invisible files, will cause the following message to appear:

     CO-DIR Directory of which drive (d)?

     Press the required drive number (spaces are ignored) or <ENTER> to use the default drive number (see DEFAULT DRIVE NUMBER under <C> - COPY FILE section above).

     If you press a legal drive number the screen will clear and a Directory of the specified drive will appear. If there are no files on the disk, CO-DIR will exit to DOS.

4.10 <F> - FREE : DOSPLUS will ask for a drive number then show
a Free Space Map of that drive

     Required response : <ENTER> or drive number
     Exit command : any other key (except <SPACE>)

Users will see the following message:

     Free space map of which drive (d)?

     Press drive number (spaces are ignored) or <ENTER> for default drive number (see DEFAULT DRIVE NUMBER under <C> - COPY FILE section above), any other key to exit command. If a legal drive number was chosen, a Free space map of that drive will be shown. A low tone will be heard to indicate that the function is completed and the following message will appear on the command line:

     Press <BREAK or CLEAR> for DOS, any other key for CO-DIR

     If you press <BREAK or CLEAR> CO-DIR will exit to DOS, otherwise the Directory page that was displayed before the Free command will be displayed and the cursors will be blinking over the same file they were over before the Free command.

4.11 <G> - GRANULE ALLOCATION, DISPLAY FILE SIZE

     Required response : <Y>
     Exit response : any other key

     This command will display a file's size in kilobytes. This will prove useful in determining whether or not a file is too large for the remaining free space on a disk BEFORE executing a COPY command.

     Position the cursor over the file you wish to interrogate and press <G>.

You will see :

Display file size (allocation) ?

Reply with a <Y> to view the size of the file.

The display will change :

File FILENAME/EXT:D is : nnK

"nnK" will be the file size in kilobytes. If this figure is "<K", it indicates that the file is LESS than one kilobyte in length.

4.12 <H> - HELP, DISPLAY COMMAND KEYS

Required response : none
Exit command : none

By pressing <H> the following line will be displayed:

Choices: <SHIFT> Password:<ENTER> A C D F G H J K L M O P R S V Z

In either case a low tone will be heard and the cursors will continue to blink over the file they were over before the command.

> NOTE:
> If you have deactivated any of the above options using CODIR2/MOD, they will not be displayed.

4.13 <J> - DO FILE ACTIVATION

Required response :
Exit command : any other key

This command allows you do activate a Do file.

Position the cursors over the DO control file (the one which will be activated with a Do command) then press <J>.

The following message will appear on the command line:

Do (activate) file?

Pressing <Y> will cause CO-DIR to clear the screen and the following message will appear at the top of the screen:

Do filename

Where filename is the name of the file which will be activated, CO-DIR will then deactivate itself and the file will be activated.

4.14 <K> - KILL FILE

Required response : <Y>
Exit command : any other key

To kill a file, move the cursors over the file you want to kill then press <K>. A sharp tone will sound. This same tone is used with the <P> and <Z> to attract you attention that a potentially dangerous command is about to be performed (you don't want to kill a file by accident).

A graphic block will appear to the right of the file which will be killed. Press <Y> to kill the file. Pressing any other key will cancel the request.

If you press <Y> the graphic block next to the file will be replaced with an asterisk (*) to indicate that the file will be killed, If the file is killed, a low tone will be heard to indicate completion of the command.

4.15 EXAMPLE OF KILL:

If you are looking at a Directory of drive 1 and you want to kill a file called DATA/OLD, follow this procedure: Move the cursors over the file then press <K>, then press <Y>. CO-DIR will attempt to kill the file on*U that*U drive*U and if all goes well (no write protect tabs etc.), a low tone will be heard and an asterisk (*) will appear on the right hand side of the file and remain there to indicate that the file was killed. The cursors will automatically move down over the next file available if there is one, otherwise they will remain blinking over the file you just killed.

The asterisk marks the file as Killed. All options dealing with files can not be preformed on that file from this point on (you can't List, Rename, Copy etc. a file which is no longer there). For example, if you move the cursors over the killed file and press L to list it, CO-DIR will ignore the command.

4.16 <L> - LIST FILE

Required response : <Y>
Exit command : any other key

This list is different from the BASIC list. CO-DIR uses the DOS library command LIST, not the BASIC list command. This command is mainly for viewing text files (word processor files). To list a file, move the cursor over the file you want to list then press the <L> key. The following message will appear on the command line:

List file?

Press <Y> to list the file, any other key to exit. If you press <Y> the screen will clear and the following message will be displayed on the top line of the video:

List filename

"filename" will be the name of the file you are listing. The file will then be listed.

You can pause the listing by pressing <SPACE BAR> then any other key to continue. To stop the listing before it would normally stop, press the <BREAK> key.

After the listing (whether you stopped it ahead of time or not) a low tone will sound and the following message will appear:

Press <BREAK or CLEAR> for DOS, any other key for CO-DIR

If you press <BREAK or CLEAR> CO-DIR will exit to DOS, otherwise the Directory page that was displayed before the List command will be displayed and the cursors will be blinking over the same file that was just listed.

4.17 <M> - MODIFY FILE

Required response : <Y>
Exit command : any other key

This option allows you to modify the specified file using the file editors supplied with your DOS diskette. MODIFY causes the program DISKDUMP to be loaded and automatically displays the first sector of the file you want to modify.

CAUTION:
This option is supplied as a convenience for people who know how to use the programs above. If you have never used these programs you should turn off this option using the CODIR2/MOD program. A file can be damaged badly by using DISKDUMP if this program is not used correctly!

To use the <M> option, users need DISKDUMP in one of the drives. If you have renamed DISKDUMP you will need to rename them to their normal names.

Users will see:

Diskdump file?

Respond with a <Y> to modify the file, any other key to cancel the command. After you modify the file and exit the program the following message will appear:

Press <BREAK or CLEAR> for DOS, any other key for CO-DIR

If you press <BREAK or CLEAR> CO-DIR will exit to DOS, otherwise the Directory page that was displayed before the Modify command will be displayed and the cursors will be blinking over the file you just modified.

4.18 <O> - LOAD FILE

Required response : <Y>
Exit command : any other key

This command loads the file into memory. The DOS library command LOAD is used, not the BASIC load command. Move the cursors over the file you want to load then press <O>. The following message will appear on the command line:

Load File??

The double question marks (??) are purposely displayed to help differentiate

between the <L>IST command because you may press <L> for LOAD instead of <O>.

Press <Y> to load the file into memory. CO-DIR deactivates itself after this command because the file being loaded may use the same memory area used by CO-DIR. After this command the DOS prompt will appear.

4.19 <P> - PRINT FILE ON PRINTER

Required response : <Y> and printer available
Exit command : any other key

This command will print the file to a printer. It is just like the LIST command except the output is sent to the printer. After pressing <P> the following message will appear on the command line:

Print file? (check printer)

If you have a printer hooked up and ready to receive the data then press <Y>. If you don't have a printer and press <Y> the computer will lock up (just as it does when an LPRINT command in BASIC is used without a printer).

After the file is printed, a low tone will sound to indicate completion of the command and the cursors will resume blinking over the file just printed.

4.20 <R> - RENAME FILE

Required response : new file name
Exit command : <BREAK>, <CLEAR>, <SHIFT> <LEFT ARROW> or <ENTER> if nothing was typed

After pressing <R> the following message will appear on the command line:

Rename to /   /

The slashes represents thin vertical graphic lines which will appear on the screen. These lines indicate how much room you have to enter the new file name.

To rename the file, type in the new name then press <ENTER>. You can use the <LEFT ARROW> to erase mistakes. If you try to pass the graphic borders by erasing too much or typing in too long a file name, a short high pitch sound will indicate an error. Spaces and most non-alphanumeric characters are not accepted, a sound will also be generated if you try to type these characters.

After you type in the new name and press <ENTER>, the old file name on the screen will be changed to the name just typed in and the file will be renamed. If there are no errors (i.e. bad file name, file already exist, etc) a low tone will sound to indicate completion of the command and a R will appear to the right of the file just renamed. The cursors will then blink over the file just renamed.

To exit the RENAME command press <SHIFT> <LEFT ARROW> or <BREAK> or <CLEAR> or press <ENTER> when there is no characters in the RENAME area (between the slashes //).

4.21 <S> - SEARCH (FROM CURSOR POSITION DOWN) FOR FILE OR
EXTENSION OF FILE ON CURRENT DIRECTORY BEING PROCESSED

Required response : <ENTER> or search string and cursors positioned where search should begin. To search the entire Directory you must press <SHIFT> <UP ARROW> before choosing the search option.
Exit command : <BREAK>, <CLEAR>, <SHIFT> <LEFT ARROW>

This command allows you to search through the files listed on the current Directory being processed for a particular file or part of that file. If the file is found, the cursors will begin blinking over that file.

After pressing <S> the following message will appear on the command line:

Search for /   /

The slashes represent thin vertical graphic lines which will appear on the screen. These lines indicate how much room you have to enter the search string.

There are two types of searches:
1) Search by File Name or part of that name.
2) Search by Extension or part of that extension.

4.22 SEARCH BY FILE NAME:
Position the cursors at the top of the Directory (by pressing <SHIFT> <UP ARROW>) or anywhere within the Directory page you want the search to begin. Type in the search string then press <ENTER>; pressing <ENTER> without any text will cause a search to begin using the last search string you typed in (see SEARCH BUFFER below).

This search will compare each character you type in with the corresponding character of the file name. In other words, the first character you type in should be what you expect the first character of the file to be, the second character you type should match the second character of the file name, etc. If you do not know what a particular character will be for a particular position you can type a question mark (?) and CO-DIR will skip to the next character as it searches the file name.

Here are some examples of what search string could be typed in for CO-DIR to find the example file (even if others are found beforehand it will eventually stop on the required file). There is also an example of one that would not.

File needed: FRIDAY5/F4

YOU TYPE   RESULT

FRIDAY/F4 <ENTER>   Success:
                           Will stop on any file with the
                           same first ten characters
                           i.e. FRIDAY5/F4A

F <ENTER>   Success:
                           Will stop on first file with
                           "F" as a first character.

??I <ENTER>   Success:
                           Will stop on first file with a
                           third character of "I".

```
F?I???/F <ENTER>            Success

DAY <ENTER> ===>           FAIL:
                           This will cause CO-DIR to search
                           for the first file which first
                           three characters are DAY.

???DAY <ENTER>             Success:
                           Will stop on first file with
                           DAY as the 4th 5th and 6th
                           characters.
```

4.23 SEARCH BY EXTENSION:

Type a slash (/) as the first character and type in up to three characters of the extension then press <ENTER>, pressing <ENTER> with no characters pressed to search using the last search string used (See SEARCH BUFFER below).

This search is just like search by File Name except CO-DIR will first scan the file name until it finds a slash / and will then begin to compare character by character. Any file which has this extension will be found regardless of how long the first part of the file name is.

Here are some examples of search by Extension:

```
YOU TYPE                   RESULT
/??D <ENTER>               Will stop on first file having
                           a "D" as a third
                           character of its extension.
                           i.e. HELLOPTI/QWD, C/SRD, GAME/CMD
                           etc.

/ <ENTER>                  Will stop on first file with any
                           extension.

/C <ENTER>                 Will stop on first file having a
                           "C" as the first character of the extension.
```

4.24 SEARCH BUFFER:

CO-DIR remembers whatever you searched for last. For example, if you typed in FR to look for FRIDAY/F4 and the cursors stopped on FROM/TXT; simply press <S> then press <ENTER> with no text. CO-DIR will display FR on the command line to show what it is looking for and will stop on the next file which begins with "FR".

The search string you typed in will remain in the Search Buffer until you type in a new search string or exit CO-DIR. You can go from drive to drive (by using the <D> command) looking for your file and simply press <S> then <ENTER> to look for it over and over again until you find it.

The Search Buffer is set to all question marks (?) when you first activate CO-DIR. If you press <S> and <ENTER> with just question marks in the Search Buffer, CO-DIR will stop on the next file, no matter what it is.

4.25 <Z> - ZERO (CLEAR) FILE

Required response : <Y>
Exit command : any other key

This command will perform a DOSPLUS library command CLEAR on the specified file.

CAUTION:
This will cause the entire contents of the file to be replaced with zeros! Use extreme caution with this option. Once you zero out a file there is no way to recover the data that used to be there (you can't restore it).

The CLEAR command does not kill the file, it modifies it destroying all data. This is something you may want to do to sensitive data before killing it. If you simply kill the file, the data will remain on the disk until that space is used by some other file. To be sure the data can not be used again, use the <Z> command then use the <K> command to kill it.

To Zero a file, place the cursor over the file to be zeroed then press <Z>. A long sharp tone will sound to get your attention since this command will destroy the data on the file. The following message will appear on the command line:

Clear (Zero!) file????

If you press <Y> the file will be cleared and a Z will appear to the right of the file to indicate that it has been filled with zeros, then the cursors will resume blinking over the file you just processed.

4.26 PASSWORD ACCESS:

If you need to process a file which is password protected, simply press then <SHIFT> key and hold it down then press the key for the command you would like to perform. The following message will appear on the command line:

Password : /   /

Type in the password (See <R> - RENAME for an explanation of how to enter characters between the slashes / / and how to exit the command). Once you type in the password, the normal command prompt will appear on the command line. If you activate the function, the password you typed in will be appended to the end of the file before the function is preformed. If you cancel the command then the password will not be used.

CO-DIR does not remember passwords so you will need to type in a password each time it is needed.

NOTE: Pressing <SHIFT> with the following options will have no effect: <A>, <F>, <H>, <S>. Pressing <SHIFT> with the <D> option will cause invisible files to be listed, you will not be asked to enter a password.

5.0   TIPS & GENERAL COMMENTS:

5.1   LAST PAGE:
To move the cursors onto the last page of the Directory Buffer, use the <S>EARCH option. Press <S> then type <@> or <;> then press <ENTER>, since CO-DIR will not find any file that starts with "@" or ";" it will search through the entire buffer leaving the cursors on the last file of the last page.

5.2   LOADING SPEED:
If a file is saved on consecutive tracks, it may be loaded from disk a little faster. You may be able to increase the loading speed of CO-DIR by making sure it is saved on consecutive tracks. The fewer programs which are on the disk when you copy D/CMD on to that disk, the greater the chance CO-DIR will be on consecutive tracks; bear this in mind when you are copying D/CMD onto one of your Dos diskettes.

To see if CO-DIR is not on consecutive tracks, LIST it. If the listing pauses and you hear the disk head moving, CO-DIR is not on consecutive tracks (use the MAP option to check).

5.3   TECHNICAL OUTLINE:
CO-DIR 2.0'S loading address is 8000 Hex. The Directory memory buffer begins at 9000 Hex and will use a minimum of 1K bytes.

Job Control Language

Job Control Language Table of Contents

I. - Job Control Language

DOSPLUS 3.5 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 3.5. 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", followed 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 <u>keyboard queue</u>. 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 capabilities, 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 <u>implicit /READ</u> 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 <u>must</u> be suffixed with a space, and JCL operators (=, EQ, NE, GT, GE, LT, LE) <u>must</u> be prefixed and suffixed with a space.

JCL Commands

The DOSPLUS 3.5 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 <u>require</u> 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 = AB

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 <u>significant</u> 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 occurred, 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 <u>significant</u> 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.

III. - The JCL Command Set

The DOSPLUS 3.5 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 $varl $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 $VARI $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
.Use CRUNCH utility on file
/DOS CRUNCH $FILE/BAS $FILE/CRN
.Display message for operator
/TYPE File: $FILE CRUNCHed
/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, and then the CRUNCH utility is used to compress the file specified by

$FILE (with the /BAS extension added) into another file specified by $FILE (with the extension /CRN added). After the operation is complete, the JCL program types the message "File: filespec CRUNCHed" 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 comparisons of JCL variables and to make decisions based on the outcome of the comparisons. 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
/TVPE 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
-LOOP
/TYPE Enter drive #:
/READ $DRIVE
/IF .$DRIVE EQ . /GOTO -ALLDONE
-LOOP1
/DOS CAT :$DRIVE
/IF $ERR$ NE 00 /GOTO ERROR
/GOTO LOOP
-ERROR
/TYPE
/TYPE Error code $ERR has occurred.
/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 $DRIVE
.create any needed delimiters
/IF .$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 3.5 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 occurred %%
/TYPE %% Procedure aborted %%
/EXIT
-NOERR
/IF $LEVEL NE 5 /RUN RECURSV $DR
/EXIT
```
==

==
/DEBUG

General format: /DEBUG

    This command is used to invoke DOSPLUS 3.5'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 issued.

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 data 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).

NOTES

Disk Basic

Disk BASIC Table of contents

Disk BASIC

Introduction

What is "Disk BASIC"? Disk BASIC is simply that, a set of enhancements to the Level II ROM BASIC that is resident upon the disk. It contains features to allow input/output to disk files for data storage and will allow you to load and run BASIC programs that are stored on the disk. In addition, Disk BASIC will allow many new commands not present in ROM BASIC. Disk BASIC comes in two forms : BASIC and TBASIC.

This manual is designed to document those features contained in these Disk BASICs. For functions concerning the ROM BASIC, you should refer to your Radio Shack owner's manual (for the Model I - "Level II BASIC manual" or for the Model III - "Model III Operations and BASIC Language Reference manual") for descriptions of these commands.

This manual is divided into two major sections. The first section is the standard functions of Disk BASIC that appear in both BASIC <u>and</u> TBASIC. The second section covers those enhanced functions found only in Extended Disk BASIC. The first section is further divided into groups of commands. There are general commands, program file handling commands, and data file handling commands.

There are several files on the disk needed for BASIC and its assorted functions. The following list details the files and their function. Those not needed may be killed.

Filename	Function
BASIC/CMD	The actual Extended Disk BASIC program file.
TBASIC/CMD	The Tiny Disk BASIC program file.
ERROR/OVL	The error message overlay used by extended BASIC for its long form error messages. If this program is removed from the disk, both BASIC and TBASIC will use the same abbreviated style of error messages.
RENUM/CMD	Extended BASIC program renumberer.
REF/CMD	Extended BASIC program cross referencer.
SR/CMD	Extended BASIC Global search and replace utility.
O/CMD	Extended BASIC array sort utility.

The actual operation of RENUM, REF, SR, and O will be detailed in the section of the BASIC manual devoted to the extended features.

Entering BASIC

To execute Disk BASIC, from the DOS command mode type either "BASIC" or "TBASIC" depending on which version you desire. The differences are

BASIC	Extended features & DOS commands. Uses "overlay" structure for enhancements and error messages.
TBASIC	Memory efficient, leaving you with as much free memory as possible. Command compatible with most programs. When loaded, TBASIC is completely memory resident, no overlays are used.

You must specify, upon calling BASIC, the number of files (greater than one) you expect to use and the amount of memory (if any) that you wish to protect from BASIC. This is true for either BASIC or TBASIC. There is a method of permanently altering the default for the number of files, but we will cover that later. The DOSPLUS distribution copy of BASIC and TBASIC default to 1 file buffer allocated.

If you are not going to be opening any data files while running your program and desire the extra memory, you may specify "0" files when executing BASIC or TBASIC.

===

You use the following general syntax :

BASIC filespec-F:nn-M:nnnn

> <u>filespec</u> is an optional file specification. BASIC will load and execute this file after loading itself.

> <u>-F:nn</u> is the number of files you expect to be using at one time (can be from 0-15). You may not specify a higher file number during an OPEN than you specify here.

> <u>-M:nnnn</u> is the highest memory address that BASIC will use. You must use this to protect any machine language routines that you might wish to load into high memory. This cannot exceed the high memory address.

===

Some examples (in these examples, we will use the program name "BASIC". you may however, substitute "TBASIC" for any of these and the syntax is still proper.) :

> BASIC

Load BASIC with 1 file, no protected memory.

> BASIC *

Re-enter BASIC (in the event you exited to DOS and have not changed user memory while there), with your program intact. Files allocated and memory protected will be the same as when BASIC was exited. Note that upon re-entry, BASIC will commence listing the program lines as confirmation of the successful return to BASIC. You may stop this listing by pressing BREAK.

> BASIC filespec

Load BASIC and run the BASIC program specified by "filespec".

> BASIC -F:3

Load BASIC and allocate 3 files.

BASIC -M:61000

Load BASIC and protect Memory above 61000 decimal.

BASIC filespec-F:3-M:61000

Load BASIC, allocate 3 files, protect memory above 61000 decimal, and load and run the specified BASIC program.

BASIC PAYROLL/BAS-F:4

This will load BASIC, and run PAYROLL/BAS allocating 4 files.

Note : Once you have entered BASIC, in order to alter the memory protection or number of file buffers allocated, you must reload BASIC with the new parameters. This may be done by returning to DOS and re-entering BASIC or, in extended BASIC, using the CMD"DOS command" function to reload BASIC from within BASIC.

Upon entering Disk BASIC, you will see the header :

DOSPLUS - Extended Z80 Disk BASIC - Ver 1.7
(c) 1983, Micro-Systems Software Inc.

READY
>

or

DOSPLUS - Tiny Disk BASIC - Ver 1.7
(c) 1983, Micro-Systems Software Inc.

READY
>

depending on which version (BASIC or TBASIC) you executed.

Once you have entered Disk BASIC, you can return to DOS by simply typing "CMD" and pressing ENTER. For compatibility's sake, "CMD'S'" will also work. If you have exited BASIC in error, you may immediately re-enter BASIC with "BASIC *" and not harm your program or data in memory. This will <u>not</u> apply if you have executed any operation after exiting BASIC that may have corrupted the information still in memory. Therefore, as a rule, always be sure to save off any program resident before returning to DOS.

<u>Note to Model I users</u>

In order for BASIC or TBASIC to operate correctly with cassette I/O, you will need to turn off the interrupts before any cassette I/O and turn them back on when I/O is complete. This includes the loading and saving of programs.

You accomplish this by using the CMD"T" function to turn interrupts off and CMD"R" to turn them back on. These commands may be issued from the command line in the direct mode or from a program line. On the Model III, this is not needed.

General commands and functions

&H (hex constant)

This function will allow you to work directly with hexadecimal values. In the case of memory addresses, this is sometimes convenient. &H is used as a prefix for the number that immediately follows it.

The general command form is :

&Hdddd

dddd is a one to four digit hexadecimal value.

The constant always represents a signed integer. Therefore any number greater than &H7FFF will be interpreted as a negative quantity. For example :

Hex number	Decimal value
&H1	1
&H2	2
&H5200	20992
&H7FFF	32767
&H8000	-32768
&H8001	-32767
&H8002	-32766
&HFFFE	-2
&HFFFF	-1

Hexadecimal values may not be entered in response to an INPUT statement nor included as numeric data in a DATA statement.

Examples

PRINT &H5200

This will print the value 20992 on the screen.

POKE &H3C00,6

This will poke a decimal 65 to the beginning address of video memory.

A=PEEK(&H37E8)

This will read the value of the printer status byte into "A".

DEF FN (define function)

This feature lets you create your own implicit functions. From then on, you need only call that function by name and the function you defined will automatically be performed.

The general command form is:

DEF FN name(var)=exp

name is the variable that will be used as the name of the function. This must be of the same variable type as the variable to be returned from the function (i.e. string, integer, etc.)

var is the variable or variables to be passed to the function. These are dummy variables, used to indicate to the function the number and type of variables it may expect in the function.

exp is the desired function.

Once a function is defined, you may call it simply by referencing "name" prefixed with "FN". When defining functions, bear in mind that you are limited to one statement. All defined functions are a single statement.

The type of variable specified in the "name" position determines the type of variable that will be returned by the function. For example, if "name" was "RN!", the value stored in it would always be single precision even if the function involved only integers.

On the other hand, the type of variable(s) specified in "var" determine what types of variables will be used with the actual function. For example, if "var" was "X%,F$", the function would expect to be passed two variables. The first an integer and the second a string. Therefore, "var" may be thought of a simply a place holding variable.

For example :

DEF FN TV#(X!)=TC!*FA%*100

will ALWAYS return a double precision value in no matter what the precision of the variable used in the actual calculation. It will expect to receive a one single precision variable upon being called. After the DEF FN statement, all you must do to implement it is enter the statement :

V#=FN TV#(TC!)

"V#" will contain the result of "(TC!*FA%/100)". It doesn't matter what variable name you use to pass information to the function or what variable name you use to store the result. For example, "Q#=FN TV#(D!)" would have worked just as well. The variable passed to the function is needed only as a vehicle for giving the function needed data. The important item is the function name, not the result variable or the entry variable. The function must be defined with at least one argument, even if this argument is not actually used to pass a value to the function.

DEFUSR (define entry address for user machine language subroutine)

This will allow you to define the entry point for your machine language USR routines (USR routines are explained later in this section).

The general command form is:

DEFUSR n=address

n is the USR routine number. It is a number between 0 and 9. If it is omitted, 0 will be assumed. You will later reference your routine via this number.

address is the address of the machine language routine's entry address.

Examples

DEFUSR0=&HFF00

This will set the entry point of USR routine number 0 to hex FF00 (dec 65280). When your program calls USR0, control will pass to the subroutine located at hex FF00. Note that if the address is greater than 32767, and you wish to express it in decimal form, you must use the form "address-65536" when determining the value to be used in the statement.

To get a USR routine into RAM, you can either have the actual opcodes in a DATA statement (in decimal form, of course) and then POKE them into memory or you can use an Assembler to create a disk file from source code and then load it into RAM via CMD"LOAD filespec" (from Extended Disk BASIC).

In either case, make certain that you protect the area of memory that the routine is destined for by using the "-M:nnnn" syntax. If you do not, then BASIC may overlay the routine by using the area for string storage or workspace of some kind.

Refer to the section on USR routines for further detail on user routines.

INSTR (string search function)

This will allow you to search any string for a specified sub-string.

The general command form is:

INSTR(pos,targ$,sub$)

pos is the position in the string at which you wish the search to begin. If omitted, position 1 will be assumed. Position 1 is defined as the first character of the string.

targ$ is the name of the string to be searched.

sub$ is the sub-string you want to search for.

INSTR will search a specified target string for a specified sub-string starting at a specified position. You may omit the starting position, if you like, and INSTR will begin with the first character of the target string. If INSTR finds the sub-string within the search string, it will return the starting position of the sub-string, otherwise, it will return a zero.

INSTR will also return a zero if either the target or search string is a null, or if you have specified an illegal value for the starting position.

Note : The entire sub-string MUST be contained within the search string, or zero is returned. Also, it will find only the FIRST occurrence of the sub-string, starting at the position you specify.

Examples

(assume A$="TEST ONE")

Search expression	Resultant A
A=INSTR(A$,"ONE")	6
A=INSTR(A$,"one")	0
A=INSTR(A$,"123")	0
A=INSTR(A$," ")	5
A=INSTR(3,"123123","12")	4

Note that in the second example, "one" was not found because the letters "ONE" in the target string are in upper case. INSTR does not ignore case.

LINE INPUT (input a line of text from keyboard)

This function allows you to input an entire line of text from the keyboard including all characters that normally serve as delimiters. It functions in the same manner as INPUT except that only ENTER serves as a delimiter.

The general command form is:

LINE INPUT"prompt";var$

Prompt is the prompt to be displayed on the screen. It is optional.

var$ is the name of the string in which you wish LINE INPUT to return you the information.

It differs from INPUT in that :

*   When waiting for input, no question mark is displayed.
*   Each line input statement can assign a value to only ONE variable.
*   Commas and quotes will be accepted as part of the string input.
*   Leading blanks are not ignored, they become part of "var$".
*   The only way to terminate the string input is to press ENTER.

LINE INPUT is used when you wish to input a string that contains data normally considered string terminators by the INPUT command (see INPUT).

LINE INPUT will serve well in the event that you need to input a string that includes leading blanks, commas, and quotes.

Examples

LINE INPUT A$

This will input A$ without displaying any prompts or question marks.

LINE INPUT"Type in last name, first ";N$

This will print the prompt on the screen and accept any input into N$. Commas will not terminate the input string.

MID$= (replace portion of a string)

This statement lets you replace any part of a string with a specified sub-string, giving you a powerful string editing capability.

The general command form is:

   MID$(var$,pos,len)=rep$

        var$ is the name of the string to be edited.

        pos is the starting position for the replacement.

        len specifies how many characters will be replaced.

        "rep$" is the string you wish to replace "var$" with.

Note : The length of "var$" is never changed. If "rep$" is longer than "var$", the extra characters at the right of "rep$" will be ignored. However, if you specify the number of characters to be replaced, and this number is larger than the replacement string, then the length of the replacement string overrides the length you specified (e.g. it will not replace more characters than are in "rep$").

   The "len" parameter may be omitted. If it is, the length of "rep$" will serve as the "len" parameter. Please note also that this is only BASIC function that can appear on the left side of the "=" sign.

Examples

(assume A$="ABCDEFG")

Expression	Resultant A$
MID$(A$,3,4)="12345"	AB1234G
MID$(A$,1,2)=""	ABCDEFG
MID$(A$,5)="12345"	ABCD123
MID$(A$,5)="01"	ABCD01G
MID$(A$,1,3)="***"	***DEFG

=MID$ (duplicate a portion of a string)

This function allows you to duplicate a portion of a string into another string without affecting the first string at all.

The general command form is:

new$=MID$(old$,pos,len)

> new$ is the string we will create in this operation.
>
> old$ is the original string containing the information we want to duplicate.
>
> pos is the point of "old$" that we wish to start duplicating information from.
>
> len is the number of characters to duplicate.

This demonstrates the other function of MID$. MID$ can also appear on the right hand side of the "=" symbol. For example, you can say :

B$=MID$(A$,2,3)

This would set B$ equal to three characters of A$ beginning with the second character without affecting A$ at all. In this manner, MID$ becomes an effective method on interrogating each element or sub-string in a string, determining whether or not it needs to be altered, and then replacing it if needed. MID$ is a very powerful and extremely useful BASIC programming tool.

For example, if we wanted to determine if a date input was in the "MM/DD/YY" format, we could use the two forms of MID$ to make certain that the third and sixth characters of the input string were slash marks ("/").

Assume that DATE$="01/02.83". We would execute the following statements :

SL$="/"
IF SL$<>MID$(DATE$,3,1) THEN MID$(DATE$,3,1)=SL$
IF SL$<>MID$(DATE$,6,l) THEN MID$(DATE$,6,1)=SL$

The first statement creates a string to be used for comparison, and if needed, replacement. The second statement checks the third character of the screen to determine if it is a slash mark. If it is not, the character is replace with a slash mark. The third statement does the same thing for the sixth character. When these three statements had been executed, DATE$ would equal "01/02/83". Note that in the first part of the statements, MID$ appears on the right hand side of the "=" and in the second, on the left.

USR n (call to user's external subroutine)

This feature allows you to transfer control from BASIC to a machine language subroutine located somewhere in memory.

The general command form is:

USR n(var)

n is the USR routine number. It may be from 0-9. If omitted, 0 is assumed.

var is the variable to be passed to the USR routine.

When a USR call is encountered in your program, control is transferred to the USR routine at the address specified in your DEFUSR statement. This address specifies the entry point to your machine language routine.

Note : If you should attempt to call a USR routine before entering a DEFUSR statement, the error "Illegal function call" will occur.

You can pass one argument directly to and from the routine via the USR call itself, and others can be POKEed into RAM, manipulated by the USR routine, and then PEEKed out by BASIC after returning.

For example :

A=USR1(X)

This passes the value in "X" to the USR routine number one. This USR routine must be defined earlier in the program.

To pass arguments :

POKE values into reserved locations of memory and the machine language routine may retrieve them when it needs them. Have the machine language routine also store its results in reserved memory and the when BASIC gets control back, you can PEEK out the values stored there. This is the ONLY way to pass two or more arguments back and forth from a USR routine.

You can pass one argument as the value from the USR routine, then use special ROM calls to get this argument and return a value to BASIC. This method is limited to sending one variable to and from the routine.

Technical notes

Upon entry to your USR routine, DOSPLUS Disk BASIC will have set up the following information for you :

A=   variable type
HL=> floating point accumulator
DE=> string address

The A register will contain a value that indicates the type of variable passed as an argument. They are :

Value	Type
2	Integer
3	String
4	Single precision
8	Double precision

The HL register pair will point to the floating point accumulator. For numeric arguments, this will contain the actual value.

The DE register pair will point to the string address when a type 3 variable is present. Therefore, if the A register contains a 3, the DE register tells you where to find the string.

At the location DE points to, you will find three bytes of information. The first byte is the length of the string and the second and third bytes contain the actual location of the string in memory (LSB/MSB format).

ROM calls

When passing integer variables to and from your USR routine, there are two ROM calls that may be of service. They are as follows :

CALL 0A7FH   Puts the USR argument in the HL register pair; H contains the msb, L contains the lsb. This CALL should be the first instruction of your routine (if you are going to seek to pull the argument from BASIC).

JP 0A9AH   This sends the integer stored in HL to the output variable of the USR routine. If you don't care about the result of the routine, you may simply execute a "RET" to get back to BASIC instead of this jump.

Disk related functions

This next section will cover those functions directly relating to disk I/O.

There are really two areas to this. The first is file manipulation. The second is file access. File manipulation deals with addressing the file as a whole entity, while file access concerns itself with addressing specific records within the file. We will cover each in turn.

Commands under file manipulation

KILL	Delete a program or data file from the disk.
LOAD	Load a BASIC program from the disk.
MERGE	Merge an ASCII-format BASIC program on disk with one currently resident.
RUN	Load and execute a BASIC program stored on the disk.
SAVE	Write the BASIC program currently resident to disk.

Commands and functions under file access

Commands

OPEN	Open a file for access.
CLOSE	Close access to the file.
INPUT#	Read from a file in sequential mode.
LINE INPUT#	Input a line of data from a file in sequential mode.
PRINT#	Write to a file in sequential mode.
GET	Read from a file in random access mode.
PUT	Write to a file in random access mode.
FIELD	Assign field sizes and names to a random access record buffer.
LSET	Place value in specified buffer field, adding blanks on the right side to fill field.
RSET	Place value in specified buffer field, adding blanks on the left side to fill field.

Functions

CVD	Restore double precision number to numeric form after reading from a file.
CVS	Restore single precision number to numeric form after reading from a file.
CVI	Restore integer to numeric form after reading from a file.
EOF	Check to see if "end of file" was encountered during.
LOF	Return number of logical records in a file.
MKD$	Convert double precision number to an eight byte string so that it can be written to a file.
MKI$	Convert integer number to a two byte string so it can be written to a file.
MKS$	Convert single precision value to a four byte string so that it can be written to a file.

KILL (delete a program or data file from the disk)

This command will kill a program or data file from the disk.

The general command form is:

KILL"filespec"

<u>filespec</u> is a standard DOSPLUS file specification.

This command functions essentially the same as the DOSPLUS library command KILL (with the exception that you may not use BASIC's KILL to kill a device). If no drive specification is made during the calling of the command, it will do a global search and delete the first occurrence of the file. If the specified file is not found, a "File not found" error will be returned.

<u>Examples</u>

KILL"PAYROLL/BAS"

This command will search for the program "PAYROLL/BAS" delete it.

KILL"ACCREC/DAT:2"

This will seek to delete the file "ACCREC/DAT" from drive 2. If it does not find the file on drive 2, an error will be generated.

Note : Do <u>not</u> KILL an open file. Although DOSPLUS will make provision for this and the diskette will not be permanently harmed, it is NOT a good programming practice and should not be done.

LOAD (load a BASIC program file from disk into memory)

This command will allow you to load BASIC programs stored in either the standard compressed format OR ASCII format from the disk into memory.

The general command form is:

LOAD "filespec",option

filespec is a standard DOSPLUS file specification.

option is one or more of three special options you may engage.

Your options are:

Option	Effect
R	Runs program after loading. Any open files will not be closed.
V	Preserves all currently set string and numeric variables. FIELD statements are also preserved.
line number	If the "R" option is used at the same time, program execution will begin with that line.

A simple LOAD without any options will wipe out any BASIC program currently resident, clears all set variables, and closes the files. To load a file with the "R" option still wipes out the current resident program and all set variables, but it does not close the files before it runs the new program. When you load a file with the "V" option it wipes the old program out and closes the files, but does not clear all currently set variables. Using the "R" and "V" options together will only overlay the resident program. All variables, FIELD statements, file buffers, etc. will be unaffected.

A standard load and a load without the "R" option both return to the direct command mode when complete. LOAD"filespec",R is the same as RUN"filespec",R (see RUN).

If you use the "R" and "V" options together, you will have "chained" two programs (i.e. branch from one module to the next without loss of variables or having to re-open the files).

If you attempt to load a non-BASIC file the statement "Direct statement in file" will result.

ASCII loads will be much slower than the standard loads because ASCII must be loaded a byte at a time and each byte must be interpreted separately (ASCII loads are treated just as if the data were being typed in from the keyboard). You do not need to specify ASCII format when loading, only when saving. If any line of the program exceeds 240 characters in length, the error "Direct statement in file" will occur. Because of the lack of tokenized keywords in an ASCII file, program lines have a tendency to expand in length when saved in ASCII.

<u>Examples</u>

        LOAD"ACCREC/MOD"

This command will load the first occurrence of ACCREC/MOD that it finds.

        LOAD"ACCREC/MOD",V

This will load the first occurrence of ACCREC/MOD it finds but preserving all the variables from the previous program.

        LOAD"ACCREC/MOD:2",R ,V

This will chain the program ACCREC/MOD from drive two with the current program in memory without closing files or destroying variables.

        LOAD"ACCREC/MOD",R,100

This command will load the file ACCREC/MOD and run the program starting at line number 100. Had you not specified the "R" option, the line number would have been ignored.

MERGE (merge a program from disk with resident program)

This command allows you to merge a disk file stored in ASCII format with a program currently resident in RAM.

The general command form is:

MERGE"filespec",option

> filespec is a standard DOSPLUS file specification defining a file saved in ASCII format (see SAVE).
>
> option is an optional switch that instructs MERGE not to interrupt program execution after the new file is loaded.

Merge is similar to load except that the resident program is not cleared out before the new program is loaded. Instead, the new program is merged in with the old one.

The program lines from the new program will be inserted into their respective positions in the old program. Any new line numbers that are the same as resident line numbers currently existing will overlay the current lines.

MERGE provides a convenient method of putting modular programs together. For example, often used BASIC subroutines can be saved on the disk in ASCII format and merged in to the program to save re-typing them.

Normally, MERGE ceases program operation and returns to the command level after loading the specified file. However, you may include the "R" option to indicate that you wish to continue after the MERGE. Please note that all files will be closed and all variables erased even if you specify the "R" option. All this option does is prevent MERGE from ceasing program execution.

Programs to be merged must be saved in ASCII format.

RUN (load and execute a program from the disk)

This command will load and execute a file from the disk.

The general command form is:

RUN"filespec",option

filespec is a standard DOSPLUS file specification.

option is one of three special options you may engage.

Your options are:

Option	Effect
R	Any open files will not be closed.
V	Preserves all currently set string and numeric variables. FIELD statements are also preserved.
line number	Runs the program starting at the specified line number.

When this command is selected, any currently resident program will be replaced by the program specified.

If you enter "RUN'filespec',R,V" you will have truly chained the program to be run with the program in memory. The program in memory WILL be replaced, but the variables set by the resident program will not be cleared, nor will the files opened by the resident program be closed when the new program is loaded and run.

Examples

RUN"TSTPGM/BAS"

This command will search for the first occurrence of the file TSTPGM/BAS and upon locating it, load it and run the program.

RUN"TSTPGM/BAS:1"

This command will search for the file TSTPGM/BAS only on drive 1. If it is not located, a "File not found" error will result.

RUN"TSTPGM/BAS:1",R,100

This command will load and execute the file TSTPGM/BAS from drive 1. Any files opened by the resident program will remain open and execution will begin at line 100.

Note : When using the "line number" option, if a non-existent line number is specified, program execution will begin at the next line. If the end of text is encountered first, you will be returned to the BASIC command level.

SAVE (save resident program as a disk file)

This command allows you to save your BASIC programs on disk. You can save the program in either compressed or ASCII format.

The general command form is:

SAVE"filespec",A

        <u>filespec</u> is a standard DOSPLUS file specification.

        <u>,A</u> is the special option you may engage to save a file in ASCII format.

    BASIC programs are stored on the disk in two formats : compressed and ASCII. The compressed format uses less disk space and will load and save faster than ASCII. This is the same format as BASIC uses to store the program in RAM. Be advised that if you save a program to a disk and a file by that name is already existing, that file will be lost as the program you are saving will over-write it.

    Using the ASCII format allows you to do certain things that compressed format will not. For example

        *   The MERGE command requires that the file be saved in ASCII.
        *   You can use the library command LIST to print the file from DOS.
        *   Programs to be used with many compilers must be saved in ASCII.
        *   Files created will be compatible with other systems for conversion.

    To separate your compressed programs from ASCII, you could append the "/BAS" extension for a compressed program and "/TXT" for an ASCII file.

<u>Examples</u>

        SAVE"PAYROLL/BAS:2"

    This command will save the program "PAYROLL/BAS" in compressed format on drive 2.

        SAVE"TOBECOMP/TXT",A

    This will save the file TOBECOMP/TXT on the first drive that has available space and is not write protected.

Note : You may use SAVE from a BASIC program. The program will then save itself and continue. This is useful for self-modifying programs.

<u>File access</u>

This next Section is dedicated to file access commands. As stated earlier, these are the commands that are used to deal with the file and access the actual records of the file. File manipulative commands, such as those just covered, dealt with the file as a whole entity. We will be addressing here those commands that allow you to deal with the file in part (as records).

<u>Initializing files</u>

When you entered Disk BASIC you should have been thinking ahead to this moment. Before running a program, you should have an idea of how many files that program is going to require. The number you specify when entering BASIC is the highest allowed file number.

Each file is assigned a number between 1-15. You will reference it by this number when opening the files. If you had entered BASIC by using the syntax :

    BASIC -F:3

you would have three files numbered 1,2,3.

A file buffer is a Sort of holding tank for data. All data going to or from the disk must be passed through a file buffer. The fact that you have PUT a logical record into a buffer does not necessarily mean that it has actually been physically written to the disk. When you access a file, you must tell BASIC which buffer to use, what type of I/O you intend, and what the logical record length is.

These items are handled by opening and closing a file.

OPEN (open a random or sequential disk file)

This command allows you to assign a file number to a file and initialize it for I/O.

The general command form is:

OPEN"mode",filnum,"filespec",lrl

"mode" is the type of I/O that you intend for the file. Only the first character is important and will be used. If it is a literal, it must be encased in quotes.

filnum is the file number (1-15) that you wish to assign this file. This can also be a variable if you wish.

"filespec" is the standard DOSPLUS file specification of the file that you are initializing.

lrl is the logical record length of the file. This is a value between 1 and 256. Only 1-255 may be specified. The default value is 256, and if it is desired, the lrl parameter should be omitted.

This will initialize a file I/O buffer for disk access. Let's examine each parameter in more detail.

The "mode" parameter is the access mode for the file. There are three :

Character	Mode
R	Random access mode
I	Sequential input mode
O	Sequential output mode
E	Sequential output extension mode
D	Direct access mode (equivalent of "R")

If a file opened for random access, "R", does not exist, BASIC will create it. If it does exist, merely opening it does not affect existing data. The file could be closed immediately without harming the data.

If a file opened for sequential input, "I", does not exist, BASIC will return with an error. You cannot input from a file that does not exist. You also cannot input more data from a file than exists. If you try and read a record from a random access file and that record is not there, BASIC will return a buffer full of zeros. If you attempt this with a sequential input file, you will receive an "Input past end" error.

If a file opened for sequential output, "O", does not exist, BASIC will create the file. When using this mode, exercise caution that you do not unwittingly erase the current contents of a file. For example, if you were to close the file again immediately, the directory entry will be updated as if the file has no information in it.

The sequential output extension mode, "E", is identical to the standard sequential output mode except that when it opens a file, BASIC positions to the end of file and all new data is added from that point. This allows you to add data to a sequential file without having to load the file into memory, add the new data, and write the file back to the disk. Using this mode with a file will not result in the loss of current data.

The "filnum" parameter is the desired buffer number. This may be either an integer constant or variable. It cannot be greater than 15. It also cannot be greater than the number you specify when entering BASIC.

Once a file has been opened with a particular file number, that file number may not be used to open another file until the current one is closed. However, under DOSPLUS BASIC, you may specify another buffer for the same file using another number if this is desired.

The "filespec" parameter is the standard DOSPLUS file specification (see File and Device specifications) that you wish to open for input or output.

The "ln" parameter controls the logical record length of the file to be accessed. A physical record is defined as being one disk sector or 256 bytes. The number of logical records is not always equal to the number of physical records. It can be greater, but it will never be smaller. Logical record length is only used with random access files. Sequential files will always use a directory logical record length of 256. The file blocking is handled internally.

You do not have the option of specifying a logical record length greater than 256 bytes. You may specify a logical record length as small as one byte. Logical records can and do span sectors. The DOS will handle all needed file blocking for you and maintain the specified logical record length.

If you have a logical record length of 64, you would have four logical records for every one physical record. This means that you would have to write, via the PUT command, four records before one disk buffer is filled up and the data is automatically transferred to disk and the buffer cleared for more input/output. DOSPLUS only writes to the disk when it HAS to. Unless the buffer is full or you CLOSE the buffer, DOSPLUS will write to the buffer in memory, greatly increasing the I/O speed.

Examples

        OPEN"O",1,"DAILY/DAT"

This will open for sequential output the file DAILY/DAT on the first non-write protected drive. If the file does not exist, it will be created. If it is already there, the previous contents will be lost. File one will be assigned for access.

        OPEN"I",2,"COINFO/DAT:2"

This will seek to open for sequential input the file COINFO/DAT on drive 2. If the file does not exist, an error will be returned. File two will be assigned for access.

        OPEN"R",l,"DATABASE/ASC:3",24

     This will open for random access the file DATABASE/ASC on drive 3. If the file does not exist, it will be created. If the file does exist, previous contents are not necessarily destroyed. The file has a logical record length of 24 and will use file one for access.

        OPEN"E",l,"INVDATA:3"

     This command will open the file INVDATA on drive 3 for sequential output, but instead of having to re-write the entire file, you will already be positioned to the end of file. You may now extend it sequentially.

Note : While a file is open, you will reference it by file number. The filename itself appears only when you open the file.

CLOSE (close any or all open disk files)

The CLOSE function allows you to close any specific file, a set of files, or all files at once.

The general command form is:

CLOSE filnum,filnum,filnum...

> <u>filnum</u> is the file number you wish to close. You may specify numbers between 1 and 15. If numbers are omitted, all open files will be closed.

If you attempt to close a buffer that was not opened, the statement will simply have no effect. When a file is closed, its directory entry is updated.

<u>Examples</u>

CLOSE 1,2,3

This will close file buffers 1, 2, and 3. These numbers may now be re-assigned to other files with further open statements.

CLOSE

This command will close all currently open files.

The following actions will cause the files to close :

<u>Function</u>	<u>Action</u>
NEW	Erasing a program currently in memory.
RUN	Executing a new program.
MERGE	Merging two BASIC programs.
EDIT	Editing a program line.
CLEAR	Clear string space.

Also adding, editing, or deleting program lines and the use of certain Extended Disk BASIC CMD features.

INPUT # (input information from a sequential disk file)

This command is what you will use to read data from a sequential disk file.

The general command form is:

INPUT# filnum,var,var,var...

<u>filnum</u> is the file buffer number of the file that has been opened for sequential input.

<u>var</u> is the variable that you wish to use to contain the information read in from the file. You may specify as many of these as you wish.

The statement inputs data from a file that has been opened for sequential input. When the file is opened, a pointer is set to the beginning of the file. As data is read, this pointer is advanced. If you wish to reset this pointer to the beginning of the file, you must close the file and re-open it.

The format of the data on the disk is not important, nor is the length of the individual elements. The sequential read will continue until a terminating character is reached or until it has reached the end of file.

To read data successfully, you must know what form the data will take. Consider this : BASIC ignores leading blanks when inputting data from disk. It assumes the first non-blank character to be the start of the data item. It will continue to read until a terminating character is reached. Terminating characters vary as to whether you are reading string or numeric data.

<u>Numeric terminators</u>

End of file
255th character
, (comma)
carriage return (ENTER or CHR$(13))
blank/carriage return

<u>Quoted string terminators</u>

End of file
255th character
"(quote)
"/blank/, (quote followed by a space and a comma)
"/blank/carriage return

<u>Un-quoted string terminators</u>

End of file
255th character
,
carriage return

If the first character of data is a double quote, then BASIC treats it as a quoted string variable. All subsequent data (carriage returns and all) will be read into the string until the next double quote.

If the first character of the string is not a double quote, then the string is treated as a non-quoted string, and data is read up till the first terminator. Double quotes are treated as data.

When inputting numeric variables from the disk, BASIC will evaluate them in the same manner as the VAL function. What this means is that if the first character of a variable in non-numeric for any reason, the variable will be assigned a value of zero.

Technical notes :

When putting a comma on the disk to be used as a terminator, the comma MUST be a literal. That is, when you print the comma to the disk, it must be encased in quotes.

Also, if a carriage return is PRECEDED by a linefeed, both will be ignored as terminating characters.

Attempting to input a variable after the internal pointer has reached the end of file from the directory will cause an "Input past end" error.

When BASIC encounters a terminating character, it will scan ahead and attempt to read in as many terminators as it can. This is to make certain that the pointer is accurately set to the beginning of the next variable. It will always seek to take in the largest set possible.

Disk BASIC always reads in a file in 256 byte blocks. The drives will not necessarily run between each read. The current buffer may contain enough data for several variables.

The drives will not come on when a file is closed from sequential input. There is no need to update the directory entry when a file is only being read.

Examples

INPUT# 1,A

This command will attempt to read the numeric variable "A" from the file that is defined for file number one.

INPUT# 2,A$,B#,C%,D$

This will try to input first a string (A$), then a double precision number (B#), then a integer (C%) and then finally another string (D$) from the file specified by buffer two.

LINE INPUT # (input a line from a disk file into a variable)

Inputs a complete line of text sequentially from the disk. This included all punctuation or normal terminators.

The general command form is:

LINE INPUT# filnum,var

> <u>filnum</u> is the file buffer number of the file that has been opened for sequential input.

> <u>var</u> is the variable name of the variable to be used to store the string.

    This function relates to INPUT# in the same manner as LINE INPUT relates to INPUT. This function reads an entire "line" of string data into the specified variable.

    LINE INPUT # will read everything from the first character of the file up to :

*   A carriage return not preceded by a linefeed.
*   End of file.
*   The 255th character (inclusive).

    Other terminating characters will simply be included in the string.

    If the data were a BASIC program saved in ASCII, each line input would read a different line each time you did a line input. For example, after you did a "LINE INPUT# 1,A$" (assuming that the program had been OPENed with buffer one), you would have the first line of the program in "A$". Every time you looped back through that statement you would get the next line of the program in "A$". If you made this a string array, you could read the whole program as data.

<u>Example</u>

      LINE INPUT# 1,D$

    This will read from the file assigned buffer one and store the data read in the variable in "D$".

Note : In DOSPLUS BASIC, when a carriage return is followed by a linefeed, the linefeed will NOT be considered as a terminator and will be included in the next input. DOSPLUS BASIC does not output a linefeed after every carriage return during sequential output.

PRINT # (output data to a sequential file)

This command will output data to a file that has been opened for sequential output.

The general command form is:

PRINT# filnum,USING format$,var,del,var...

<u>filnum</u> is the file buffer number of the file that has been opened for sequential output.

<u>USING format$</u> is the optional format string to define the data format for the print. It will operate in the same manner as the PRINT USING function of ROM BASIC.

<u>var</u> is the variable that contains the data you wish to write.

<u>del</u> is the delimiter that must be placed between each variable that is to be written. This may be a semi colon.

This function will write data sequentially to a specified file. When you first open a file for sequential disk output a pointer is set to the beginning of the file. Therefore, your first PRINT statement places data at the beginning of the file and advances sequentially with each following statement.

PRINT# does not compress the data in any way before writing it to the disk. Everything that is written out is in ASCII format. Even numeric data is written out in ASCII. Because the data is in ASCII and the function is the same as a standard PRINT, punctuation in the actual PRINT# statement is very important. Semi-colons and commas have the same effect (if not encased in quotes) as they do in a regular print to the screen.

For example, if you do a "PRINT #1,A$,B$", the variables "A$" and "B$" will be "tabbed" on the disk. That is, they will be printed with the same number of blanks between them as if they were printed to the screen and tabbed over. If you did a "PRINT #1,A$;B$", the variables "A$" and "B$" will follow each other with no spaces in between.

Now, in the case of numeric data, this would be fine because a trailing blank is a delimiter, but in the case of string data, you will want to have an explicit delimiter on the disk. To do this, you would enter something like :

PRINT #1,A$;",";B$

That would print a comma between each piece of data on the disk. And a comma being a string delimiter, all would be well. Except in the case of a line input. For that, you really need a carriage return between each one. In that case, do this :

PRINT #l,A$;CHR$(13);B$

That prints a carriage return (CHR$(13)) between each entry, and that takes care of a line input. Remember, the first character printed in string data is the delimiter that will affect whether or not it is treated as quoted or unquoted string data when read back via an INPUT or LINE INPUT. You can use the CHR$ function to imbed any sort of control code in the text that you would like.

Also, because this function is identical to the PRINT statement for video output, you also have the USING option. It will operate in the identical manner to the USING statement for a PRINT to the screen. That is, you will define a field to be used for the output data format. For example (when A=123.456) :

PRINT #l,USING"####.##",A

would produce "123.46" on the disk, the same as it would on the screen. This is useful for padding string and rounding numeric output.

FIELD (partition the buffer used with a random file)

This will allow you to organize a random file buffer into fields defined by variables so that I/O to the file can begin.

The general command form is:

FIELD filnum,num AS var$,num AS var$...

> filnum is the file buffer number specified at time of OPEN.

> num is the length for the field.

> var$ is the variable name for the field.

You may specify as many fields as you have need for. Before you can field a file, you must first open it randomly. This assigns it the buffer number specified during a field (i.e. filnum). After you field it, data is ready to pass back and forth from the disk via GET and PUT.

As defined when we talked about OPEN, a random file buffer may have any number of bytes up to 256. But in order to be useable, it must be in variables that can be manipulated. These variables will all be string. Numbers are packed in and converted out of strings by the string packing functions discussed later in the section.

You may re-field as many times as you wish. A field statement does not change the contents of the buffer, it merely changes the manner in which you are allowed access to them. Two or more file buffers can access the same file. Two or more field statements can affect the same buffer area. You may also use a FOR-NEXT loop to field a buffer, especially in the case of an array.

When you assign a variable a name in a field statement, that variable does not use up any string space. The strings are set to point into the file buffer. You do not need to figure in field statements when calculating how much string space to clear for a program.

Now, if you access a variable name on the left side of an equals sign outside the field statement, you remove it from the field statement and place it in the normal variable area. For example, if you :

FIELD 1,23 AS A$

and then later in the same program :

A$=B$

the field variable will be nullified and A$ will be a standard string variable.

After you field a file, you read data from it via the GET statement. You use LSET or RSET to place the data in the fielded buffer so that you can write it to the disk via the PUT command.

<u>Examples</u>

        FIELD 1,23 AS A$,24 AS B$,132 AS FNME$

        FIELD 2,100 AS FIRST$,155 AS SECOND$

    Remember, if you field a file that you have not opened, you will get a "Bad file mode" error. If you field more variables than you have space for in the buffer, you will get a "Field overflow" error.

GET (read a record from a random file)

This command reads a record from a random file and increments the "current" record pointer.

The general command form is:

GET filnum,recnum

       filnum is the file buffer number.

       recnum is the logical record number. If you leave this off, the next logical record will be read.

This statement will read data from the disk into a file buffer. You must first have opened the file and then you may read from it. Before the data is useful, you must field the buffer. After all this, you may get a record and make efficient use of the data.

When BASIC encounters a GET statement, it goes out to the disk and reads in the specified record. If no record is specified, the "current" record is read. In this case, the "current" record is one higher than the last record accessed.

Examples

       GET 1,1

       GET 6

If you get a record with a number higher than the number of the end of file record, BASIC will return a buffer full of zeros and no error will be reported. You may avoid this by first checking the length of file with the LOF function.

PUT (output to a random access disk file)

This command allows you to write a logical record to a disk file that has been opened randomly.

The general command form is:

PUT filnum,recnum

filnum is the file buffer number.

recnum is the specific logical record number that you wish to write. If omitted, the "current" record will be used.

This statement transfers data from the file buffer in RAM to the disk file for permanent recording. Before you can PUT a record, you must have opened and fielded the file, LSET or RSET all the data into the buffer, and then you will be ready to place the buffer onto the disk.

The "current" record is the record one higher than the record last accessed.

Example

PUT 1,2

This command will write the second record of the file.

If the record number you PUT is higher than the length of file record, than the record you PUT becomes the new end of file record. This means that if you "PUT 1,500", not only must it write ALL the way out to record 500, but it must leave space for records 1-499 also. This can eat up disk space in a hurry.

You cannot put a record with the number 0 or with a negative number.

LSET and RSET (place data into an open file's buffer)

This command allows you to place data into a random file's buffer when the file is open. These commands must be used before the PUT command is executed.

The general command form is:

LSET var$=exp$
RSET var$=exp$

var$ is the variable that has been defined via the field statement.

exp$ is the string expression to be placed into the buffer.

When using random files, all data must be stored as string data. This means that any non-string data must be converted before storing. To convert numeric data to string, use the commands MKI$, MKS$, or MKD$. Once you have converted all data into strings, then you may proceed with the storing of this data.

These two statements place string data into a random file buffer before writing it to the disk. Before you can use these commands, the file must be opened and the buffer fielded. Since you are dealing with string data only, any numeric data must be converted at this point.

If you use LSET, it will pad the data into the field variable with trailing blanks. If you use RSET, the blanks will be leading. In the case of either, if the string is too long (i.e. longer than the variable field length), it will be truncated on the right hand side.

The manner in which they justify data into the buffer is the ONLY difference between LSET and RSET. LSET will always left justify the data and RSET will always right justify the data.

Examples

LSET A$=FD$

This command this takes the data currently in FD$ and left justifies it into the file variable A$.

RSET A$=FD$

This will accomplish an identical result with the exception that the data will be right justified into the buffer.

LSET A$=MKI$(DT%)

This takes the integer "DT%" and packs it into a two bytes string which it then places in the file variable A$. Reference the data conversion command MKI$ later in this manual.

MKI\$, MKS\$, and MKD\$ (convert numeric data into strings)

Place numeric data into a string for entering into a random file buffer.

The general command form is:

    MKI$(exp)
    MKS$(exp)
    MKD$(exp)

        exp is the numeric expression to be converted.

These functions take numeric data and alter it into a string. They alter the internal "data-type specifier" so that numeric data can be placed in a string variable.

Function	Returns
MKI$	Two byte string
MKS$	Four byte string
MKD$	Eight byte string

Function	Level of precision
MKI$	Integer
MKS$	Single precision
MKD$	Double precision

Examples

        A$=MKI$(12)

"A$" will now contain a two byte representation of the integer value 12.

        B$=MKD$(1234567.809)

"B$" will now contain an eight byte representation of the double precision value 1234567.809.

Once data is altered in this manner, you may convert it back via the conversion statements CVI, CVS, and CVD.

Bear in mind that you can use LSET and RSET in conjunction with these commands to convert a numeric value into string and place it into the buffer all at the same time. For example :

        LSET FS$=MKS$(1234)

This command will place the four byte string representation of the value 1234 directly into the file buffer as assigned by the file variable FS$.

Note : In the case of ASCII string data, no conversion is needed. For example, you would simply "LSET A$=ZP$" (assuming ZP$ to be a standard string variable), and then PUT the buffer to disk.

CVI, CVS, and CVD (convert string data into numeric values)

This command allows you to return numeric values stored as strings in a random file to their standard form.

The general command form is:

    CVI(var$)
    CVS(var$)
    CVD(var$)

            var$ is the string variable that is to be converted.

These functions are used to retrieve data after it has been stored into string form. They are the exact inverse of their previously discussed counterparts.

Function	Inverses
CVI	MKI$
CVS	MKS$
CVD	MKD$

If the length of the strings to be converted is less than what the functions are seeking, the error "Illegal function call" will result. If the length is greater, extra characters will be ignored.

Function	Length of string
CVI	Two byte
CVS	Four byte
CVD	Eight byte

Examples

In the previous section we showed the example "A$=MKI$(12)". If we now use "DEC%=CVI(A$)", the variable "DEC%" would become equal to 12.

        P#=CVD(TX$)

This will convert the eight byte string "TX$" into a double precision variable (i.e. "P#").

You can use these conversion functions in a mathematical expression (as opposed to the strings that you store on the disk). Suppose that your year to date payroll figures are in a variable "YTD$". It is an eight byte double precision value. You then have a single precision value for the year to date taxes in the variable "TT!". If you try to use "NET!=YTD$-TT!", you will get a "Type mismatch" error. After all, YTD$ is a string. However, you can use "NET!=CVD(YTD$)-TT!" and get a perfectly accurate figure. You can also use the conversion value directly or assign the value to a temporary variable.

Note : In the case of ASCII string data, no conversion is needed. You would simply use "ZP$=A$" (if A$ was an ASCII file variable).

EOF (determine if end of file has been reached)

This command affords you end of file detection. By using the command, you may be returned a value that indicates whether a particular open file buffer is currently positioned at the last record in the file.

The general command form is:

EOF(filnum)

filnum is the file buffer number of the file you are interrogating.

This feature checks to see whether or not you are at the end of a file (i.e. whether or not all the characters in a file have been accessed). This allows you to avoid "Input past end" errors in sequential input.

As long as "filnum" is the buffer number for an active (open) file, then EOF will return a 0 (logic false) if the last record has not been read and a -1 (logic true) if it has.

For example, you could have a program similar to this :

10 OPEN"I",1,"PAYROLL/DAT:1"
20 IF EOF(1) THEN CLOSE:GOTO 100
30 INPUT #1, A$
40 B$(I)A$:I=I+1:GOTO 20
100 REM:REST OF PROGRAM ….

Because EOF returns either a logical true (-1) or a logical false (0), you do not need to check "IF EOF(1)=-1". Simply the logical true/false is sufficient for the IF-THEN statement and it is faster.

The above example would avoid having an "Input past end" error when reading data into the array. It would check before every read to see if it is at the end of file.

LOF (determine the number of records in a file)

This command will return the highest logical record number in any given file.

The general command form is:

LOF(filnum)

<u>filnum</u> is the file buffer number of the file being interrogated.

This function tells you the highest (and last) record number in a file. It will work with both random and sequential files.

However, with a random file, it gives you the number of LOGICAL records. This is only equal to the number of sectors in the file if the logical record length is equal to 256. If it is not, then the LOF will be greater than the number of sectors. In a sequential file, though, BASIC does not know how the data is stored on the disk. With sequential files, LOF simply returns the number of physical sectors.

Often in random access, you wish to look at every record in a file. That program might look something like this :

```
10 OPEN"R",I ,"NAME/INX:2",12
20 FIELD 1,10 AS FNME$,2 AS REC$
30 FOR I=1 TO LOF(1)
40 GET 1,I
50 IF SEARCH$=NAME$ THEN GET 2,CVI(REC$):GOTO 100
60 NEXT I
70 PRINT"NOT FOUND!":CLOSE:GOTO 1000
100 REM:REST OF PROGRAM ....
```

In this manner, you would be certain of looking at all the records.

LOC (determine last record accessed)

This command returns the actual record number of the record last accessed.

The general command form is:

    LOC(filnum)

            <u>filnum</u> is the file buffer number of the file being interrogated.

    This function gives you the record number that you last accessed from the file buffer specified in the argument.

<u>Examples</u>

        PUT 1,LOC(1)

    This will put the data in file buffer number one into the record last accessed from that file.

        PRINT LOC(2)

    This will display the last record number accessed in file buffer number two.

Note : This command is only accurate when working with random files. It deals with logical records.

<u>Disk BASIC error codes</u>

    DOSPLUS Disk BASIC provides you with a complete set of error codes that will indicate exactly what error has occurred. These values may obtained via the ERR function as described in the ROM BASIC manual. The codes are :

Error Code	ERR Value	Error Message
1	0	NEXT without FOR
2	2	Syntax error
3	4	RETURN without GOSUB
4	6	Out of DATA
5	8	Illegal function call
6	10	Overflow
7	12	Out of memory
8	14	Undefined line number
9	16	Subscript out of range
10	18	Redimensioned array
11	20	Division by zero
12	22	Illegal direct
13	24	Type mismatch
14	26	Out of string space
15	28	String too long
16	30	String formula too complex
17	32	Can't continue
18	34	No RESUME
19	36	RESUME without error
20	38	Unprintable error
21	40	Missing operand
22	42	Bad file data
23	44	DISK BASIC feature
24	46	Undefined user function
51	100	FIELD overflow
52	102	Internal error
53	104	Bad file number
54	106	File not found
55	108	Bad file mode
56	110	File already open
58	114	Disk I/O error
59	116	File already exists
62	122	Disk full
63	124	Input past end
64	126	Bad record number
65	128	Bad file name
66	130	Mode-mismatch
67	132	Direct statement in file
68	134	Too many files
69	136	Disk write protected
70	138	File access DENIED

CMD (execute a BASIC command function)

These command functions allow you to perform certain special functions from within BASIC.

The general command form is:

CMD"option"

> option is the single character option that determines which function will be engaged.

Do not confuse this with the extended BASIC CMD function. This function in extended BASIC will allow you to perform DOS commands from BASIC and return to BASIC. At this point we are only concerned with those functions that are standard to both TBASIC and BASIC. The extended form of the command will be covered in the extended BASIC manual.

Commands and options

CMD

This command will return you to DOSPLUS.

CMD"D"

Engage and enter the system debugger.

CMD"E"

Display the last DOSPLUS error message. Many errors that the DOS differentiates between will cause the same error message in BASIC. This command will allow you to interrogate the DOSPLUS error library, which is more extensive than that of BASIC, for clarification.

CMD"R"

Enable interrupts. On the Model I, this command should be performed after tape I/O. See CMD"T" for further details.

CMD"S"

This command will also return you to DOSPLUS.

CMD"T"

Disable interrupts. On the Model I, this must be done before any tape I/O. After the tape I/O, you will have to turn the interrupts back on via CMD"T". On the Model III, this function is performed automatically by the cassette loader routine.

Extended Disk BASIC features

     This section will cover those features found only in extended BASIC. The previously documented commands will be found both in TBASIC and BASIC, but the commands documented from here forward will require Extended Disk BASIC.

Command	Function
CMD	DOS commands from BASIC
DI	Delete and Insert (BASIC program line)
DU	Duplicate (BASIC program line)
Shorthand	Shorthand commands
RENUM	Renumber BASIC program text
TAB	Expanded TAB function
TRON	Expanded trace function
REF	Reference variables, line numbers, or keywords
CMD"M"	Dynamic variable display
SR	Global editing of BASIC text
CMD"O"	BASIC array sort
INPUT@	Controlled Screen Input (string)
Labels	Indirect label addressing
Error messages	Detailed error message display

CMD" (use DOS command from BASIC)

This command will allow you to utilize DOSPLUS library commands and certain utilities from BASIC and return to DOS without exiting BASIC or disturbing resident programs and variables.

The general command form is:

CMD"DOS command"

DOS command is any library command or utility.

The CMD function has been enhanced in Extended Disk BASIC to allow you to execute any DOS command from BASIC and return to your BASIC program with your program and all variables intact. By the use of this function you can execute any of the DOS commands and continue the operation of your program. This can even be done under program control. "DOS command" can be a literal in quotes, OR it can be a BASIC string variable.

Examples

CMD"DIR"
is equal to
CMDD$

When D$="DIR". This means that you can build your DOS commands into a BASIC string and do your own error checking before passing them to DOSPLUS.

CMD"CLEAR "+FS$+" (DATA=E5H)"

Assuming FS$="TEST/DAT", this statement will fill the file TEST/DAT with E5 hex. Notice the use of a BASIC string variable as a DOS command. Notice also the "+" between the variable and the literals. You must use this if you are combining the two.

CMD"FREE :2 TO @PR"

This will output a free space map of drive 2 to the line printer.

CMD"LOAD SUBROUT/OBJ:3"

This will load the file SUBROUT/OBJ and return control to BASIC.

CMD"JOIN @DO @PR"

This will duplicate video output to the line printer.

Note : Certain of the DOSPLUS commands and utilities may use memory outside of the normal DOS command overlay area (such as DIR or CAT with the alphabetical option). This might cause disruption of the BASIC program currently resident. Exercise caution when using this option.

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.

The general command form is:

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.

The general command form is:

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.

SHORTHAND (command abbreviations)

Several new EDIT commands have been added to make it much easier to edit your BASIC programs. You also have certain direct commands in shorthand now. In addition to the standard TRS-80 edit commands, you now have the following:

Command	Function
; (Semi colon)	List first line of program
shift up arrow	List first line of program
/ (Slash mark)	List last line of program
down arrow	List next line of program
up arrow	List preceding line of program
L	Abbreviation for LIST (L10-20)
D	Abbreviation for DELETE (D10-20)
E	Abbreviation for EDIT (E10)
G	Abbreviation for GOTO (G 100)
A	Abbreviation for AUTO (A10,5)
N	Abbreviation for NAME (N TEST)
R or R"	Abbreviation for RUN (R"GAME1/BAS")
L"	Abbreviation for LOAD (L"T1/BAS:1")
S"	Abbreviation for SAVE (S"LOAN/BAS")
K"	Abbreviation for KILL (K"PAY/DAT")
. (PERIOD)	List current line of program
, (COMMA)	Edit current line of program.

These are valid for BASIC only. Use of the above shorthand commands with TBASIC will only result in syntax errors.

Please be certain that any of these commands using non-alphabetic appear as the first character on a line. They must also 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.

Alphabetic shorthand characters (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.

Also, when using the arrow keys to list program lines, occasionally a line will exceed 240 characters in length. When this happens, it will not be listed correctly. Use standard LIST or LLIST to display that line. The line is not harmed in the least.

RENUM (renumber BASIC text)

This allows you to renumber the lines in your BASIC program.

The general command form is:

CMD"RENUM",nln,inc,sln,eln

> nln is the new line number. This is what you wish to call the new line number that your renumbered text will begin with. The default line number is 10.

> inc is the increment you wish to use. The default value is 10.

> sln is the line number that you wish to begin renumbering at. Default value is the beginning of the text.

> eln is the line number that you wish to stop renumbering at. Default value is the end of text.

This renumbering utility will change the number of all BASIC lines and all references to these lines in the program. It will check the line numbering of a BASIC program and if it finds any unlisted line number error it will print an ERROR MESSAGE. To use RENUM, you must first load the program to be renumbered into memory.

The renumberer DOES NOT do a block move of text!

Note : You may also specify an exclamation point in place of the above listed parameters. This will cause RENUM to check the text for numbering errors such as an non-existent line number. When it finds it, it will print the error message. This feature checks the text only! It will not renumber it.

Examples

CMD"RENUM",!

This will check the text for any errors.

CMD"RENUM",10,5

This will renumber the entire text, calling the first line 10 and incrementing by fives. (10, 15, 20, 25, etc.)

CMD"RENUM",100,10,80,150

This will renumber the block of text beginning at line 80 and ending at line 150. It will start with line 100 and increment by 10. Please note : If this would cause the text being changed to overlay another line, RENUM will abort with an error. It will NOT move a block of text.

To move a block of text, delete all but the block you wish to move. Then renumber that block to where you want it and save it to disk under a temporary file name (maybe MOVE/ASC) is ASCII format. Then re-load the program you are working on and MERGE in the temporary file (see MERGE).

If you wish to allow one of the parameters to default, simply don't include it. However, you still must place a comma in its place if you wish to use further parameters. For example :

CMD"RENUM",10,,5,20

The "inc" space is blank and so will use the default value of 10. The only exception is when you have included all the parameters you wanted. For example :

CMD"RENUM",10

Notice that there are no "place-holding" commas. Because the "nln" slot was the only one used, we could stop there.

TAB (TAB to the line printer)

This command will allow you to tab to positions greater than 64 when using LPRINT.

The general command form is:

LPRINT TAB(pos)

pos is the desired tab position (0-255). For reference, see LPRINT in your Level II manual.

This command will allow you to space out further on the line printer than is possible under ROM BASIC. If you are using it with a PRINT command to place an entry on the video screen, it will work the same way that it does in ROM BASIC.

However, if you are using it with a LPRINT command to print an entry on your printer, you can now specify a TAB larger than 64. For example, to print at the 100th character position on your printer you would

LPRINT TAB(100)

This would TAB over 100 spaces before beginning to print. This new feature will make it much easier to format your printer output. LPRINT TABs up to 255 are now legal.

TRON (single step trace function)

This command allows you to engage the system "single stepper".

The general command form is:

TRON

There are no parameters.

This is entered exactly as in ROM BASIC. However, you now single step through a BASIC program one statement at a time. You press any key every time you want to execute the next statement or the <BREAK> key to abort.

The TRON in TBASIC still functions in the old manner. Entering a direct command while in the TRON mode will cause the command to be displayed with a colon in front of it. For example :

RUN

While in the TRON mode will produce ":RUN" on the display. This is normal.

To disengage this function, type "TROFF" and press ENTER.

REF (reference BASIC program text)

This command gives you a comprehensive BASIC program cross referencer.

The general command form is:

CMD"REF",option,option...

<u>option</u> can be any of the parameters legal for this command.

<u>Parameter</u>	<u>Function</u>
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

CMD"REF",K,L,V

This will reference the program for all three. If you specify a P also (CMD"REF",K,L,V,P), it will do the same thing, but it will output it to the line printer. Please note that any ASCII numbers occurring in text will be referenced as line numbers.

To display a single variable you use the "S=" syntax. For example :

CMD"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 :

CMD"REF",S=A$

will only hunt up references to the variable "A" when it is being used as a string variable. And still further :

CMD"REF",S=A$(

will look up only references to "A" as an ARRAY string variable. It will also take complex variable names like :

CMD"REF",S=FINDIT

This will look for all occurrences of the variable "FI". The same syntax applies for a single line number or a single keyword. For example :

CMD"REF",S=PRINT

Will reference all the PRINT statements.

CMD"M" (dynamic variable dump)

This feature will instantly display all the simple variables currently allocated, and what values they currently hold.

The general command form is:

CMD"M"

You can stop the listing with shift , and abort with <BREAK>.

If no variables are set, nothing will be displayed. This is NOT a reference utility. Use REF for that. Also, variables will be PRINTed to the screen. Any video control codes will have their standard effect.

To obtain hard copy, enter :

CMD"M",P

This will dump the variables to the printer.

It will not display arrays. Simple variables only.

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.

The general command form is:

CMD"SR",sexp,rexp,sln-eln

       <u>sexp</u> is the search expression. This can be any valid string expression, a literal, or a combination of both string variables AND literals.

       <u>rexp</u> is the replace expression. This can also be any valid string expression, literal, or combination of both.

       <u>sln-eln</u> 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 CMD"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

CMD"SR","Test"

It will look through the text and list every line with the word "Test" in it. If you type :

CMD"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 :

CMD"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 :

CMD"SR",":",CHR$(10)+":"

This will go through the whole text and insert a line feed in front of every colon.

CMD"O" (sort BASIC arrays)

This command allows you to sort BASIC arrays of any type in ascending or descending order.

The general command form is:

   CMD"O",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 CMD"O" will consider when sorting. A "tag" array, on the other hand, is simply "along for the ride". When CMD"O" 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 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 (-). Do not use commas. After you append the first array with a comma, CMD"O" 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 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.

Example

        CMD"O",100,A$(1)+B$-C$,D$,E$,F$

This command line would instruct CMD"O" 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.

Example

        CMD"O",N%,A$(1)

This will function exactly the same as TRSDOS CMD"O". It will sort "A$" in ascending order starting at element one and proceeding for "N%" elements.

Technical notes : You cannot operate this sort from TBASIC. Also, 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$(EF),RN%(EF)
40 FOR I=1TOEF
50 GET 1,I
60 A$(I)=FNME$:RN%(I)=LOC(1):NEXT I
70 CLOSE
80 CMD"O",EF,A$(1),RN%
90 OPEN"R",1,"MAIL/INX",2
100 FIELD 1,2 AS NR%
110 FOR I=1TOEF
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.

When using this sort, you may have up to ten key arrays and twenty tag arrays for a maximum of thirty arrays total.

INPUT@ (controlled screen input)

This command allows you to input string data from anywhere on the screen with control of format and character entry.

The general command form is:

INPUT@ <pos> ,"prompt",fl,it;var$

<u><pos></u> is the screen position you wish to input at. It will begin here with the prompt string if one is specified.

<u>,"prompt"</u> is the prompt message you wish to have displayed on the screen in front of your input field. This must be a literal.

<u>,fl</u> is an integer expression that defines field length.

<u>,it</u> 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.

<u>;var$</u> 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 1023. 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 or CLEAR to be pressed to proceed. For example :

> "$*" - Alphanumeric field, return when full.

If ENTER or CLEAR alone is pressed (i.e. no input), the return variable will be equal to ASCII 13 or ASCII 31 depending on which key was pressed. Otherwise, this will be suppressed in the actual input data (e.g. if you enter "Test", this will return "Test" and not "Test"+CHR$(13)).

";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 only, it will still come to you as a string and you must get the VAL of it (see VAL in your Level II 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 or CLEAR is pressed to exit and will return the input data in "NA$".

> INPUT@256,40,"$*";SI$

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 256 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.

The general command form is:

    NAME label
    GOTO label
    GOSUB label

> <u>NAME</u> 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 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.

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.

<u>Examples</u>

    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 DEFFNTESTER

This is an example of an invalid label. This label contains the reserved word "DEFFN". BASIC will reject this as a label.

Error messages (detailed error message display)

This feature of DOSPLUS BASIC 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 Extended Disk 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 : NEXT J

In this example, there is a "NEXT without FOR" error in line 30. Our loop is controlled by the variable I and we issue a "NEXT J" command.

The printout will appear something like this :

        NEXT without FOR in 30
        --> : NEXT J

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.

Drivers and Filters

DOSPLUS 3.5 Drivers and Filters Manual Table of Contents

KI/DVR

This is the DOSPLUS keyboard driver which offers features such as variable keyboard debounce and repeat delay, variable key repeat rate, MacroKeys, and character filtering.

==
ASSIGN [FROM] @KI [TO] KI/DVR [USING] filespec (param=exp)

"filespec"   is the name of an optional Macrokey definition file

Allowable parameters are:

DB=value   Debounce delay

RD=value   Delay before key repeat begins

RR=value   Key repeat rate

Abbreviations:
DB   D
==

The DOSPLUS 3.5 keyboard driver offers many special features. Like all DOSPLUS device drivers, it supports character translation using tables installed with the DOSPLUS FILTER command.

The KI/DVR program also provides for access to all standard ASCII codes with the TRS-80 keyboard. The standard TRS-80 keyboard does not provide keys for many special characters in the ASCII character set, but with KI/DVR these characters can be entered from the keyboard. The usual ASCII control codes, control-A through control-Z, may be typed by depressing the <SHIFT>-<DOWN ARROW> key pair at the same time as the alphabetic key whose control character is to be generated. For example, to generate a control-R, press <SHIFT>-<DOWN ARROW> and R.

The ASCII codes not present on the TRS-80 keyboard are generated in much the same manner as control characters; that is, by depressing the <SHIFT>-<DOWN ARROW> keys and another key. The table below details which control keys generate which characters:

Key	Character
Control-6	^
Control-7	~
Control-8	[
Control-9	]
Control-0	|
Control--	_
Control-;	DEL, 7FH
Control-/	\
Control-,	{
Control-.	}

When installing the KI/DVR program with the ASSIGN command, certain timing values may be specified. The first, DB, is used to determine the amount of delay used to suppress multiple characters due to keybounce. Normally, the default value of this parameter will be sufficient, but it may be altered for more or less effect if desired. Note that the value of the DB parameter is <u>not</u> directly proportional to the amount of delay achieved. Rather, the delay is proportional to the <u>product</u> of the least significant byte of the delay and the most significant byte of the delay. Also note that this relationship holds for the other two timing parameters available under KI/DVR.

The RD parameter allows the operator to specify the amount of time that elapses before the keyboard driver begins the auto-repeat function. The default value of this parameter should be suitable for almost every situation, but if desired, this value may be changed with the RD parameter.

The rate at which the auto-repeat feature operates may be altered from the default by the use of the RR parameter. Once again, the default value will most likely be close to optimum, but the RR parameter provides a means of tailoring the driver's "feel" in any manner desired.

The DOSPLUS keyboard driver provides a feature called "Macrokeys". MacroKeys are programmable keys which may contain any number of pre-defined keystrokes. For instance, we might define the "D" key to contain the keystrokes "DIR :1 (S,I,A)". Now, by pressing the <CLEAR> key followed by the "D" key, the entire string of characters, "DIR :1 (S,I,A)" will be quickly and automatically entered into the current input line.

Before KI/DVR can use MacroKeys, a MacroKey definition file must be created. This file may be created using the DOSPLUS BUILD command, or with a word processor. MacroKey definition files consist of one or more Macrokey definition lines, each of which have the following format:

    X=string

where "x" is any key, and "string" is the set of keystrokes which the key "x" shall represent. To enter the example above, of the "D" key containing the characters "DIR :1 (S,I,A)", the following line would be entered:

    D=DIR :1 (S,I,A)\

The backslash character as shown in the line above is used by KI/DVR to represent a carriage return, or the <ENTER> key. Any time it is desired to include a carriage return in a MacroKey, type the backslash character (control-/).

Note that MacroKey definition files may also contain comment lines. Comment lines begin with a period symbol, ".", and are ignored by the keyboard driver.

To use a MacroKey after defining the file and installing the KI/DVR program, press the <CLEAR> key followed by the Macrokey associated with the keystroke sequence to be used. Each time a program requests a character from the keyboard driver, the driver will supply a character from the MacroKey until all the characters in the MacroKey are exhausted, at which time characters will once again be accepted from the keyboard. If it is necessary to use the <CLEAR> key, simply press the <CLEAR> key twice.

MacroKeys may be "linked" together by using the tilde symbol, "~", followed by another MacroKey. For example, take the following two MacroKey definitions:

```
B=BASIC\LOAD"MYPROG/BAS"\~L
L=CMD"FORMS P=66,L=60,W=80"\LLIST\
```

Using the "L" MacroKey will result in the FORMS command being executed from BASIC and will cause an LLIST of a BASIC program in memory to take place. If the "B" MacroKey is used, it will load and execute BASIC from DOS and load the BASIC program MYPROG/BAS. The "L" characters in the MacroKey instruct the keyboard driver to link into the "L" MacroKey and carry on as if <CLEAR>-L had been pressed.

Use caution when linking MacroKeys together; specifically, avoid creating loops in which a MacroKey links back into itself or links into another MacroKey which eventually links back into the first MacroKey. Such loops cannot be terminated, and may be exited only by a system reset.

DO/DVR

This is the DOSPLUS video display driver program. It offers character translation on both the Model J and Model III, and allows use of a special cursor character and lower case support on the Model I.

```
======================================================================
ASSIGN [FROM] @DO [TO] DO/DVR

Allowable parameters are:

UCASE=switch        Enables/disables lower-case support (Model I only)

BLINK=value         Set cursor blink rate (Model I only)

Abbreviations:
UCASE       U
BLINK       B
======================================================================
```

The DOSPLUS display driver supports character translation with tables installed using the DOSPLUS FILTER command. On the Model 1 TRS-80, it also provides the ability to use a cursor character other than the standard underline, cursor blink, and lower-case support. All of these features are quite automatic, and the operator need only perform the ASSIGN command shown above.

The UCASE parameter, if included in the command line, will cause DO/DVR to translate all lower-case alphabetic characters to upper-case before displaying them.

The BLINK parameter provides a means of setting the cursor blink rate. The parameter may he set to any value from 1 to 15, 1 being the fastest rate and 15 being the slowest.

PR/DVR

This is the DOSPLUS printer driver, which offers both parallel and serial printer support, spooling, automatic pagination, indent, and other features.

```
=======================================================================
ASSIGN [FROM] @PR [TO] PR/DVR (param=exp)
```

Allowable parameters are:

LF=switch	Determines whether linefeeds are sent to printer after carriage returns
NULL=switch	Determines whether printer will advance a line when a null line is printed
SPOOL=value	Optional spool buffer size
MAX=value	Optional number of attempts to output character to printer during spooler interrupt
INDENT=value	Number of spaces to indent text
XLATE=switch	Determines whether form feed characters are translated
SERIAL=switch	Determines whether parallel or serial driver is used
DELAY=value	Optional number of null characters after LF or CR
CTS=switch	Optional handshaking parameter
DSR=switch	Optional handshaking parameter
CD=switch	Optional handshaking parameter
RI=switch	Optional handshaking parameter
INVERT=switch	Determines whether serial printer handshaking is normal or inverted

Abbreviations:
```
LF        L
NULL      N
SPOOL     SP
MAX       M
INDENT    I
XLATE     X
SERIAL    S
DELAY     D
CTS       C
DSR       DS
RI        R
INVERT    INV
=======================================================================
```

The DOSPLUS printer driver offers unparalleled flexibility and versatility. Beside supporting the powerful filter table structure of DOSPLUS 3.5, it offers spooling, indent, serial or parallel printer support, and other sophisticated features.

Many printers require that the computer send a linefeed character following each carriage return. If the LF parameter is used, the printer driver will provide linefeeds after each carriage return. This parameter defaults to off.

The NULL parameter is used to inform the driver whether or not it should ignore null print lines; that is, lines that consist of a carriage return and nothing more. When set, the NULL parameter will cause the printer driver to ignore null lines. If the NULL parameter is off, null lines will result in a linefeed on the printer. The NULL parameter defaults to off.

The SPOOL parameter may be used to specify the size, in bytes, of an optional printer spool buffer. If no value is specified for SPOOL, no printer spooling takes place. If some value is given for SPOOL, the driver will allocate space in memory for the spool buffer, and all output to the printer will pass through the spooler. Spooled printer output means that the computer may output data to the spool buffer faster than the printer itself can accept data.

The MAX parameter is used in conjunction with the SPOOL command to determine how often data is sent to the line printer. On the Model I, the spooler attempts to output data to the printer 40 times each second; on the Model III, it attempts to output data 30 times each second. The MAX parameter determines how many attempts are made to output a character to the printer each of those 30 or 40 times per second. Generally speaking, the more output attempts that are made, the faster the spooler will output data to the printer. However, the as the number of printer output attempts increases, overall system efficiency declines.

The INDENT parameter provides a means of effectively moving the left margin of the printer over any given number of character spaces. For example, if INDENT=10 is specified, this would cause the printer driver to begin each printed line indented 10 characters from the actual left margin of the printer. This parameter defaults to an indent of 0.

The XLATE parameter determines whether form feed and vertical tab characters are translated by the driver into the proper number of linefeed characters to position the printer at top-of-form, or if the form feed and vertical tab codes are passed to the printer unchanged. "Intelligent" printers which are capable of performing top-of-form operations may operate with the XLATE parameter turned off, and printer that cannot perform a top-of-form will require the XLATE parameter on. The XLATE parameter defaults to ON.

The DOSPLUS printer driver supports serial printers as well as parallel printers. By default, the driver uses a parallel printer driver, but by specifying the SERIAL parameter, the serial driver takes effect. With the serial driver engaged, all printer output is directed to the TRS-80's RS232 port. The RS232 parameters (baud rate, word length, stop bits, etc.) must be set with the RS232 library command.

If the serial driver is to be used, the DELAY parameter may be specified in order to make the driver send any number (from 1-255) null characters following linefeeds and carriage returns. Many printers require some number of nulls after receipt of linefeeds and carriage returns in order for the printer mechanism to complete the time-consuming operations. Setting the DELAY parameter to the printer's recommended number of nulls will accommodate such printers.

Four parameters, CTS, DSR, CD, and RI, are used to specify to the serial driver which of the four RS232 status lines must be high (or low) before characters are transmitted to the serial printer. Since many printers signal the computer when they are ready to receive data by changing the status of one or more of these sense lines, these parameters provide a means of informing the driver which lines it should monitor. For example, if data should only be sent to a certain printer when the CTS and DSR lines are high, the driver assignment might be as follows:

ASSIGN @PR PR/DVR,SERIAL,CTS,DSR

The INVERT parameter is used in conjunction with the four handshaking parameters to specify to the driver whether the indicated handshaking lines should be in a high or a low state before data is sent. Normally, the TRS-80 operates in an inverted logic state, although various printers may differ in this respect.

RS/DVR

This is the DOSPLUS RS232 serial interface driver program. Like all DOSPLUS 3.5 drivers, it performs any character translation called for tables installed with the DOSPLUS FILTER command.

==
ASSIGN [FROM] @RS [TO] RS/DVR

There are no parameters for this driver
==

Note that DOSPLUS does not normally have any RS232 driver routines installed until the above ASSIGN is performed or a /CFG file containing the driver, and therefore the driver <u>must</u> be installed before any I/O is attempted using the RS232 interface.

The disk files which we term file disks are created by the FILE/DVR program using the SIZE parameter. With the SIZE parameter we (a) inform FILE/DVR that we wish to create a new file disk, and (b) specify how large we wish the file to be. The value supplied for the SIZE parameter is the desired file disk size in kilobytes. The file creation and the ASSIGN may be performed as one operation. For instance, let's ASSIGN drive 2 to the file disk MYFILES on drive 7, and that we wish MYFILES to be 50 kilobytes in length. The proper command would be:

> ASSIGN :2 FILE/DVR MYFILES:7,INST,SIZE=50

Note the use of the INST parameter, which implies that FILE/DVR has not been previously installed.

Once a disk drive device is ASSIGNed to a file disk, the drive device will function almost <u>exactly</u> as if it were an actual disk drive (except for floppy disk-only functions, such as BACKUP or FORMAT). In fact, you can display a directory or a catalog of the "diskette", read and write files, use the DIRCHECK utility, etc.

After the FILE/DVR program has been installed, additional drives may be ASSIGNed to file disks. For example, assume that FILE/DVR has already been installed in the system, and that we now wish to ASSIGN drive 5 to a file disk named CMDFILES on drive 3. The proper command would be:

> ASSIGN :5 FILE/DVR CMDFILES:3

Note that the INST parameter was omitted from this command as the FILE/DVR program had already been installed in the system.

The primary use of FILE/DVR is on rigid drives, although it is useful on floppy systems as well. FILE/DVR makes possible very efficient use of available disk space, while maximizing directory space. Each file disk may contain up to 48 directory entries, yet the file disk itself occupies only one (more if the file is segmented) directory entry on the diskette upon which it is resident.

DVORAK/FLT

This is a filter file intended for-use with the keyboard, @KI, device.

===

FILTER [FROM] @KI [TO] DVORAK/FLT

===

    This keyboard filter will redefine the TRS-80 keyboard to conform to the layout of the highly efficient Dvorak style keyboard.

EPSON/ELT

This is a filter file intended for use with the printer, @PR, device.

===

FILTER [FROM] @PR [TO] EPSON/FLT

===

This filter file is used with EPSON MX-80 printers in order to translate those character codes which lie in the 128-191 region (TRS-80 block graphics) into codes in the 140-223 region, which the EPSON printer uses to display the TRS-80's graphics characters.

Technical Information

DOSPLUS 3.5 - Model I/III Disk Operating System - Technical Manual

This portion of the DOSPLUS 3.5 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 3.5.

This manual is divided into several sections:

I. - Rigid Drive Partitioning

With DOSPLUS 3.5, 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 3.5 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 3.5) 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 3.5 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 drivers 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 <u>all</u> 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 3.5 allocates space to its files. When DOSPLUS creates space on a diskette for a file, it allocates that space in units called <u>granules</u>, or <u>grans</u>. 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 3.5 system is constructed on an <u>overlay</u> 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 3.5 is composed of several <u>system files</u>. 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 <u>must</u> be present on all DOSPLUS 3.5 system diskettes for proper operation. SYS0 resides from 4000H-4DFFH.

2. 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 4E00H-51FFH. All low overlay group system files should be present on all DOSPLUS 3.5 system diskettes, or improper operation can result.

3. The high overlay group, SYS9-SYS16. The files contain DOSPLUS's <u>library</u> commands, such as DIR, COPY, DO, etc. The high overlay group is resident in the area 5200H-5BFFH. If desired, the user may delete modules from this group, but only <u>at risk</u>. 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	EVAL@, WILD@.
SYS8	RAMDIR@, CAT@, FILPTR@, 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.

III. - Directory Structure

Each DOSPLUS 3.5 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 underline 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 3.5 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 06H. This byte has the value FBH. Converting this byte into its binary equivalent, we get:

FBH = 1111 1011

Recalling that a set bit indicates allocated, or unavailable granules, and that a reset bit indicates free granules, we can see that cylinder 06H has a single free granule, and it is granule number 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 26H 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 26H 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 3.5 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. Therefore, all diskettes formatted under DOSPLUS 3.5 will bear a version code of 35H.

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 <u>master password</u> 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 3.5" and the date is "01/18/83".

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 STARTUP. If a carriage return is present in the first byte of the AUTO command field, the diskette effectively has no AUTO command set.

<u>NOTE:</u> It is possible to place an auto command on the GAT of a <u>data disk</u>, either using the DOSPLUS command AUTO, or by means of a user program. However, the AUTO command is only executed from the <u>system disk</u> 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

```
00: FFFF FFFF FFFF FBF8 FFF8 F9F8 FBF8 FFFF .................
10: FFFF FFFF FFFF FFFF FFF9 FFF9 FFF9 FFFF .................
20: FBF8 F8F8 F8F8 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 FBF8 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 FF35 0500 9642 ...........5...B
D0: 444F 5320 332E 3520 3031 2F31 382F 3833 DOS 3.5 01/18/83
E0: 5354 4152 5455 500D 2020 2020 2020 2020 STARTUP.
F0: 2020 2020 2020 2020 2020 2020 2020 2020
```

Figure 1

Page - T/8

HIT Organization

The <u>hash index table</u>, or HIT is located in sector 1 of the file DIR/SYS, and is one sector in length. DOSPLUS 3.5 uses the HIT to locate files in the directory. When a file is created, the operating system generates a 1-byte <u>hash code</u> 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 <u>hashing</u> 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 <u>logical file number</u>, 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 4BH 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 4BH.

Sample Hash Index Table

```
00: A2C4 2E2F 2C2D 2A2B 2829 2627 27A7 26A6 .../,-*+()&''.&.
10: 25A5 2400 00C1 0000 00F2 C400 0000 A6D9 %.$.............
20: 0000 0000 0000 EE00 00E2 4BDF 5200 3800 ..........K.R.8.
30: 0000 0000 0000 0000 0000 0000 0000 0096 ................
40: 47F8 0000 D4C2 F686 E800 B130 4800 9EBA G..........0H...
50: FE00 5EB7 C4E4 00C4 0000 0000 EC00 001C ..^.............
60: 0000 0000 0000 43C4 AF00 0029 EA00 00A0 ......C....)....
70: 0000 0014 0000 0000 0000 0000 0000 0000 ................
80: 00D2 00E9 C344 4007 0000 8E4B 9200 0044 .....D@....K...D
90: 00F0 0000 00D8 0000 0000 2600 C500 0000 ..........&.....
A0: 0000 00A6 0000 8044 0000 0000 0000 0000 .......D........
B0: 0000 0000 0000 0000 0000 0000 0000 0000 ................
C0: 0000 00BE 00AF 00B3 F44D 0000 00E8 0000 .........M......
D0: 0000 00C3 0000 0000 00C6 7300 0000 0056 ..........s....V
E0: 0000 0000 0000 0029 0000 0000 00C3 0000 .......)........
F0: 0000 00FB 0000 0000 0000 0000 0000 0000 ..........0.....
```

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:

LFNByte: <u>0 1 0</u> <u>0 1 0 1 1</u>
               | |
          Entry # Sector #-2

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 4BH

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 4BH indicates that the file primary directory entry is located on directory sector 0DH in entry 2.

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 -
		Bit 6: - Unused -
		Bit 5: - Unused -
		Bit 4: - Same -
		Bit 3: - Unused -
		
		Bits 0-2: - Unused -

Entry+01H	Flags
	Bit: 7: 0=Shrinkab1e, 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 <u>system files</u> and <u>user files</u>. Only DOSPLUS 3.5 /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 3.5, 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 3.5 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 3.5 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 <u>bootstrap</u>, is to provide a small program which is used to load the DOSPLUS 3.5 system file SYS0/SYS. When the computer is booted, a routine in the machine's ROM reads a sector from cylinder 0 of the system diskette (sector 0 on Model-I TRS-80's, sector 1 on Model-III's) into RAM at 4200H on the Model-I, 4300H on the Model-III. 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 4200H (Model-I) or 4300H (Model-III). 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 <u>alternate system driver</u> 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 3.5 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 3.5 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 <u>alternate system driver</u>. 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 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 Table & DCB Organization

Device Control Blocks (DCBs)

A <u>device control block</u>, or <u>DCB</u>, is an area of RAM that contains certain data that is used to control the flow of data to and from character-oriented devices. Six devices are available under DOSPLUS 3.5: @KI (the keyboard), @DO (the video display), @PR (the printer), @RS (the RS232 serial interface), and two user-defined devices, @U1 & @U2.

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: Extended DCB. 0=Normal, 1=Extended
            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

The first byte of all DCBs contains 8 flags that describe the current status of the device. Bit 7 is used to indicate that one or more of three special conditions are in effect: the device is JOINed to another device or file, the device is FORCEd to another device or file, or the device is NIL. If any one (or more) of these conditions are true, bit 7 will be set. Otherwise the bit will be reset.

Bit 6 is used to inform the device driver that a filter table is installed for the device and that it is active. It is the driver's responsibility to translate any characters passing to or from the device through the filter table.

Bit 5 is set whenever the device is currently JOINed to another device or a file. When this bit is set, bit 7 must also be set or the JOIN will not be performed.

Bit 4 is used to flag an active FORCE to another device or a file. If bit 4 is set, bit 7 must also be set or the FORCE will be ignored.

Bit 3 is used to indicate that a device is currently <u>NIL</u>. A NIL device does not output characters, nor does it pass any input characters. If bit 3 is set to flag a NIL, bit 7 must be set or the NIL will have no effect.

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 <u>control</u> data. By control, we mean any data transfer operation that does not fall into the classifications of <u>input</u> or <u>output</u>. 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.

The second and third bytes of a DCB 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 subroutine, called <u>character input/output</u>, or <u>CIO</u>. This routine prepares for input or output from or to the device and then transfers control to the driver address contained in the DCB.

During an active JOIN or FILTER, the contents of DCB+01H and DCB+02H are replaced with a pointer to the destination DCB (in the case of a FORCE) or with the address of the linking device's ENTRY+04H position in the DCBTBL (in the case of a JOIN). The old driver address, and the original contents of DCB+00H (the DCB type) are preserved in the DCB table, described below.

The DCB Table

DOSPLUS 3.5 maintains a table that references the DCBs of all six character-oriented system devices. This table, called DCBTBL, consists of six 11-byte entries. Each entry has the following structure:

Byte	Data
0,1	DCB address
2,3	Filter table address
4,5	Linking DCB address
6,7	Driver address (if JOINed, FORCEd)
8	DCB type (if JOINed, FORCEd)
9,10	Device name, 2 character

Bytes 0 & 1 contain a pointer to the actual device control block. If this pointer contains a value of FFFFH, the device has no currently defined DCB.

Bytes 2 & 3 point to any filter table that may be installed in RAM for the device. Refer to section X, Writing Drivers for DOSPLUS 3.5 for details on the filter table.

Bytes 4 & 5 are used during a JOIN to point to the address of the linked device's DCB. For instance, if the @KI device were JOINed to the @RS device, this byte in @KI's DCBTBL entry would contain a pointer to the @RS DCB.

Bytes 6 & 7 are used during an active JOIN or FORCE to store the device driver address. This address is displaced from the DCB during a JOIN or FORCE when a linking device DCBTBL ENTRY+04H address is installed (for a JOIN), or when the address of a destination DCB is inserted (for a FORCE).

Byte 8 stores the device's DCB type as it was before a FORCE, JOIN, or a KILL modifies it. It is used to restore the status of the DCB type after FORCE, JOIN, or NIL have been terminated.

The last two bytes in each DCBTBL entry contain the two-character name of the device. The device name is left-justified and padded with a blank if less than two characters in length.

DOSPLUS 3.5 supports six character-oriented devices. Two of these devices, @U1 & @U2 are user-defined and have no pre-defined DCB area within the system. @RS also has no fixed DCB address within the DOS. The three remaining devices, @KI, @DO, & @PR all have DCB locations reserved in system RAM. If desired, the user may relocate and redefine the DCBs. The information presented below refers to the standard location of these DCBs and the meaning of the information within them.

===
Device name: @KI
DCB address: 4015H

     Byte   Meaning
     DCB+00H   DCB type (1 byte)
     DCB+01H   Driver address (2 bytes)
     DCB+03H   Reserved (1 byte)
     DCB+04H   Caps lock switch. Zero=No caps, Non-zero=Caps (1 byte)
     DCB+05H   Cursor blink count. (1 byte)
     DCB+06H   Cursor blink status. Zero=Off, Non-zero=On (1 byte)
     DCB+07H   Cursor blink switch. Zero=Blink, Non-Zero=No blink (1 byte)
===
Device name: @DO
DCB address: 401DH

     Byte   Meaning
     DCB+00H   DCB type (1 byte)
     DCB+01H   Driver address (2 bytes)
     DCB+03H   Cursor position (1 byte)
     DCB+05H   Cursor switch. Zero=Off, Non-zero=On (1 byte)
     DCB+06H   Cursor character (1 byte)
     DCB+07H   Special chr./tabs switch. Zero=Tabs, Non-zero=Special (1 byte)
                       (Model-III only)
===
Device name: @PR
DCB address: 4025H

     Byte   Meaning
     DCB+00H   DCB type (1 byte)
     DCB+01H   Driver address (2 bytes)
     DCB+03H   Page length, in lines (1 byte)
     DCB+04H   Lines printed (1 byte)
     DCB+05H   Character count (1 byte)
     DCB+06H   Maximum width (1 byte)
     DCB+07H   Maximum lines/page (1 byte)
===
    In addition, there exists on the Model-III version of DOSPLUS 3.5, an RS232 initialization DCB, which is arranged as follows:

DCB address: 41F5H

     Byte   Meaning
     DCB+00H   DCB type (1 byte)
     DCB+01H   Driver address (2 bytes)
     DCB+03H   Baud rate code - MSN=Transmit rate, LSN=Receive rate (1 byte)
     DCB+04H   UART configuration code (1 byte)
     DCB+05H   Wait switch. Zero=No wait, Non-zero=Wait. (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 3.5, 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 FCB 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 <u>sequential</u> access. The flag is set when a <u>random</u> 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 <u>end of file</u> byte, or <u>EOF</u>. 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 <u>next record number</u>, or <u>NRN</u>. This is simply the number of the next physical record in the file.

FCB+0CH and FCB+0DH contain the total number of <u>complete</u> sectors in the file. Please note that partial sectors are <u>not</u> 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 3.5 maintains eight <u>drive control tables</u>, or <u>DCTs</u>, 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 a table, called DCBTBL, which contains the addresses of each DCT. DCTTBL consists of eight 4-byte entries arranged as follows:

<u>Byte</u>   <u>Meaning</u>

ENTRY+00H Pointer to DCT (2 bytes)
ENTRY+02H Device name (2 bytes)

The first two bytes of a DCTTBL entry point to the actual location of the DCT. If this pointer has a value of FFFFH, the drive currently has no DCT assigned to it.

The next, and last, two bytes of a DCTTBL entry contain the two-character name of the drive. If the name is less than two characters in length, the rightmost character is padded with a blank.

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 3.5'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
<u>Byte</u>   <u>Meaning</u>
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 3.5 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 <u>log</u> 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.

<u>Perishable data</u>

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 <u>data address mark</u>. 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.

VIII. - System Entry Points

DOSPLUS 3.5 provides the user with many system functions that can be used to good advantage in applications programs, which are detailed in the following pages.

Routine name	Address		Page
Name	**Model I**	**Model III**	**Number**
ABORT@	4030H	4030H	T/30
BKSP@	4445H	4445H	T/28
CAT@	445AH	4419H	T/33
CKEOF@	4451H	4457H	T/29
CLOSE@	4428H	4428H	T/25
CMD@	4400H	4400H	T/32
CMNDI@	4405H	4405H	T/32
DEBUG@	440DH	440DH	T/32
DISKIO%	4485H	4488H	T/44
DIVD@	444EH	4451H	T/36
DSP@	0033H	0033H	T/49
DSPLY@	4467H	4467H	T/38
ERROR@	4409H	4409H	T/32
EVAL@	4479H	4479H	T/39
EXIT@	402DH	402DH	T/30
FEXT@	4473H	444BH	T/29
FILPTR@	4454H	428DH	T/30
FSPEC@	441CH	441CH	T/24
GET@	0013H	0013H	T/48
GTDATE@	4470H	4470H	T/39
GTTIME@	446DH	446DH	T/38
INIT@	4420H	4420H	T/24
KBD@	002BH	002BH	T/48
KEY@	0049H	0049H	T/49
KEYIN@	0040H	0040H	T/49
KILL@	442CH	442CH	T/25
LOAD@	4430H	4430H	T/26
LOCDCB%	4488H	44A0H	T/46
LOCDCT%	448BH	44A3H	T/47
LOCDEV@	447FH	447FH	T/42
MULT@	444BH	444EH	T/36
OPEN@	4424H	4424H	T/25
PARAM@	4476H	4454H	T/37
PEOF@	4448H	4448H	T/28
POSN@	4442H	4442H	T/28
PRINT@	446AH	446AH	T/38
PRT@	003BH	003BH	T/49
PUT@	001BH	001BH	T/48
RAMDIR@	4457H	4290H	T/31
READ@	4436H	4436H	T/26
RESET@	4413H	4416H	T/35
REW@	443FH	443FH	T/27
RUN@	4433H	4433H	T/26
SET@	4410H	4413H	T/34
SORT@	4482H	4482H	T/43
VER@	443CH	443CH	T/27
WILD@	447CH	447CH	T/42
WRITE@	4439H	4439H	T/27

File Handling Routines

All file-handling routines require that the DE register pair point to a valid DOSPLUS 3.5 DCB or FCB (see sections V & VI of this manual). Note that many file-handling routines (FSPEC@, OPEN@, CLOSE@, KILL@) are useful for both file and character-oriented devices.

==
FSPEC@
Address: 441CH (Model I & III)

FSPEC@ is used to move a file or device specification from one area in RAM (typically a command line buffer) into a DCB or FCB. FSPEC@ moves the file or device specification character by character until (a) a terminating character (space, comma, semicolon, carriage return, etc.) is encountered, or (b) an invalid specification character is found. If an invalid character is encountered, FSPEC@ will terminate with an error. FSPEC@ will ignore leading spaces, commas, or the delimiter words FROM, TO, & USING. Note that FSPEC@ does not return a specific error code, but merely error status.

ENTRY: DE=> FCB or DCB to accept file or device spec
            HL=> RAM buffer containing file or device specification

EXIT: BC is altered
            DE=> file/device specification, terminated with ETX (ASCII 03)
            HL=> Next field in buffer
            Flags: Z=No error, NZ=Error
            If NZ, A=Offending character
==

==
INIT@
Address: 4420H (Model I & III)

INIT@ is used to OPEN files and devices for I/O, and may also create new files on a diskette. Before INIT@ is called, the DE register pair should point to a 32-byte FCB or DCB containing the name of the file or device to be OPENed (or created and OPENed, in the case of a new file). If a file is being INITed, the HL register pair must point to a 256-byte I/O buffer, and the B register contains the logical record length under which the file is to be OPENed. A value of 0 indicates an LRL of 256. Upon return from INIT@, the Z register contains error status, and the A register contains the error code, if any.

ENTRY: B =   Logical record length
            DE=> FCB or DCB
            HL=> 256-byte file I/O buffer

EXIT: A =   Error code
            Flags: Z=No error, NZ=Error
                   C=File created
==

==
OPEN@
Address: 4424H (Model I & III)

OPEN@ functions in much the same manner as INIT@, with the exception that OPEN@ will not create a new file. If an attempt is made to OPEN a file that does not exist, OPEN@ will return a "file not found" error.

ENTRY: B =   Logical record length
              DE=> FCB or DCB
              HL=> 256-byte file I/O buffer

EXIT: A =   Error code
              Flags: Z=No error, NZ=Error
==

==
CLOSE@
Address: 4428H (Model I & III)

This routine is used to terminate all I/O to and from a file or device. When CLOSE@ is called, any data that remains in the disk I/O buffer is written to the file, and the length of file is updated, if necessary. CLOSE@ should always be used after a file has been written to, but it is not necessary to use CLOSE@ if data has only been read from the file.

After calling CLOSE@, the FCB or DCB will once again contain the name of the file or device, and in the case of a file, the drive number upon which the file is resident. The password is not restored into the FCB.

ENTRY: DE=> OPEN FCB or DCB

EXIT: A =   Error code
              Flags: Z=No error, NZ=Error -
==

==
KILL@
Address: 442CH (Model I & III)

This routine is used to delete a file from a diskette, freeing the disk space occupied by the file. The file's directory entry is not destroyed by KILL@, and therefore the file may be recovered at a later time with the RESTORE utility. Note that KILL@ must be used with an open FCB, not a closed FCB. After the execution of KILL@, the contents of the FCB are destroyed. Please note that KILL@ cannot be used to kill system devices.

ENTRY: DE=> Open FCB

EXIT: A =   Error code
              Flags: Z=No error, NZ=Error
==

==
LOAD@
Address: 4430H (Model I & III)

    This routine will load an object file into RAM. The file must be saved on diskette in <u>load file format</u>. Upon entry to LOAD@, DE should point to an FCB containing the name of the file to be loaded. After exit from LOAD@, the HL register pair will contain the transfer address of the load format file.

ENTRY: DE=> Unopen FCB

EXIT: A=   Error code
               HL=   Program transfer address
               Flags: Z=No error, NZ=Error
==

==
RUN@
Address: 4433H (Model I & III)

    RUN@ works in the same manner as LOAD@, loading object files in load file format into RAM. After loading the file into memory, RUN@ also automatically begins execution of the program file at the file's transfer address.

ENTRY: DE=> Unopen FCB

EXIT: Control transferred to program file, if successful
               If unsuccessful, routine exits to DOS through ABORT@
==

==
READ@
Address: 4436H (Model I & III)

    This routine is used to read a single logical record from a file into RAM. DE points to the open file control block, and HL points to a <u>user data record</u>. The user data record is not necessary if the file is OPENed with an LRL of 256. In this case, the record is read into the file I/O buffer specified by the user program at time of OPEN or INIT. If the LRL of the file is other than 256, the user must specify a buffer of length=LRL to contain the record to be read from disk.

ENTRY: DE=> Open FCB
    If LRL<>256
               HL=> User data area to receive file record
    If LRL=256
               HL is ignored, data placed in file I/O buffer

EXIT: A=   Error code
               Flags: Z=No error, NZ=Error
==

==
WRITE@
Address: 4439H (Model I & III)

WRITE@ is used to write a single logical record to a file. If the file was OPENed or INITed with a logical record length of 256, the data in the file's I/O buffer, also specified at OPEN or INIT, is written to the file. If the LRL <> 256, then the HL register pair is used to point to a user data record containing the data to be written to the file. As always, the DE register pair points to the open FCB.

ENTRY: DE=> Open FCB
      If LRL <> 256
           HL=> User data record to be written to file
      If LRL 256
           HL is ignored

EXIT: A=   Error code
          Flags: Z=No error, NZ=Error
==

==
VER@
Address: 443CH (Model I & III)

The VER@ routine functions exactly as the WRITE@ routine, above. In addition, VER@ reads the record after writing to the file in order to verify that the write operation was successful. If the record cannot be properly read, an error is reported.

ENTRY: DE=> Open FCB
      If LRL <> 256
           HL=> User data record to be written to file
      If LRL = 256
           HL is ignored

EXIT: A=   Error code
          Flags: Z=No error, NZ=Error
==

==
REW@
Address: 443FH (Model I & III)

REW@ is used to position the file's next record pointer to the first logical record in the file. Before adjusting the contents of the FCB, REW@ writes any residual data in the file I/O buffer to disk.

ENTRY: DE=> Open FCB

EXIT: A=   Error code
          Flags: Z=No error, NZ=Error
==

==
POSN@
Address: 4442H (Model I & III)

This routine allows the user to position a file's next record pointer to any logical record desired. POSN@ will, of course, write any residual data in the file's blocking buffer to disk before moving to a new record. Note that the use of this routine is the only way the <u>random access</u> mode may be set for a file.

ENTRY: BC= Logical record number
               DE=> Open FCB

EXIT: A=   Error code
               Flags: Z=No error, NZ=Error
==

==
BKSP@
Address: 4445H (Model I & III)

BKSP@ is used to position a file's next record pointer to the <u>previous</u> logical record, after writing any residual data in the file's blocking buffer to disk. For instance, if the current record number for a file is 89, executing the BKSP@ routine would <u>backspace</u> the file one record, to number 88. Note that BKSP@ will not read the record into RAM; it merely adjusts the information in the FCB such that the next record read (or written) is the previous record. Note that if one attempts to BKSP@ from record 0, this will result in a current logical record number of FFFFH, or 65,535, and most likely, an end-of-file error.

ENTRY: DE=> Open FCB

EXIT: A=   Error Code
               Flags: Z=No error, NZ=Error
==

==
PEOF@
Address: 4448H (Model I & III)

PEOF@ is used to position the file's next record pointer to the record following the last record in the file. This is useful when it is desired to lengthen a file without first reading the entire file into RAM. Note that PEOF@ <u>normally</u> returns error code 1CH, end-of-file. This error should be ignored, and other errors handled normally. As all other positioning routines, PEOF@ writes any residual data in the file's I/O buffer to disk before moving on to another record.

ENTRY: DE=> Open FCB

EXIT: A=   Error code
               Flags: Z=No error, NZ=Error
==

==
FEXT@
Address: 4473H (Model I) & 444BH (Model III)

    FEXT@ is used to append a file extension to a filename. On entry to FEXT@, the DE register pair should point to the unopen FCB containing a valid DOSPLUS filename (extension, password, and drivespec optional), and the HL register pair should point to a 3 byte file extension. If the extension is less than 3 characters in length, it should be terminated with an ASCII 03H (ETX) or 0DH (CR). When FEXT@ is executed, it will add the extension to the filespec. If the filespec already contained an extension, FEXT@ will have no effect.

    Note that the address of this routine is different for the Model I and Model III versions of DOSPLUS. The Model I entry point is valid under both Model I & III DOSPLUS, however, the Model III entry point <u>is not</u> valid on the Model I.

ENTRY: DE=> Unopen FCB
               HL=> File extension text

EXIT: A=   Error code
           BC is altered
           Flags: Z=Extension added, NZ=Extension not added
==

==
CKEOF@
Address: 4451H (Model I) & 4457H (Model III)

    CKEOF@ provides a method of testing for end-of-file; that is, CKEOF@ will return status in the flags register which indicates whether the file's next record pointer is currently at the end of the file.

    Note that if CKEOF@ determines that the file is indeed at the end of file, an error code will be returned in the A register: 1CH or 1DH, both indicating that the end of the file has been reached.

    Also note that CKEOF@ is present at different locations on the Model I and III.

ENTRY: DE=> Open FCB

EXIT: A=   Error code
           Flags: Z=Not EOF, NZ=EOF
==

System Control Routines

This group of system entry points allow the user to return from a running program to the DOS command prompt, to pass commands to the operating system, display error messages, enter the DEBUG monitor, and display a diskette directory.

==
EXIT@
Address: 402DH (Model I & III)

Executing a jump to this routine will return the computer to the DOSPLUS command level. Note that the user need not preserve the stack when terminating a program via the EXIT@ vector, since the DOS re-initializes the stack on entry to EXIT@.
==

==
ABORT@
Address: 4030H (Model I & III)

The ABORT@ vector functions in a manner similar to that of EXIT@, but if ABORT@ is used to exit a program on a DOS error condition, ABORT@ will invoke the DEBUG monitor, if it is active. In the event of no error, or DEBUG inactive, ABORT@ has the same effect as EXIT@. User programs should exit through this vector upon abnormal program termination.
==

==
FILPTR@
Address: 4454H (Model I) & 428DH (Model III)

FILPTR@ is used to obtain two pieces of information concerning files: The drive upon which the file is resident, and the logical file number (LFN) of its primary directory entry.

Note that the address of this routine is not identical for Model I and Model III DOSPLUS.

ENTRY: DE=> Open FCB

EXIT: A=   Error code
                B=   Drive device number (0-7)
                C=   Logical file number
                Flags: Z=No error, NZ=Error
==

===
RAMDIR@
Address: 4457H (Model I) & 4290H (Model III)

    The RAMDIR@ routine allows a user program to examine the contents of one or more file directory entries, or to determine the amount of free and allocated space present on a diskette.

    Note that the address for RAMDIR@ is not identical for Model I and Model III DOSPLUS.

ENTRY:	B=	Drive device number (0-7)
	C=	Function switch (0, 1-254, 255)
	HL=>	RAM buffer

If C=0:   All FPDE's will be read into the RAM buffer specified by HL. The buffer size must be adequate to contain the total number of active FPDE's on the diskette. The maximum buffer size may be calculated from:

          Bufsiz = (!(Dirsiz-2) x 8" x 22) + 1

          where <u>Dirsiz</u> is the size of the diskette's directory in sectors.

          Each directory entry in the buffer is constructed on the following format:

          ENTRY+00H: 15-character filename, extension, and drivespec. The string
                        is left-justified and padded with spaces on the right.
          ENTRY+0FH: Protection level, 0-6.
          ENTRY+10H: EOF byte.
          ENTRY+11H: Logical record length.
          ENTRY+12H: Ending record number (2-byte, LSB, MSB).
          ENTRY+14H: Unused.

          The last directory entry in the buffer is followed by an ASCII 2BH, "+", to mark the end of the list.

If C=1-254:   single directory entry is read into a 22-byte RAM buffer specified by the HL register pair. The structure of the directory entry is the same as above.

If C=255:   Four bytes of free-space information are stored in a RAM buffer pointed to by the HL register pair. The first two bytes represent the number of kilobytes of disk space allocated to files on the diskette, and the last two bytes are the number of kilobytes free on the diskette. Both values are stored in LSB, MSB format.

EXIT:	A=	Error code
	Flags:	Z=No error, NZ=Error

===

===
CMD@
Address: 4400H (Model I & III)

A jump to the CMD@ vector results in an exit to. the DOSPLUS command level, as does EXIT@.
===

===
CMNDI@
Address: 4405H (Model I & III)

This routine is the DOSPLUS command interpreter. The user may pass a valid DOSPLUS command line (pointed to by the HL register pair) to CMNDI@, and DOSPLUS will execute the command, returning to the calling program afterward. If an error is encountered during execution of the command, CMNDI@ will return to the DOSPLUS command level through the ABORT@ routine.

ENTRY: HL=> Command buffer, terminated with a carriage return, ASCII 0DH.
===

===
ERROR@
Address: 4409H (Model I & III)

This routine posts, or displays, DOSPLUS error messages. The error message number to display is passed to ERROR@ in bits 0-5 of the A register. Bit 6 controls whether DOSPLUS will print a normal or emphasized error message. An emphasized error message is framed in asterisks; ** File not found **• The normal error message is produced when bit 6 is set, and the emphasized message when bit 6 is reset. Bit 7 controls whether the ERROR@ routine returns to the calling program or exits to the DOSPLUS command level through the ABORT@ routine. When set, ERROR@ returns to the calling program; when reset, ERROR@ exits through the ABORT@ vector.

ENTRY: A=   Error code
===

===
DEBUG@
Address: 440DH (Model I & III)

This routine invokes the DEBUG monitor, regardless of the active or inactive status of the monitor.
===

===
CAT@
Address: 445AH (Model I) & 4419H (Model III)

This routine is used to display a diskette directory. The drive number whose directory is to be displayed may be specified in either of two ways: (a) An ASCII-encoded drive device number "0" through "7" may be loaded into the RAM location 442BH (Model I) or 4271H (Model III), or (b) the binary drive device number may be loaded into the C register. Method (a) is compatible with TRSDOS; method (b) makes better sense.

ENTRY: C=   Binary drive device number, 0 - 7
                        OR
    (442BH)=   ASCII-encoded drive device number, "0" - "7" (Model I)
                        OR
    (4271H)=   ASCII-encoded drive device number, "0" - "7" (Model III)
===

Interrupt Chain Routines

DOSPLUS 3.5 provides two routines that are used to insert and delete user interrupt routines from the system's <u>interrupt chain</u>. The interrupt chain is a series of two-byte pointers maintained by DOSPLUS. Each pointer in the interrupt chain is referred to as an <u>interrupt slot</u>. These pointers are used to point to <u>task blocks</u>, which consist of a pointer to the actual interrupt processing task and any other information required by the interrupt routine.

Upon entry to the interrupt task, the IX register will point to the first byte of the interrupt task block. For instance, examine the routine below:

```
                LD         A,0          ;INT SLOT #0, 133.33 MS
                LD         DE,INTTSK    ;INT TASK BLOCK
                CALL       SET@         ;INSERT TASK INTO INT CHAIN
                .
                .
                .
INTTSK          DEFW       TASK         ;POINTER TO TASK ADDRESS
COUNT           DEFW       0            ;COUNTER WORD
;
TASK            LD         L,(IX+2)     ;GET COUNTER LSB
                LD         H,(IX+3)     ;GET MSB
                INC        HL           ;INC COUNT
                LD         DE,2250      ;COUNT=2250?
                PUSH       HL           ;SAVE COUNT
                OR         A            ;CLEAR CARRY
                SBC        HL,DE        ;COMPARE HL&DE
                POP        HL           ;RESTORE COUNT
                JR         NZ,TASK1     ;COUNT<>2250
                CALL       BEEPER       ;MAKE A NOISE
                LD         HL,0         ;RESTART COUNT
TASK1           LD         (IX+2),L     ;PUT COUNTER LSB BACK
                LD         (IX+3),H     ;PUT MSB BACK
                RET
```

This is a trivial routine that will make a beeping noise (generated by the imaginary routine BEEPER) once every five minutes (2250 x 133.33 mS). Note how the interrupt task can take advantage of the IX register pointing to the task block. It is common practice to locate any variables used by the interrupt task in the task block to simplify the interrupt task's job.

DOSPLUS actually provides two interrupt chains: a fast chain and a slow chain. On the TRS-80, the <u>real-time clock</u>, or <u>RTC</u>, generates interrupts several times each second. The RTC on the Model I generates 40 interrupts per second (each 25 milliseconds) and the Model III RTC generates 30 interrupts per second (each 33.33 mS). The fast interrupt chain is entered each 25 mS on the Model I and each 33.33 mS on the Model III. There are 2 slots in the Model I's fast interrupt chain, numbered 4 & 5. Four slots are allocated to the Model III's fast interrupt chain, numbered 4-7.

The slow interrupt chain is only entered each 100 mS on the Model I, 133.33 mS on the Model III. Both the Model I and III have four slow interrupt chain slots, numbered 0-3.

The table below contains information regarding which interrupt slots are used by DOSPLUS to perform certain tasks. Those marked FREE are not used by the system, and may be used by user programs without sacrificing DOS features.

Interrupt chain slot allocation table - Model I & III DOSPLUS 3.5

Slot #	Model I		Model III	
0	FREE	100 mS	FREE	133 mS
1	@DO driver cursor blink	100 mS	FREE	133 mS
2	Clock display	100 mS	FREE	133 mS
3	Time update	100 mS	FREE	133 mS
4	FREE	25 mS	FREE	33 mS
5	@PR driver spooler	25 mS	FREE	33 mS
6	N/A		@PR driver spooler	33 mS
7	N/A		ROM int. tasks	33 mS

==
SET@
Address: 4410H (Model I) & 4413H (Model III)

SET@ is used to insert a pointer into DOSPLUS's interrupt chain. This pointer must point to an interrupt task block which in turn points to the actual interrupt processing task. Before entry to the interrupt task, all registers have been saved on the stack (NOTE: Under Model I DOSPLUS, the IY register is not saved on the stack), and the user program need only execute a RET instruction after the interrupt task is complete.

Note that the SET@ entry point is different for Model I and Model III DOSPLUS.

ENTRY: A=   Interrupt slot number
             DE=> Interrupt task block

EXIT: BC is altered
==

==
RESET@
Address: 4413H (Model I) & 4416H (Model III)

    The RESET@ routine is used to remove a task from the interrupt chain.

    Note that RESET@ has different entry points on Model I and III DOSPLUS

ENTRY: A=   Slot number containing task to be removed.

EXIT: BC   is altered
==

Miscellaneous System Routines

DOSPLUS 3.5 provides many useful routines that do not fall into any handy category, including the parameter parser and command evaluator, the wildcard evaluator, the sort routine, arithmetic routines, and others.

===
MULT@
Address: 444BH (Model I) & 444EH (Model III)

This routine will perform a 16-bit by 8-bit multiplication. Note that this routine is not located at the same address on both the Model I and Model III versions of DOSPLUS.

ENTRY: A=   Multiplier
            HL   Multiplicand

EXIT: A=   LSB
           L=   NSB
           H=   MSB
===

===
DIVD@
Address: 444EH (Model I) & 4451H (Model III)

DIVD@ will perform a 16-bit by 8-bit division. Note that DIVD@ is located at different addresses on the Model I and Model III.

ENTRY: A=   Divisor
           HL=   Dividend

EXIT: A=   Remainder
           HL=   Quotient
===

===
PARAM@
Address: 4476H (Model I) & 4454H (Model III)

   This routine is used to scan a parameter field for parameters and their values. For instance, the command line:

   DUMP TEST/CMD (START=7000H,END=71FFH,TRA=7120H)

contains three parameters in its parameter field, START, END, and TRA, whose values are 700011, 71FFH, and 7120H respectively. It is PARAM@'s job to scan such a line and assign values to each parameter.

   PARAM@ requires a <u>parameter block</u>. The parameter block consists of a series of entries, each of which contains a 6-byte parameter name, left justified and padded with blanks, and a 2-byte pointer. The pointer is used to indicate to PARAM@ where the value of the parameter is to be stored in RAM. The parameter block is terminated with a 00H. A typical parameter block is illustrated below:

```
        PRMBLK      DEFM        'START '    ;START PARAM
                    DEFW        SRTVAL      ;POINTER
                    DEFM        'END '      ;END PARAM
                    DEFW        ENDVAL      ;POINTER
                    DEFM        'TRA '      ;TRA PARAM
                    DEFW        TRAVAL      ;POINTER
                    DEFB        0           ;END OF BLOCK
```

   This is a parameter block that might be used for the DUMP example given above. In this case, the value of START (7000H) would be stored in the RAM location labeled SRTVAL, END (71FFH) would be placed in ENDVAL, and TRA (7120H) would be loaded into the address labeled TRAVAL.

   If PARAM@ encounters a word in the parameter field which does not occur in its parameter list, it will return with NZ status to indicate that a parameter error has been detected, and the character causing the error will appear in the A register. Note that PARAM@ does not return an error code.

   PARAM@ may accept either <u>numeric</u> or <u>logical</u> values. A numeric value is one such as "DATA=82". A logical value is one which has a "YES/ON" (TRUE) or "NO/OFF" (FALSE) value, such as "PARITY=NO". Note that if a parameter is given without a value, as in "DIR :3 (SYS)", the parameter, in this case SYS, is assigned a logical value of YES. A logical value of TRUE is represented by placing an FFFFH in the parameter's value, and a value of FALSE is signaled by a 0000H.

   PARAM@ will skip any leading spaces or commas preceding the parameter field.

Note that PARAM@ has different locations on the Model I and Model III versions of DOSPLUS. The Model I address may be used under Model III DOSPLUS, but the Model III address may not be used under Model I DOSPLUS.

ENTRY: DE=> Parameter block
                 HL=> Parameter field text

EXIT: BC   is altered
                 Flags: Z=No error, NZ=Error

==

==

DSPLY@
Address: 4467H (Model I & III)

This routine allows a user program to display an entire block of text on the video display (subject to FORCE and JOIN). Two terminating characters are available for this routine: ASCII 03H (ETX), which terminates the block display and leaves the cursor in its current position, and ASCII 0DH (CR), which is displayed, and then terminates the block display

ENTRY: HL=> Text to be displayed, terminated with 03H or 0DH.

==

==

PRINT@
Address: 446AH (Model I & III)

PRINT@ is used to print a block of text on the printer (@PR), subject to any FORCEing or JOINing currently in effect. The block of text to be printed may be terminated with either an ASCII 03H (ETX), or with an ASCII 0DH (CR) which causes the printer to begin a new line.

ENTRY: HL=> Text to be printed, terminated with a 03H or 0DH.

==

==

GTTIME@
Address: 446DH (Model I & III)

This routine will create an ASCII string representing the current system time in a user-specified area of RAM in the format HH:MM:SS.

ENTRY: HL=> 8-byte ASCII text buffer

==

==
GTDATE@
Address: 4470H (Model I & III)

This routine will create an ASCII string representation of the current system data in a user-specified area of RAM in the format MM/DD/YY.

ENTRY: HL>   8-byte ASCII text buffer
==

==
EVAL@
Address: 4479H (Model I & III)

This 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 signaled by a comma or a left parenthesis. 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:

```
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 and 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:          Reserved
        Bit 3:          Device field contains drivespec
        Bits 0-2:       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 DCII contains a file specification, and bit 5 is set if the filespec contains wildcard characters.

Bit 3 is set if the device specification flagged with bit 7 belongs to a disk drive device.

Bits 0-2 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% routines detailed elsewhere in this manual.

The parameter block used for EVAL@ is slightly different than that 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:

```
        PRMBLK+00H         6-byte parameter name
        PRMBLK+06H         2-byte pointer
        PRMBLK+08H         1-byte type specifier
```

The first eight bytes are identical to those used by PARAM@, 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 <u>must</u> point to a 00H byte.

ENTRY: HL=> Command line text, terminated with ASCII 0311, or 0DH.
                 IX=> Evaluation block

EXIT: A=   Error code
                 IX=> Evaluation flags
                 Flags: Z=No error, NZ=Error

Note: If an error occurs, the HL register will point to the character causing the error.
==

===
WILD@
Address: 447CH (Model I & III)

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 involves setting the wildmask to be used in any following comparisons, and the second function is the actual filespec/wildmask comparison. The filespec or wildmask provided to WILD@ must be properly terminated with an ASCII 03H (ETX) or 0DH (CR).

ENTRY: B=   Function switch
                     If B=0, then compare filespec with wildmask
                     If B=1, then set new wildmask
      HL=> Filespec to compare, or wildmask to set

EXIT: Flags: If B=0, Z=Filespec matches wildmask, NZ=Filespec does not match
                  If B=1, Z=Wildmask set, NZ=Invalid wildmask
===

===
LOCDEV@
Address: 447FH (Model I & III)

This routine will return a device number, given a device specification. Since all devices under DOSPLUS 3.5 may be renamed at will, this routine is very handy when it is necessary to determine what device number is referred to by any given device name. The device number may in turn be used to located the DCB or DCT belonging to the device by means of the LOCDCB% and LOCDCT% routines. The device number is returned in the A register, bits 0-2. Bit 3 of the device number is set if the device is a disk drive device, and reset if it is not.

ENTRY: DE=> 2-character device name prefixed with "@" or ":"

EXIT: A=   Bit 3:   Disk drive device flag
                     Bits 0-2:   Device number
               Flags: Z=Device located, NZ=Device not found
===

==
SORT@
Address: 4482H (Model I & III)

This routine 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. H=0: Ascending sort, H<>0: Descending sort
                    L=   Length of sort key
                    IX=> 1st byte of first entry in list

EXIT: A=   Error code
                    Flags: Z=No error, NZ=Error
==

Internal System Vectors

Three very useful routines fall into this category: The disk sector I/O system, and two routines used to locate system DCBs and DCTs.

==
DISKIO%

Address: 4485H (Model 1) & 4488H (Model III)

This routine is DOSPLUS 3.5's disk sector I/O system. With it, user programs can directly access any cylinder and sector on any drive. The DISKIO% routine has ten basic 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= Function code
C= Drive device number (0-7)
D= Cylinder number
E= Logical sector number
HL=> 256-byte disk I/O buffer

EXIT: A= Error code
Flags: Z=No error, NZ=Error

With all of the ten DISKIO% functions, the proper function code, taken from the table above, is loaded into the A register.

The C register should contain the disk drive device number (the number returned by the LOCDEV@ and EVAL@ routines, or, the position of the drive's DCTTBL entry).

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, the flag status will be Z if the drive is ready for I/O, and NZ if the drive is not ready. The C flag will also be set if 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.

SREAD performs another important function under DOSPLUS 3.5. When an SREAD is executed, the routine initializes the perishable portion of the drive's DCT if the log disk flag (DCT+03H, bit 0) is set. On floppy diskettes, this means reading the DCT with data stored in the driver itself, as well as reading the diskette's GAT to determine whether the diskette is single- or double-sided. With this information the surface count data in the DCT is updated.

On rigid drives, the SREAD DCT initialization is simpler. SREAD merely needs to read the perishable DCT information stored in sector 2 of the BOOT/SYS file located on cylinder 0, sector 2, and load that information into the DCT.

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 3.5 formats a single track.
===

===
LOCDCB%
Address: 4488H (Model I) & 44A0H (Model III)

This routine may be used to locate the device control block of any character-oriented device. LOCDCB% will return the address of any of the six character-oriented I/O devices, given the device number. This is most useful, since DOSPLUS 3.5 allows user programs to relocate DCBs to any region of RAM, and indeed, certain devices have no DCB address until such time as a driver is ASSIGNed to them. If LOCDCB% is used on a device that has no DCB currently defined, the DCB address returned will be FFFFH.

Please note that this routine resides in different locations on the Model I and Model III version of DOSPLUS.

ENTRY: A=   Device number (0-7)

EXIT: DE=> Device control block
===

==
LOCDCT%
Address: 448BH (Model I) & 44A3H (Model III)

    This routine is used to locate the drive control table for any of the eight disk drive devices supported under DOSPLUS 3.5. Since DOSPLUS allows user program to relocate existing DCTs and define new DCTs, this routine is of great usefulness.

ENTRY: A=   Disk drive device number

EXIT: IY=> Drive control table
==

Useful ROM Routines

In addition to the many DOSPLUS system entry points, the TRS-80's ROM provides several very handy routines, some of which are discussed below:

==
GET@
Address: 0013H (Model I & III)

This is part of the <u>character input/output</u>, or <u>CIO</u> system of the TRS-80's ROM. The GET@ routine is used to fetch a single byte, or character, from any system device or file. When fetching data from a device, any error status returned by GET@ should be ignored.

ENTRY: DE=> DCB or open FCB

EXIT: A=   Character from device or file
              Flags: Unknown if device (dependent on driver)
                     If file, Z=No error, NZ=Error, with error code in A register
==

==
PUT@
Address: 001BH (Model I & III)

This routine is the character output portion of the system's character I/O routine. With it, a single byte may be output to any character-oriented device or to a file. When outputting data to a device any error status returned from PUT@ should be ignored.

ENTRY: A=   Character to output to device/file
              DE=> DCB or open FCB

EXIT: Flags: Unknown if device (dependent on driver)
                     If file, Z=No error, NZ=Error, with error code in A register
==

==
KBD@
Address: 002BH (Model I & III)

This routine is used to fetch a single character from the keyboard driver. Typically, the driver will return status in the Z flag, although this is dependent on the particular keyboard driver in use. The standard ROM keyboard driver, and the driver furnished with DOSPLUS 3.5 will return NZ status if a character is found, and Z status if no character is present.

EXIT: A=   Key depressed
            DE   is altered
==

===
DSP@
Address: 0033H (Model I & III)

This routine will output a single character to the system's display driver.

ENTRY: A=   Character to display

EXIT: DE   is altered
===

===
PRT@
Address: 003BH (Model I & III)

PRT@ will output a single character to the printer driver.

ENTRY: A=   Character to print

EXIT: DE   is altered
===

===
KEYIN@
Address: 0040H (Model I & III)

The KEYIN@ routine is used to accept an entire line of input from the TRS-80's keyboard, terminating when <ENTER> or <BREAK> is pressed.

ENTRY: B=   Maximum input length
               HL=> Input buffer

EXIT: B=   # of characters entered, less terminator
               C=   Maximum input length
               DE   is altered
               Flags: Z if null entry
                      C if <BREAK> pressed
===

===
KEY@
Address: 0049H (Model I & III)

The INKEY@ routine will scan the keyboard for a keystroke, much as KBD@ does, but INKEY@ does not return to the calling program until a keystroke is received.

EXIT: A=   Key pressed
               DE   is altered
===

IX. - Important Memory Addresses

There are several data areas in DOSPLUS 3.5 that can be of use to those writing programs for use under DOSPLUS. Note that those addresses marked with an asterisk, "*", are subject to change in future versions of DOSPLUS.

==
BREAK$
Address: 4312H (Model I) & 42ADH (Model III)

This address contains the <u>break key vector</u>, or that location to which program control is transferred when the keyboard driver senses that the <BREAK> key has been depressed. It is a 3-byte area that normally contains a Z-80 RET instruction, but user programs may install a vector to any routine desired.
==

==
DATE$
Address: 4044H (Model I) & 421AH (Model III)

This 3-byte area contains the system date, stored in year-day-month order.
==

==
DODCB$ *
Address: 435AH (Model I) & 42A9H (Model III)

This 2-byte pointer contains the address of the 288-byte (32-byte FCB followed by a 256-byte I/O buffer) block of high RAM assigned by the system for use by DO files. Bit 6 in SFLAG1, below, is used to flag whether a DO block has been assigned.
==

==
HMEM$
Address: 4049H (Model I) & 4411H (Model III)

This 2-byte value contains a pointer to the highest free address in user RAM at any given time. This value is changed from time to time as various drivers, filters, and programs are loaded into high memory. It is the responsibility of every program running under DOSPLUS to respect the HMEM$ value; that is, no program must make use of the area of memory above that value contained in HMEM$.
==

==
INPBUF$
Address: 4318H (Model I) & 4225H (Model III)

This is the DOSPLUS command buffer area. All commands typed at the DOSPLUS command level are placed in this 64-character buffer.
==

===
OVLYH$ *
Address: 430BH (Model I) & 429EH (Model III)

    This byte contains a value which indicates which of DOSPLUS 3.5's high overlay group /SYS files is currently resident in RAM. The most significant 4 bits of this byte contain the system number less 1. User programs use this byte primarily to force the system to re-load a system file by placing an FFH in OVLYH$.
===

===
OVLYL$ *
Address: 430AH (Model I) & 429DH (Model III)

    This byte contains a value which indicates which of DOSPLUS 3.5's low overlay group /SYS files is currently resident in RAM. The most significant 4 bits of this byte contain the system number less 1. User programs use this byte primarily to force the system to re-load a system file by placing an FFH in OVLYL$.
===

===
SFLAG1$ *
Address: 4302H (Model I) & 4297 H (Model III)

    This byte contains eight flags that reflect the current status of the system:

Bit	Meaning
7	Reserved
6	DO block allocated
5	Active DO flag
4	Reserved
3	Verify flag
2	Debug flag
1	Run access flag
0	Break key disable flag

    Bit 6 is used by the system to note whether a 288-byte block of RAM has been assigned in high RAM for use by the DO command.

    Bit 5 is set whenever an DO file is currently active, informing the keyboard driver to accept data from the DO file.

    Bit 3 is set to instruct the operating system to perform read-after-write verification of all data written to disk.

    Bit 2, when set, activates the DEBUG monitor.

    Bit 1 is set by the operating system during execution of the RUN@ system routine, in order to prevent "File access denied due to password protection" errors when attempting to execute file with run-only protection status.

    Bit 0, when set, disables the <BREAK> key such that no keycode is returned when the <BREAK> key is depressed.
===

==
TIME$
Address: 4041H (Model I) & 4217H (Model III)

    This 3-byte area contains the system time stored in seconds-minutes-hours order.
==

==
OSVER$
Address: 403EH (Model I only)

    This address holds a 1-byte DOS version number, which is stored in BCD format. The most significant 4 bits contain the DOS version number, and the least significant 4 bits contain the release number.
==

X. - Writing Drivers for DOSPLUS 3.5

DOSPLUS 3.5 allows the user to write <u>device drivers</u>, or programs that interface with he 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 3.5.

Writing device drivers for DOSPLUS 3.5 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 six character-oriented devices (@KI, @DO, @PR, @RS, @U1, @U2) or one of the eight disk drive devices (:0 through :7), 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: BC=> Lowest available address in user RAM
     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 HMEM$ 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 high RAM for the device. This address will be automatically inserted into the DCBTBL or DCTTBL before control is passed to the driver program, and the address of the newly created DCB/DCT will be present in the IX register.

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 3.5 disk drivers must support the ten functions explained in section VIII under the system routine 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: A=   Function code (0-9)
              C=   Device number (0-7)
              D=   Logical cylinder number
              E=   Logical sector number
              HL=> 256-byte disk I/O buffer
              IY=> Device DCT

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 six 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 1-80 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
            Flags: Z=Write operation, NZ=Read operation

Drivers may use any registers required, as the CIO system saves all registers (except IY) 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 the Z flag if no character was available, and set NZ 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.

Character I/O drivers are responsible for performing <u>filtering</u>, or translation of character values. The DCBTBL entry for each character-oriented device contains a pointer to a filter table, and the DCB itself contains a bit in the DCB type byte which indicates to the driver whether filtration should be performed. When this bit (bit 6 of DCB+00H) is set, the driver should perform character translation on all bytes passing to or from the driver.

The device filter table, pointed in the DCBTBL by bytes ENTRY+02H and ENTRY+03H, is of variable length. The first byte of the table contains the number of entries in the table. A value of 0 indicates a 256-entry filter table. Following the length byte is a series of filter table entries, each two bytes in length. The first byte of the entry is the value of the character to be filtered, and the second byte is the value of the character into which it will be translated. Any characters not in the filter table should be passed by the driver unchanged.

A sample RS232 serial driver is included in this manual to provide an example of character-I/O device driver structure and execution.

XI. - DOSPLUS 3.5 Error Codes

The following is a list of the error codes generated by DOSPLUS 3.5 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 locked/deleted data record
7	07	Attempted to read system data record
8	08	Drive 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	Directory space full
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
31	1F	Program not found
32	20	Improper drive number
33	21	No memory 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	I/O attempted to unopen file
39	27	Device in use
40	28	Protected system device
41	29	Device not available
42	2A	No device space available
43	2B	Illegal devicespec
44	2C	Illegal filespec
45	2D	Invalid data provided
46	2E	Invalid parameter
47	2F	I/O field not found
48	30	Terminated
49	31	File already exists
50	32	Device already exists

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 3.5.

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 3.5, 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 3.5 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 Block. 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 3.5, 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 3.5 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 3.5, 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 3.5 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 hard ware 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 3.5 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 granules 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 <u>system</u> attribute set as part of the file protection status. Normally, users may not create files with the system attribute.

Track

A single circular magnetic recording area on a diskette. Diskettes generally contain many circular tracks.

Trapdoor Code

Under DOSPLUS 3.5, 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.

<u>Disk Drive Device Driver Example</u>

```
;          THIS DRIVER IS DESIGNED FOR THE WD1000 HARD
;          DRIVE CONTROLLER.
;
;          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 -      $DWRTEA     - WRITE SECTOR W/AM
;          8 -      $DFORMT     - FORMAT TRACK W/SEEK
;
;          LOCAL EQUIVALENCES
;
DREAD#     EQU      3               ;READ SECTOR
DVERF#     EQU      4               ;WRITE SECTOR
ETX        EQU      3               ;ASCII ETX
LF         EQU      10              ;ASCII L/F
CR         EQU      13              ;ASCII C/R
;
;          HARDWARE ADDRESSING
;
BASE       EQU      10H             ;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 =>        LOW$
;                    DE =>        HIGH$
;                    HL =>        COMMAND LINE
;                    IX =>        DCT
;                    IY =>        DCT DESCRIPTOR
;
           ORG       5200H
;
START      PUSH      DE           ;SAVE HIGH
           LD        HL,TITLE     ;'DRIVER TITLE'
           CALL      DSPLY@       ;LEN+TERM
           POP       HL           ;HL > HIGH$
;
;          FETCH MEMORY FROM DOS
;
           LD        BC,ENDDVR-BEGDVR
           OR        A            ;CLR CRY
           SBC       HL,BC        ;HIGH$-LEN DRIVER
           LD        (HMEM$),HL   ;NEW TOP MEM
           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         ;OFFSET
           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
                SET     0,(IX+3)    ;LOG IT!
                RES     3,(IX+0)    ;DEVICE ACTIVE
        ;
                LD      HL,MES1     ;'DRIVER INSTALLED AT'
                CALL    DSPLY@
                LD      L,(IX+1)
                LD      H,(IX+2)    ;HL => DRIVER
                CALL    BINHEX      ;BIN TO HEX CONV
                LD      A,CR        ;C/R
                CALL    DSP@        ;OUTPUT
                RET
        ;
        ;       BIN TO ASCII HEX OUTPUT
        ;
        ;       ENT     HL =        BINARY WORD
        ;
        BINHEX  LD      A,H         ;MSB
                CALL    OHEX1       ;OUTPUT
                LD      A,L         ;LSB
        OHEX0   CALL    OHEX1       ;OUTPUT
                LD      A,'H'       ;HEX ADDRESS
                JP      DSP@        ;OUTPUT
        ;
        OHEX1   PUSH    AF          ;SAVE BYTE
                RRA
                RRA
                RRA
                RRA                 ;MSB TO LSB
                CALL    OHEX2       ;OUTPUT DIGIT
                POP     AF          ;GET BYTE
        OHEX2   AND     0FH         ;LSB
                ADD     A,90H
                DAA
                ADC     A,40H       ;CONVERT TRICK
                DAA
                JP      DSP@        ;OUTPUT
```

```
        ;
        ;          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 - 3.50'
         DEFB     LF
         DEFM     '(c) Copyright 1983, Micro-Systems Software Inc.'
         DEFB     LF
         DEFB     CR
        ;
MES1     DEFM     'Driver installed at '
         DEFB     ETX
        ;
        ;          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 SECTOR
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+3    ;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        B =         LSN HIGH
;                      E =         LSN LOW
;                      D =         CYLINDER
;                      HL =>       I/O BUFFER
;
DREAD       CALL       DIOP
RL7         EQU        $-2
            DEFB       HREAD       ;HDC READ
            DEFB       5           ;RETRY COUNT
            DEFB       50          ;ERROR OFFSET
            DEFB       1           ;I/O TYPE
;
DVERF       CALL       DIOP
RL8         EQU        $-2
            DEFB       HREAD       ;HDC READ
            DEFB       2           ;RETRY COUNT
            DEFB       50          ;ERROR OFFSET
            DEFB       1           ;I/O TYPE
;
;           WRITE SECTOR ROUTINES
;
;           ENT        B =         LSN HIGH
;                      E =         LSN LOW
;                      D =         CYLINDER
;                      HL =>       I/O BUFFER
;
DWRITE      CALL       DIOP
RL9         EQU        $-2
            DEFB       HWRTE       ;HDC WRITE
            DEFB       5           ;RETRY COUNT
            DEFB       50          ;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       50          ;ERROR OFFSET
            DEFB       2           ;I/O TYPE
```

```
;
;         FORMAT TRACK ROUTINE
;         ENT       B =       LSN HIGH
;                   E =       LSN LOW
;                   D =       CYLINDER
;                   HL =>     I/O BUFFER
;
DFORM   CALL      DIOP
RL11    EQU       $-2
        DEFB      HFRMT       ;HDC FORMAT
        DEFB      1           ;RETRY COUNT
        DEFB      108         ;ERROR OFFSET
        DEFB      4           ;I/O TYPE
;
;         DISK I/O OPERATION
;
;         ENT       B =       LSN HIGH
;                   E =       LSN LOW
;                   D =       CYLINDER
;                   HL =>     I/O BUFFER
;                   IY =>     DRIVE CONTROL TABLE
;                   SP =>     FDC FUNCTION
;                   SP+1 => RETRY COUNT
;                   SP+2 => ERROR CODE OFFSET
;                   SP+3 => XFER OPCODE
;
DIOP    POP       IX          ;IX => INFO
        BIT       0,(IX+3)    ;INPUT?
        JR        NZ,$+9      ;IF YES
        LD        A,(IY+3)    ;GET FLAGS
        AND       40H         ;WP?
        JR        NZ,DIOERR   ;ERROR!
;
;         WINCHESTER I/O ROUTINE
;
IOP1    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      B =        LSN HIGH
;                   E =        LSN LOW
;                   D =        CYLINDER NUMBER
;                   IY =>      DCT
;
UPTASK   PUSH     DE           ;SAVE
         LD       A,(IY+6)     ;CYL OFFSET
         ADD      A,D          ;CYLINDER
         OUT      (HD#L),A     ;SET CYL LOW
         LD       A,(IY+7)     ;CYL OFFSET
         ADC      A,0          ;MSB
         OUT      (HD#H),A     ;SET CYL HIGH
         LD       A,(IY+13)    ;SEC/TRACK
         CALL     SDIVD%       ;GET SRFCE,SEC
         LD       D,A          ;D = HEAD
         LD       A,(IY+8)     ;SECTOR OFFSET
         ADD      A,E
         OUT      (HD#S),A     ;SET SECTOR
         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           ;RESTORE
         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

```
;
;         MODEL I/III RS232 DRIVER
;
ENT#      EQU      1
;
;         CHARACTER DEFINITIONS
;
ETX       EQU      3
LF        EQU      10
CR        EQU      13
;
;         HARDWARE ADDRESSES
;
RSTAT     EQU      0EAH           ;UART STAT PORT
RDATA     EQU      0EBH           ;RS232 DATA PORT
;
;         INITIALIZATION CODE SEQUENCE
;
;         ENT      BC =>          LOW$
;                  DE =>          HIGH$
;                  HL =>          COMMAND LINE
;                  IX =>          DCB
;                  IY =>          DCB DESCRIPTOR
;
          ORG      5200H
;
START     PUSH     DE             ;SAVE HIGH
          LD       HL,TITLE       ;'DRIVER TITLE'
          CALL     DSPLY@         ;LEN+TERM
;
;         FETCH MEMORY FROM DOS
;
          POP      HL             ;GET HIGH$
          LD       BC,ENDDVR-BEGDVR
          OR       A              ;CLR CRY
          SBC      HL,BC          ;HIGH$-LEN DRIVER
          LD       (HMEM$),HL     ;NEW TOP MEM
          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
;
          LD       (MOD00),IY     ;PUT DCBTBL ENTRY IN PRG
```

```
        ;
        ;       ADJUST DRIVER ADDRESSES
        ;
                LD        IY,TABLE      ;IY => ADD TABLE
                LD        A,ENT#        ;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          ;OFFSET
                ADD       IY,DE         ;IY => NEXT ADD
                DEC       A             ;DONE?
                JR        NZ,ADJ1       ;IF NOT
                POP       DE            ;DE => START FREE
        ;
        ;       INSTALL DRIVER HERE
        ;
                LD        A,40H         ;GET FILTER STAT
                AND       (IX+0)        ;FROM OLD DCB
                OR        3             ;SET INPUT/OUTPUT TYPE
                LD        (IX+0),A      ;AND REPLACE IN DCB
                LD        (IX+1),E      ;ADD TO DCB
                LD        (IX+2),D      ;ADD TO DCB
                LD        HL,BEGDVR     ;HL => DRIVER
                LD        BC,ENDDVR-BEGDVR
                LDIR                    ;MOVE!
        ;
                LD        HL,MES1       ;'DRIVER INSTALLED AT'
                CALL      DSPLY@
                LD        L,(IX+1)
                LD        H,(IX+2)      ;HL => DRIVER
                CALL      BINHEX        ;BIN TO HEX CONV
                LD        A,CR          ;C/R
                CALL      DSP@          ;OUTPUT
        ;
                RET
```

```
;
;       BIN TO ASCII HEX OUTPUT
;
;       ENT     HL =        BINARY WORD
;
BINHEX  LD      A,H         ;MSB
        CALL    OHEX1       ;OUTPUT
        LD      A,L         ;LSB
OHEX0   CALL    OHEX1       ;OUTPUT
        LD      A,'H'       ;HEX ADDRESS
        JP      DSP@        ;OUTPUT
;
OHEX1   PUSH    AF          ;SAVE BYTE
        RRA
        RRA
        RRA
        RRA                 ;MSB TO LSB
        CALL    OHEX2       ;OUTPUT DIGIT
        POP     AF          ;GET BYTE
OHEX2   AND     0FH         ;LSB
        ADD     A,90H
        DAA
        ADC     A,40H       ;CONVERT TRICK
        DAA
        JP      DSP@        ;OUTPUT
;
;       TABLE OF NON-RELOCATABLE ADDRESSES
;
TABLE   DEFW    REF00
;
;       MESSAGES AND TEXT STRINGS
;
TITLE   DEFM    'RS/DVR - DOSPLUS RS232 driver - 3.50'
        DEFB    LF
        DEFM    '(c) Copyright 1983, Micro-Systems Software Inc.'
        DEFB    LF
        DEFB    CR
;
MES1    DEFM    'Driver installed at '
        DEFB    ETX
;
;       RS232 DRIVER
;
;       ENT:    C=CHR TO OUTPUT
;               IX=>DCB
;
;       DETERMINE INPUT OR OUTPUT
;
BEGDVR  EQU     $
        PUSH    IY          ;SAVE REG
MOD00   EQU     $+2
        LD      IY,$-$      ;GET DCB TABLE ENTRY
        LD      L,(IY+2)    ;GET FILTER ADDR
        LD      H,(IY+3)
        POP     IY          ;RESTORE
        BIT     1,B         ;OUTPUT OPERATION?
        JR      NZ,RSOUT    ;YES
```

```
        ;
        ;       INPUT BYTE FROM RS232
        ;
RSIN0   IN      A,(RSTAT)   ;RX REG FULL?
        AND     80H         ;WELL?
        RET     Z           ;IF, EFTY
        IN      A,(RDATA)   ;GET DATA BYTE
        JR      RFLT        ;FILTER CHR AND RETURN
        ;
        ;       OUTPUT BYTE TO RS232
        ;
RSOUT   IN      A,(RSTAT)   ;TX REG EVTY?
        AND     40H         ;WELL?
        JR      Z,RSOUT     ;WAIT FOR EMPTY
        LD      A,C         ;GET CHR
REF00   EQU     $+1
        CALL    RFLT        ;FILTER CHR
        OUT     (RDATA),A   ;XMIT CHR
        RET
        ;
        ;       FILTER CHR
        ;
RFLT    BIT     6,(IX+0)    ;FILTER?
        RET     Z           ;NO
        LD      B,(HL)      ;GET TABLE LENGTH
        INC     HL          ;HL=>TABLE ENTRIES
RFLT0   CP      (HL)        ;MATCH?
        INC     HL          ;HL=>XLATED VALUE
        JR      NZ,RFLT1    ;NO MATCH
        LD      A,(HL)      ;GET XLATED CHR
RFLT1   INC     HL          ;HL=>NEXT ENTRY
        JR      Z,RFLT2     ;IF MATCH
        DJNZ    RFLT0       ;TILL DONE
RFLT2   OR      A           ;NZ STAT
        RET
        ;
ENDDVR  EQU     $
        END     START
```

NOTES