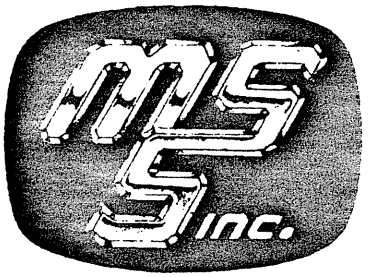
MICRO-SYSTEMS SOFTWARE INC.

5846 Funston Street Hollywood, FL 33023
(305) 983-3390



User's Manual Ver A.2
for
DOSPLUS Ver 3.4/4.0
Disk Operating System

MICRO-SYSTEMS SOFTWARE INC.

5846 Funston Street Hollywood, FL 33023
(305) 983-3390

To : All Dosplus purchasers
(namely the owner of this manual)

Re : Future things.

==========

Dear owner:

Welcome to Dosplus. I think you're gonna like our new system. We are really proud of it. I think that Dosplus 3.4/4.0 is just what the industry ordered.

Remember Dosplus 3.3? Well, it had a $100 bug reward. I paid off five times (three on the Model I, two on the Model III). None of them truly major. Mostly typos or cosmetic errors. However, you would not BELIEVE the volume of letters I had to read from people that didn't have a bug at all. It is for this reason only, that we are discontinuing the reward. If you feel that you have found a bug, here's what the policy is.

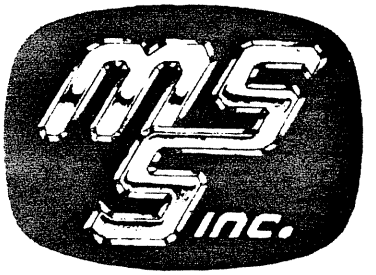
All software is sold on and "as-is" basis. Micro-Systems Software Inc. and its subsidiaries make no promises regarding operation of its programs other than those implicitly stated in this letter. If a bug is found that requires a "zap" to fix, the "zap" will be published in the bi-monthly newsletter and credit given to the person who located it. Please remember, there is a difference between a bug and a shortcoming. Nothing is perfect. There are some things we may not do the way you would like us to. These are not bugs and will probably not be altered until the next system.

Next point. You saw that I mentioned a newsletter. That's right, we are finally going to do a newsletter. How do you get one? Simply fill out the registration card (if you are previously registered, you do not have to do it again) and return it to me. Every two months, you will be getting a letter from me with applications and other good stuff.

Some of the inclusions will be :

* Bug fixes (if any)
* Announcements of new products
* Applications from users
* Manual inserts
* Public domain programs
* Announcements of documentation or DOS upgrades

MICRO SYSTEMS SOFTWARE INC.
5846 Funston Street Hollywood, FL 33023
(305) 983-3390



I'm looking for YOUR help in this. This IS your letter. Send me applications information, information on programs that you have purchased to use with Dosplus and like. This is your chance to be published to the thousands of registered Dosplus owners. See your name in print. Send me reviews, sample programs, ANYTHING!

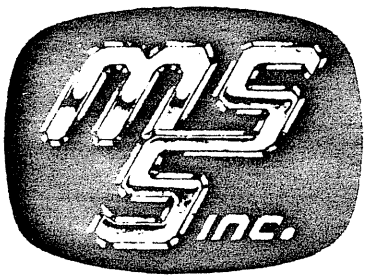
Policy :

* Send in a printed copy.
* Send a disk containing that file for my transfer. (Must be compatible with Lazy Writer, NewScript, or Scripsit).
* Send me written authorization to publish what you have written.

I will return all diskettes to you after copying the file off.

I can't tell you how many good ideas and applications you folks give me over the phone. Dosplus user's seem to be the most inventive people on earth! Write them down. We'll try and get them all in. The final judgement is mine. I will decide what goes in and what doesn't. I will try to limit it to things that are broadly useful, but every so often I may include specific applications.

MICRO-SYSTEMS SOFTWARE INC.
5846 Funston Street Hollywood, FL 33023
(305) 983-3390



Upgrade policy :

Micro-Systems Software Inc. does not believe in releasing "updates" every other week. It takes time to develop and test a new system. Your upgrades will come at best about once a year. We will upgrade you at slighty over cost. Documentation upgrades are another matter. We will be altering the manual all year long, as errors are found and suggestions incorporated. These will be sent to you via the newsletter. However, after enough red pencil marks, we will probably run a new manual. I will announce it in the newsletter and you may order the documentation upgrade for what it will cost me to print and mail such a package (probably about $10).

I hope you've enjoyed looking through our new manual (see, we really do listen to you). Be a part of the next one. Keep the cards and letters coming! You folks are great!

Sincerely,

Mark R. Lautenschlager G.M.

Mark R. Lautenschlager
General Manager
Micro-Systems Software Inc.

MRL/wp

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

TABLE OF CONTENTS (cont.)

Congratulations on your purchase of DOSPLUS! We at Micro-Systems Software feel that DOSPLUS is THE fastest, most reliable and easy-to-use operating system available.

DOSPLUS has been designed for ease of operation, and this manual is designed to help you become familiar with the many functions, commands and utilities provided with the System. If you have been using another operating system, we think that you'll be pleasantly suprised with DOSPLUS. There are, however, some differences between DOSPLUS and other DOS's, and therefore, we strongly recommend that you read this manual before attempting to use the system.

DOSPLUS is provided with two versions of Disk Basic - The Extended Z80 Disk Basic, and TBASIC, a memory-stingy BASIC. Both of these BASIC's operate with the speed and efficiency of DOSPLUS, but once again, there are some differences in this BASIC and other TRS-80 BASIC's, so please read the BASIC section of this manual in order to use the system to its greatest potential.

In this manual, we will make use of numerous examples of command entry, in order to fully illustrate the correct format for each type of command.

If you DO damage your DOSPLUS Master diskette, and have no backup copy, Micro-Systems Software will replace it for you. To obtain a replacement copy, mail your ORIGINAL diskette (The one with the Micro-Systems label on it) along with a short note explaining the problem to :

Micro-Systems Software
5846 Funston Street
Hollywood, FL 33023

or

Call (305) 983-3390 for instructions.

Be sure to include your name and address.

STARTUP -

In this section we will cover :

1. Booting up the DOS.
2. Duplicating the DOSPLUS diskette (BACKUP).
3. Making a data diskette (FORMAT).

BOOTING UP -

In order to use DOSPLUS, turn on the computer and insert the diskette in drive zero. Press the reset button. The red light (LED) on the drive should come on, and the

drive should run. Then one of the the following messages will appear on the screen (depending on which system you have) :

```
DOSPLUS - MODEL III OPERATING SYSTEM - VER 3.4  xxK
                    or
DOSPLUS - HARD DISK OPERATING SYSTEM - VER 4.0  xxK
                    or
DOSPLUS - MODEL I OPERATING SYSTEM - VER 3.4  xxK
                    and
COPYRIGHT (C) 1981, MICRO-SYSTEMS SOFTWARE INC.
```

The "xxK" is the memory size of your machine. For example a machine with 32 kilobytes of RAM will display "32K" at that location. If this does not match the amount of RAM in the machine, an error has ocurred.

After the message, this prompt will appear :

```
DOS PLUS
#
```

Whenever you see this prompt, you are at what is called the "DOS command mode", and can enter a DOSPLUS command or the name of a machine language program to be executed. For a more detailed explanation of what a "DOS command" is, see the library command "LIB". Remember the term "DOS command mode" and what it stands for, because this manual will be referring to it again.

After you reach this point correctly, it will be time to BACKUP the Dosplus system diskette.

BACKUP -

The FIRST thing that you should do (After reading this manual) upon receiving your DOSPLUS diskette is to make a backup copy. If, for any reason, you damage your DOSPLUS master diskette, the backup copy will still be intact. If you do not have a backup copy, a new copy may be provided for you if you return your original copy to Micro-Systems Software, but due to the amount of turn-around time in the mail, that is a time-consuming process (repeated replacement of disks will result in a service charge, however).

To make a backup copy, follow these instructions :

1. Boot up the Computer.
2. From the DOS command mode type BACKUP.

At this point, the Computer's screen should go blank for a moment, and then it should display the program header and then prompt you :

Source drive number ?

You should type a 0 (Zero). DO NOT type an alphabetic 'O', but rather a numeric '0'. Press <Enter>.

****
If you are a SINGLE drive user ONLY -
(multi drive user's refer to the section labeled "Multi drive user's only" further on in the text)
****

3. Now the Computer will display :

Destination drive number ?

4. Answer the 'Destination drive' question with a '0' (Zero), and press <ENTER>.

5. The Computer will display :

Backup date (MM/DD/YYY) ?

Enter the date, using two digits for the month, two digits for the day, and two digits for the year, and press <ENTER>. For example :

October 3, 1980 would be entered as	10/03/80
July 2, 1776	07/02/76
January 1, 2001	01/01/01

6. The screen will flash :

Insert SOURCE Disk (ENTER)

When this happens, press the <ENTER> key.

7. The display now flashes :

Insert DESTINATION Disk (ENTER)

At this point, load the blank or previously formatted diskette into drive #0 (After taking the DOSPLUS diskette out.). Press <ENTER>.

If the diskette is blank, or has been erased, DOSPLUS will 'FORMAT' it at this time. If the Diskette was previously formatted, the Computer will ask you :

Diskette contains data. Use it?

If you DO NOT wish to use this diskette, enter a 'N' (for NO). If you DO wish to use the

diskette, enter a 'Y' (for YES) or a 'U' (for USE). If you wish to re-format the destination disk first, enter a 'F' (for FORMAT).
Press <ENTER>.

8. You will be asked to swap the two diskettes back and forth in this manner several times in order to backup your DOSPLUS with a single drive. Eventually, the display will flash :

Insert SYSTEM disk (ENTER)

You may now press the <ENTER> key. Your DOSPLUS backup copy is finished. File your Master copy away in a safe place.

****
Multi-Drive users Only -
****

3. Insert a blank (or previously formatted) disk into drive #1, and close the door. Answer the 'Destination drive' question with a '1' and press <ENTER>.

4. The screen will now show :

Backup date (MM/DD/YY) ?

You should enter today's date using the MM/DD/YY format, that is, use TWO digits for the month, followed by a slash ("/") and two digits for the day, another slash, and two digits for the year followed by <ENTER>.

Example -

July 4, 1981 would be entered as :	07/04/81
December 25, 1980	12/25/80
January 1, 1982	01/01/82

5. If the disk in drive #1 has been previously formatted, the Computer will ask you :

Diskette contains Data. Use or not ?

If you do not wish to use that 'Destination' disk, type 'N' (for NO) and press <ENTER>. This will halt the BACKUP from being made.

If you DO want to continue, type a 'Y' (for YES) or 'U' (for USE) and the Computer will proceed to make the backup copy automatically.
If you wish to re-format the destination disk

first, type a 'F' (for FORMAT) and press <ENTER>.

In a few moments, when the backup is complete, the screen will flash :

Insert System Disk <Enter>

You should now press the <ENTER> key. Your backup copy is now in drive #1. Use it for your working copy of DOSPLUS, and put the Master copy in a safe place.

FORMAT -

In order to create a data diskette that contains no operating system, but has the information needed to receive data (multi drive systems only!), you must follow this procedure :

1. Boot up the computer.
2. From the DOS command mode type "FORMAT".

The screen will clear, the program header will be displayed and you will see the prompt :

Which drive is to be used ?

Answer this question with the drive number that contains the disk to be formatted. Be certain that if you are going to use drive zero, you should remove the Dosplus system disk and insert the disk to be formatted at this time.

Format date (MM/DD/YY) ?

Answer this in the same manner as you did when making a BACKUP. (see Making a BACKUP)

Password ?

Answer this with the password that you wish to assign to that diskette. This will be the "Diskette master password" that is used with the PROT command (see library commands). If no special protection is needed, we recommend a null password. This is accomplished by answering this question with pressing <ENTER>. This is the way you received your Dosplus diskette. Whatever you set your password to, don't forget it.

Number of cylinders (35-96) ?

Answer this question with the number of cylinders you wish to be formatted. This is limited by your hardware. Consult the owner's manual with your disk drives if you are uncertain. The

standard Radio Shack drives are 40 track.

Single or double density ?

Unless you are going to take this disk back to a Model I, answer this question with 'D' (for double). If you wish to create a single density diskette, type 'S' (for single). For further details see the FORMAT section in the utilities portion of the manual.

If the diskette contains data, you will see the message :

Diskette contains data, Use or not ?

Answer this question with a 'N' (for no) if you do not wish to use it. Answer it with a 'Y' (for yes), or a 'U' (for use) if you do. If you tell the machine to use the disk, it will erase all previous data. Please be careful.

After a few moments, the screen will flash :

Insert SYSTEM disk <ENTER>

At this point make sure that the Dosplus system diskette is in drive zero, and then press ENTER. The format is complete.

FILE AND DRIVE SPECIFICATIONS -

The only way to store and retrieve organized data under the Dosplus system is to put it into a FILE. A file can store the data on the disk until you are ready to retrieve it. The data is then accessed via the filename that you gave it when you created or last renamed the file. In this sense, the disk is really nothing more than a large electronic file cabinet.

A file specification (FILESPEC for short) will be in the following general format :

filename/ext.password:d

"filename" consists of an alphabetic character (A-Z) followed by up to seven optional characters or numbers. The DOS will store all filenames in upper case letters only.

"/ext" is the optional filename extension consisting of up to three letters or numbers. Like a filename, it must begin with a letter.

".password" is the optional file password consisting of up to eight letters or numbers; again starting with a letter.

":d" is an optional drive specification noting which drive this particular file is located upon. (0, 1, 2, or 3)

Note that there can be NO blank spaces inside of a filespec. Dosplus will terminate the filespec at the first blank space.

For example, the filespec :

PAYROLL/DAT.PAYDAY:1

references the file PAYROLL/DAT with the password PAYDAY located on drive one.

The name, extension, and drive all contribute to the file's uniqueness. The password does not. It merely controls access into the file.

FURTHER DETAIL AND EXAMPLES REGARDING FILES -

FILESPECS -

Throughout this manual, we will be dealing with data

and program FILES. A file is a group of data, which may represent a customer file, a price list, a BASIC program, a Z-80 object code program, or ANY other type of meaningful data. In most cases, DOSPLUS will never "know" what type of data is contained in any given file.

All files have a name, or file specification (FILESPEC for short). A filespec consists of up to four parts. For instance, given the filespec :

PRICE/DAT.DOLLAR:1

This filespec has all four parts. The first part is the name 'PRICE'. This name can be up to eight characters in length, and may contain any alphabetic or numeric character (A-Z, 0-9). It MUST, however, begin with an alphabetic character. Some legal and illegal names are listed below.

Legal	Illegal	
MONEY	MONEY$	'$' is an illegal character.
JUNSALES	JUNESALES	Too many characters.
YEAR	12MONTHS	Doesn't begin with an alphabetic character.

The second part of a filespec is the EXTENSION. In our example, PRICE/DAT.DOLLAR:1, The extension is DAT. The extension is separated from the name by a slash ("/"). An extension may use alphabetic and numeric characters (A-Z, 0-9) may be up to three characters in length, and must begin with an alphabetic character. The extension is useful when trying to invent a descriptive filespec. Some common extensions are listed below.

Extension	Use
BAS	BASIC language program.
SOR	Assembly language source code.
TXT	Text file.
DAT	Data file.
CMD	Executable Z-80 object code. Sometimes called 'Command' file.
BLD	'Build' file. A file created by the DOSPLUS 'BUILD' command, used to allow automatic key-in of commands.

The extension is not a required part of the filespec, but is often used to more completely describe a file's contents. For example, we may have a number of different files sharing 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

In our example, PRICE/DAT.DOLLAR:1, the third part of the filespec is DOLLAR, and it is called the PASSWORD. A password can be given to any file in order to restrict access to it. You may only use a password-protected file IF you have the password. Otherwise, DOSPLUS will not allow you to access it. (for a further explanation of passwords, see the library command ATTRIB)

A password may be up to eight (8) characters in length, and must begin with an alphabetic symbol. It is separated from the filename or extension by a period (.). The password is an optional portion of the filespec, and may be omitted.

Once you have created a file with a password, BE SURE to remember the password. If you forget it, you will NOT be able to access that file again, except through the use of the PROT command detailed elsewhere in this manual.

DRIVE SPECIFICATION-

The final element of the filespec is the DRIVE SPECIFICATION. The example PRICE/DAT.DOLLAR:1 contains a drive specification of 1. This drive spec simply informs DOSPLUS that the file PRICE/DAT is located on disk drive # 1.

A drive spec consists of a number (0,1,2, or 3).

The drive spec is an optional element of the filespec. If you do not give a drive spec, DOSPLUS will automatically search all of the drives in your system, starting with drive # 0 up, until it finds a filespec matching the one you have given.

What makes a FILESPEC Unique -

It is important that we know what parts of the filespec distinguish it from other filespecs. If, for instance, we have written a BASIC program and we wish to store it on the disk, we must give it a filespec, or name.

It is important to us that the filespec we assign to the program does not conflict or duplicate another filespec already on the disk, because if it did, the Computer could not tell the difference between them, and the existing file would be destroyed and replaced with new data.

Three of the four parts of the filespec determine uniqueness : The Name, Extension, and the Drive Spec. The password does NOT. What this meaans is that if two filespecs have the same name and drive spec, but a different extension, they are two distinct files. If, however, two files have the same name, extension, and drive spec, but different passwords THEY DENOTE THE SAME FILE. For instance :

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

Please do not read the following section until you have completed the sections GETTING STARTED and FILESPECS.

GENERAL SYNTAX -

When using the Dosplus system, there are several "rules of thumb" that will help you with overall operation. For example, you have several different types of DOS commands. There are :

No file commands (such as DIR, FREE, CAT, etc.)

command (options)

The options are a list of one or more parameters that may be needed by the command. Some commands use no options. The parenthesis around the options are always included unless the options themselves are omitted.

One file commands (such as LIST, ATTRIB, etc.)

command filename (options)

The filename is a standard Dosplus file specification. The options are the same as above.

Two file commands (such as COPY, APPEND, etc.)

command filename delimiter filename (options)

The filename is a standard Dosplus file specification. The delimiter is one of the following :

- a blank space
- the word TO surrounded by blank spaces
- sometimes a comma

The options are the same as above.

WHEN A COMMAND IS ENTERED -

When you issue a command to the Dosplus system, it will proceed in the following steps :

1. First it checks to see if what you typed in is a Dosplus command. If it is Dosplus executes it immediately...

2. If it is not, the it will assume it to be an executable program (as in filename/CMD), and

check the drives for it.

3. When searching for a file, Dosplus starts with drive zero, then one, etc., etc...

4. Unless you specify a drive, in which case the DOS will search only that drive for the program.

5. When Dosplus finds the correct file, it will attempt to load and execute it. If this is not feasible for some reason, you will get an error message.

DEFINITIONS AND OPERATING TERMS -

SYSTEM OR DATA DISK -

Throughout the manual, we will refer to two distinct types of diskettes. Distinct in the TYPE of data that they hold rather than in the manner that they hold it.

The "system" disk is so called because it contains the Disk Operating System or DOS. The Dosplus diskette that we sent you was such a disk. The system disk contains the program that actually makes the computer run.

Without this, the computer would be remarkably like a child before any formal education. Capable of speech and rudimentary comprehension (due to the ROM), but not able to perform advanced functions without aid.

The drive zero diskette must always be a system disk!

A "data" diskette is one that holds the programs and files that you have created. You make a data disk by using the FORMAT utility (see GETTING STARTED and FORMAT).

Data (or system) diskettes can be used in drives 1, 2, or 3.

MASTER PASSWORD -

At the time of FORMAT or BACKUP, each diskette is assigned a "Master Password". The master password on the Dosplus diskette was not set. (Press ENTER when it asks you for the password). Dosplus is one of the only systems on the market that allows you to enter an <ENTER> for a password and leave it null.

The master password allows you access to the data stored on the diskette; much the same way as a file's password allows you access to the information in the file.

If you know the disk's password, you can change the password using PROT. (see PROT for other uses of the password)

USER FILE OR SYSTEM FILE -

Dosplus has two types of files: System files and user files. (For a definition of a "file", see the FILESPEC section)

System files are those that are essential to the proper operation of Dosplus and its library commands. They are denoted by the "/SYS" extension and by the "*" in the ATTRB column of the DIR function. (see FILESPECS and DIR)

The general layout of the system files is :

SYS0-SYS4

These systems are VITAL to the operation of Dosplus. If they are not present, the DOS will NOT function properly to even load and execute programs.

SYS5

Error messages. This system also cannot be safely purged from the disk.

SYS6

Debug system. If not being used, it can be purged.

SYS7-SYS15

These contain the library commands. They are interactive (call one another). If you are going to remove one for the sake of room, you may as well just purge them all and forfeit your library commands.

Therefore, a minimum system disk to load and execute machine language programs would be systems 0-5. (to remove system files, see PURGE)

User files on the other hand, are files that you have created (or that we created for you), and may be BASIC programs, machine language programs, or data files.

None of these programs are vital to the DOS's safe operation, and so may be stored on a data disk (or the system disk if there is room).

INTRODUCTION TO HARD DRIVES:

This section is designed to introduce you to basic hard disk operation. This is an entirely new realm for the user who previously only had floppy disk drives. Although, if you are not using Dosplus 4.0 for a hard drive this section is not required reading, we still recommend that you read it in order to glean an understanding of the new double headed operation and also what a cylinder is.

Dosplus 3.4/4.0 has several areas of operation that will seem entirely alien to the user's of previous Dosplus systems. The first and formost of these is :

OPERATION OF HARD DISKS -

Dosplus 4.0 runs with a variety of controller boards and interfaces with many of the currently sold hard drives. The new CONFIG section includes the needed parameters. However, a more detailed explanation may assist.

"Platter count or PC". A hard disk is comprised of platters. They can have between one and four platters (the drives we will be using). Picture each platter as something similar to a phonograph record. Each platter has two recording areas. Each of these is called a "surface".

Each surface has a certain number of formatted "tracks" for recording data. This track is the same type of track as you have been used to in floppy disks. For every track on the top surface of a platter, there is a corresponding track on the bottom surface. This is true for each platter. Therefore, a two platter hard drive would have FOUR track zeros. Track zero, top and bottom, for the first platter and again for the second.

The idea in hard drives is to PREVENT HEAD TRAVEL. If you can move the head as little as possible, you will minimize data access time. And that is the name of the game. This brings me to "cylinders".

Before, in the floppy drives, the head moved in and out one track at a time. In the hard drive, with two surfaces for each of multiple platters, this would result in incredible head travel and inefficient use of the drive. Therefore we have cylinders. Dosplus 4.0 no longer steps in and out by tracks. It advances an entire cylinder.

In the hard drive each cylinder is comprised of all corresponding tracks on both sides of all platters. What this means is as the DOS is writing to the disk it will write to track zero on platter one's top surface, and the track zero on platter one's bottom surface, and then track zero on platter two's top surface, and finally track zero on platter two's bottom surface before advancing to track

one. This is of course assuming a two platter hard drive, yours could have more. Therefore, each cylinder consists of (PC*2) tracks.

This is why so much information is needed in CONFIG when setting up your hard drives. It is important to know how many sectors are in each cylinder. You get this figure by multiplying the number of sectors in each track times the number of surfaces (remember, each platter has two surfaces). No cylinder may have more than 255 sectors in it. This is important, the number of sectors per cylinder, because the granule size parameter must divide evenly into this figure. No granule may span a cylinder and the maximum number of granules on any one cylinder is eight.

Example -

Let us say that I have a Seagate five megabyte hard drive. This is a two platter drive capable of 33 sectors per track.

Therefore I would set track size equal to 33. Now, according to my formula CS=(TS*NS), cylinder size equals track size times number of surfaces, each cylinder would have (33*4) sectors or 132.

Now, my granule size must divide evenly into this figure. I discover, through careful calculation, that 22 divides evenly into 132 six times. If I set granule size to 22, it will give me six granules on each cylinder with each granule about 5K. This is optimum, so that is my chosen setting.

Also, since we know that ideally the directory should not be more than one granule, we would set the directory size to 22 sectors. This will allow about 180 user files.

These therefore are the optimum settings for operation of the Seagate five megabyte hard drive :

```
PC=2   Two platters.
TS=33  33 sectors per track.
GS=22  22 sectors per granule.
DS=22  22 sectors for the directory.
```

This would leave you with 132 sector cylinders and six granules per cylinder. You can have anywhere from 100 to 200 cylinders in the hard drive. The hard disk formatter will query you as to how many cylinders you wish to format. Please note that hard disks are not as forgiving as floppies. Do NOT try to squeeze a "couple more cylinders" out of the hard drives. Answer the question with the correct number of cylinders per your drive owner's manual.

This brings me to the second great change in Dosplus 4.0 :

OPERATION OF FLOPPY DRIVES -

By now you are wondering if there is anything we left alone. Nope. It is a whole new ball game when you start working with hard drives. What we have done is change the floppy disk operation to be uniform with the hard disk.

There are no more tracks on floppies. We now have cylinders. Now, on a single headed drive, a cylinder is identical to what a track used to be. However, on a double headed drive, a cylinder functions like it does on a hard drive. It uses the corresponding track on the second side of the disk. This means that Dosplus 3.4/4.0 sees a double headed drive as one drive. It simply has bigger cylinders.

The table looks like this :

Disk formats	Sec/cyl	Grn/cyl	Sec/grn
5" SDEN 1 SIDE	10	2	5
5" DDEN 1 SIDE	18	3	6
5" SDEN 2 SIDE	20	4	5
5" DDEN 2 SIDE	36	6	6
8" SDEN 1 SIDE	16	2	8
8" DDEN 1 SIDE	30	5	6
8" SDEN 2 SIDE	32	4	8
8" DDEN 2 SIDE	60	6	10

Each sector has 256 bytes.

This is all controlled through the "size=" and "sides=" parameters in CONFIG. This is all internal to the DOS. Simply set each drive for the correct size and number of sides and the DOS controls the rest.

The floppy disk format utility will ask you for the number of cylinders. From then on, as always, Dosplus will handle things automatically. Dosplus 3.4/4.0 is no less automatic then any other version ever was. It will automatically adjust itself to the number of cylinders per drive and the density of that drive. But unless you CONFIG it, we have no way of knowing when a drive is eight inch or double headed.

A FINAL WORD -

Please do not be intimidated by this new form of doing things. It is not that much different. You will get used to it quickly. All we have changed is that double headed drives are now viewed as ONE drive. It just has more sectors per cylinder since each cylinder "wraps around" to include the same track on the second side.

This can cause problems converting your old double

headed disks. For the most part, your double headed data disks must be treated as single headed and TRANSFERed between the old disk and the new one.

However, in the event of a double headed drive zero, you will require a special system disk if you wish to utilize that drive double headed. You have the SYSGEN utility for that (see SYSGEN). This is not also true for eight inch drive zeros. Eight inch drive zeros require a disk special ordered from us. As a rule, I would remain with a 5.25 inch drive zero.

Remember, you cannot do a direct backup between a single and a double headed disk drive.

BUILT IN FEATURES -

REPEAT LAST COMMAND - You can repeat the last DOS command entered by typing a "/" (slash mark), and pressing <ENTER>. This will repeat the last command given at the DOS command level. This is zeroed every time the system is re-booted.

LOWER CASE - You may switch to lower case by typing <shift> <0>. You have two modes, lower case with shift upper and upper case lock. The shift zero toggles the two modes.

DEBUGGER - Upon power-up (or reboot), you can hold down the "D" key as the system is booting and enter the debugger (see DEBUG) directly. This does not turn the debugger on, but merely engages it for the moment.

ROM DRIVERS - You can boot up DOSPLUS and default to the ROM I/O drivers by holding the <shift> and <up arrow> keys together. Successful completion of this will be indicated by the DOSPLUS prompt with the blinking cursor. Although it resets the printer driver also, this is primarily designed for games and the like that require the ROM keyboard driver in order to work properly. It is NOT recommeded for serious programming because certain DOSPLUS features such as the spooler and a "do" file will not work without our keyboard driver.

LISTS - Most listing type functions (LIST, MAP, DIR, CAT, etc., etc.) can be halted by pressing <shift> <@> and aborted by hitting <BREAK>. Most utilities will halt at one screen full of output and wait for an <ENTER> to continue.

REPEATING KEYS - Simply holding down a key will cause it to repeat. The system will delay 1/2 second and then begin repeating the character until it is released.

SCREEN PRINTER - Dosplus now uses the ROM screen printer in the Model III. It will vary as to your machine. On some it is "S" and "P" pressed together, on others it is <shift> <down arrow> "*" pressed together. In the Model I, it is still <shift> <clear>.

LIBRARY COMMANDS:

The following are the library commands for DOSPLUS. They are entered from the command mode.

APPEND - Merges ASCII disk files together
ATTRIB - Alter a file's protection attributes
AUTO - Auto command execution from system start-up
BREAK - Disables/enables the <BREAK> key
BUILD - Create a DO file
CAT - Displays file catalog or names and extensions
CLEAR - Fills memory and disk files. Adjusts HIMEM
CLOCK - Displays real time clock on screen
CONFIG - Sets disk drive system parameters
COPY - Transfers files from one disk drive to another
CREATE - Preallocates space for a disk file
DATE - Displays or sets the date
DEBUG - Enables or disables DEBUG program
DIR - Displays Directory of disk files
DO - Begins automatic execution of BUILD file
DUMP - Transfers RAM file to disk
FORCE - Reroutes logical devices
FORMS - Sets parameters for printer drivers
FREE - Displays disk map and amount of free space
JOIN - Links two logical devices
KILL - Deletes a file from the disk
LIB - Displays list of library commands
LIST - Displays a file from disk
LOAD - Loads file into memory without execution
PAUSE - Halts execution awaiting operator action
PROT - Alters protection status using master password
RENAME - Changes the name of a disk file
RS232 - Reads and sets RS232 Serial Port
TIME - Displays or sets the time
VERIFY - Automatic read after write

APPEND -

This command is used to add one file to the end of another. This command is to be used for DATA files or BASIC programs that have been saved to disk in ASCII only. Do not attempt to use it for regular BASIC programs. If you do, you may not be able to load it. The format is:

APPEND filespec1 TO filespec2

This will cause the first data file (filespec1) to be added to the end of the second (filespec2). The name of the new combined file will be the same as that of the original second file (filespec2). For a more complete explanation of FILSPEC, see the section of the manual labeled 'NOTES ON FILSPECS'.

Example : APPEND DATA1:1 TO DATA:0 <ENTER>

The name of the new file on drive 0 will be DATA.

NOTE: The separator TO in DOSPLUS is optional. In the above example, you could have used the format:

APPEND DATA1:1 DATA:0 <ENTER>

WARNING: If you append a BASIC program to the end of another; when you load the combined program into the computer the lines with the same numbers will be merged with the second line overlaying the first.

Append can be used as a sort of dynamic disk merge. Just realize that when you load the program back in, the lines from the first file will be the last to load in and will therefore overlay the lines from the second file. You are ADDING THE FIRST FILE TO THE END OF THE SECOND. It is that simple.

Now for the complicated part. APPEND has been expanded to allow you to append a logical device onto the end of a file. This would allow you to, for example, append any keyboard input to the end of a build file. Or possibly to route the RS232 input to the end of an ASCII file that had earlier been created using the method of copying a device to a file (see COPY).

This gives you what is called "device independence". That is simply a fancy term for being able to tell the various devices what to do and with whom.

Appending a device to a file is essentially the same thing as copying a device to or from a file (see COPY), except that if you append a device to a file, it will position to the end of the file instead of overwriting.

This is useful in routing additions to text files.

Again, please use extreme caution when re-routing logical devices. In order to give the degree of flexbility desired, these areas are not as "goof-proof" as other areas of the DOS. Carefully think out your logic. The novice user/programmer may find himself "hanging up" his system.

SAMPLE USE -

Suppose you had a BASIC program on drive zero named TEST1. You also had a program named TEST on the same drive. You could type :

```
APPEND TEST1:0 TEST:0
```

And the DOS will combine the two. To display this :

```
TEST

10 PRINT"This is a test          ":GOTO 15
15 PRINT"This is also a test     ":GOTO 20
20 PRINT"Looping back            ":GOTO 10

TEST1

20 PRINT"This is still a test    ":GOTO 25
25 PRINT"Looping back            ":GOTO 10
```

After you typed the above command and appended TEST1 onto the end of TEST, you would have this file for TEST :

```
TEST

10 PRINT"This is a test          ":GOTO 15
15 PRINT"This is also a test     ":GOTO 20
20 PRINT"This is still a test    ":GOTO 25
25 PRINT"Looping back            ":GOTO 10
```

You will notice that the line 20 in TEST1 overlaid the line 20 in TEST AFTER the program has been loaded into BASIC. If you were to list the diskfile (see LIST), you would see TWO line twentys, but BASIC will overlay the second on top of the first when the program is reloaded.

TEST1 will remain unaffected by this.

MASTER PSWD = .XANTH

ATTRIB -

ATTRIB is used to change the attributes of a disk file. You can alter the following attributes:

1. Make a file visible/invisible.
2. Change access or update passwords.
3. Change the level of protection.

The format is:

ATTRIB filespec:d (A=,U=,P=,I) <ENTER>

Where the parameters can be:

I - Make a file visible/invisible.
A - Access password.
U - Update password.
P - Protection level

The protection levels are:

Level	Parameter	Function
0	FULL	No Protection
1	KILL	Able to delete file
2	RENA	Rename, Write, Read, Execute
3	----	Not used at this time
4	WRIT	Write, Read, Execute
5	READ	Read, Execute
6	EXEC	Execute only
7	NONE	No access, can't be set by user

Access level 7 is set by Micro-Systems. It is for system files only. You can not make use of it. The only function that can affect these files is the PURGE command.

EXAMPLE : ATTRIB TEST1/BAS (I) <ENTER>

This will make the program TEST1/BAS invisible. However, if the program was already invisible, it will make it visible again.

Technical note : If you alter a file's visible/invisible status, it will remove whatever protection level that you have set. It will NOT affect password status.

EXAMPLE : ATTRIB TEST1/BAS (A=password) <ENTER>

This will assign the specified password as the access password for the program TEST1/BAS.

EXAMPLE : ATTRIB TEST1/BAS (U=password) <ENTER>

This will assign the specified password as the update password for the program TEST1/BAS.

NOTE : To remove a password, you need only to enter the parameter A= or U=.

EXAMPLE : ATTRIB TEST1/BAS (U=PASSWORD,P=EXEC)

This will set the protection level so that the program can be executed only. Remember, you must enter the WORD for the level of protection you desire and not the DIGIT.

You can use more than one parameter at the same time. The access password, if set, must be included in the filespec in order to execute the program. If the access password is not set, then the program can be RUN without knowing the password (depending on the protection level).

The update password, on the other hand, is the one you must know in order to update the file. Updating the file is writing to the file, using DISKDUMP on the file, or in any way physically altering the file.

You could, for example run a file that you did not know the update password for, but you could not load or list it.

Note that each level of protection allows all specified access, plus the the access of all lower levels.

SAMPLE USE -

If you have a payroll program that you want your secretary to RUN but not MODIFY, you would enter :

ATTRIB PAYROLL (A=PAYROLL,U=SECRET,P=EXEC)

Now tell your secretary to use the password PAYROLL when running the program (BASIC PAYROLL.PAYROLL). This allows him/her to execute the program ONLY! Only you, by loading it with the password SECRET, can modify the file.

Note that all BASIC files require a status of at least READ in order to load and run.

If you wish to set up a BASIC program with "Run only" protection, you would type :

ATTRIB PAYROLL (A=,U=SECRET,P=EXEC)

Now tell the operator to run the BASIC file PAYROLL. He/she will not need a password to do this as no ACCESS password has been set. However, they will only be able to RUN the program. They cannot LOAD, LIST, or STOP EXECUTION of the program without knowing the password SECRET.

Even BASIC * cannot retrieve the file.

AUTO -

This command will allow you to execute a system command or a program from system power up or reboot. The format is :

AUTO parameter

where the parameter is a library command or a machine language program name.

EXAMPLE : AUTO DIR

This will automatically display the directory of the disk in drive 0 on power-up.

You can abort an auto by holding down the ENTER key as you boot up. However, this can be disabled by making an asterisk the first character of your auto statement.

EXAMPLE : AUTO *BASIC PAYROLL-F:4

This will load BASIC with four files open and run the program 'PAYROLL' even if you hold down the ENTER key. If the asterisk is the first character, the auto becomes non-breakable.

AUTO can also be used to enter a "do" file. For example :

AUTO DO START

would execute the build file "START/BLD" automatiaclly upon powerup.

Ths system does not check for command errors at the time that the AUTO is entered, but rather will detect it at the next powerup. Remember that it will interpret whatever is typed JUST AS IF YOU TYPED IT FROM THE KEYBOARD. If you give it the name of a program, it has to executable from the DOS command mode. Otherwise, BASIC must be run first. By using the BASIC auto-run feature, though, you can still jump right in to your BASIC programs. (see BASIC)

To reset the AUTO parameter, simply type AUTO with no command line. Example :

AUTO <ENTER>

This will remove the current auto, and return that system diskette to normal bootup.

BREAK -

This will allow you to enable/disable the break key. The format is :

BREAK (OFF) to disable
BREAK (ON) or simply BREAK to enable

This command can be executed from BASIC, and is useful in preventing unwanted interruptions in a program.

There should be a certain level of caution used in this command. For example, once the BREAK key is disabled, there is no escape from the LIST command in BASIC or in DOS.

The BREAK key is used by many utilities (DISKDUMP, PURGE, FORMAT, BACKUP, etc.) as the method of aborting the operation. If you have rendered the BREAK key inoperative, you are in effect "locked in", and may have no recourse short of rebooting the system.

The best rule is simply to remember that whenever you disable the BREAK key going into a program, you should restore it when you leave.

BUILD -

This feature allows you to create a DO file. This file can contain a series of command lines up to 63 characters in length. These command lines contain DOS commands or program names.

The primary function of this feature is to allow you to execute a number of DOS commands right from power-up, and then enter your program directly without any operator intervention.

It enables the programmer to set up a "ifitialize" sequence such as loading the routine for a sort program, set FORMS, set the RS232, and then load BASIC, and protect the memory locatign of the machine code, etc., etc.; all without operator intervention. This means that the operator does not have to be familiar with the DOS, just his program.

The format is:

BUILD filespec

If you don't specify a file extension, DOSPLUS will automatically add /BLD onto your filespec. You will then see:

Type in up to 63 chars

At this point, type in a single command line (you can't have two commands on one line) and then press <ENTER>. DOSPLUS will then prompt you for the next command line. When you have finished entering commands, press the <BREAK> key in response to the prompt and your file will automatically be saved on the disk in drive 0.

EXAMPLE :

```
BUILD START <ENTER>

Type in up to 63 chars
FORMS (S) <ENTER>
Type in up to 63 chars
BASIC LEDG/BAS.PWD-F:3-M:61285 <ENTER>
Type in up to 63 chars
<BREAK>
```

This will create a file on your disk named START/BLD and return you to DOS. To use this file type:

AUTO DO START

Now, when ever you boot your system with this disk in drive 0, you will automatically initialize the Serial printer driver, protect memory above 61285, open 3 files, and then run the BASIC program LEDG/BAS that is password

protected.

A build file can be created from BASIC or any program as long as you remember to never exceed more than 63 characters without including a carriage return.

Be certain the file is saved in ASCII format. Remember that the file you are creating will be accepted as if it were a DOS command line. The file must contain data that would normally be typed in via the DOS command mode.

CAT -

This command will display a file catalog of the diskette (sometimes called a short directory). The file catalog consists of the filename and extension ONLY. No other information will be available with this command. The syntax is :

CAT :d (parameter,parameter,parameter)

You have the following parameters :

I - display invisible files also.
T - Model III TRSDOS disk.
P - send output to printer.

If the parameter (I) is not used, only the visible user files will be displayed. System files cannot be displayed from this command, it is for user files only. System files may be displayed from the DIR command.

The (T) parameter will allow you to catalog a disk that has been formatted by, or contains Model III TRSDOS. DOSPLUS still does not support standard I/O to TRSDOS, but the ability to pull a short directory when using CONVERT (see Utilities) could save time invested in having to boot up each TRSDOS disk just to see if there is a file on it that you want to convert. If you use the (T) parameter on a DOSPLUS diskette, the error message "Disk not as expected" will appear.

The (P) parameter, if included in the parameter field, will send the output that normally goes to the screen to the line printer. If your line printer is not available, the system will halt until the printer is brought on line.

Your output should look something like this :

```
FILE CATALOG DRIVE: 0  - DOS:3.3
TESTPGM/BAS     BASIC/CMD      S/CMD          BRUN/CMD
LETTER01/LC     PRE/LC
```

There will be a maximum of four entries on a line or sixty on a screen. When one screenful is reached, the system will pause and wait for you to press ENTER before proceeding to display the next screenful.

For those of you developing machine language programs, CAT is now available as a system call. Consult the technical section.

CLEAR -

This command is used to clear a disk file or it can be used to clear user memory. It is also used to reset all logical device I/O vectors, adjust high memory. The syntax is :

CLEAR (parameter,parameter)

You have the choice of the following parameters.

HIGH - Adjusts high memory pointer at 4411H to specified address. It will clear memory up to specified address. Sets addresses for other parameters.

RESET - Sets high memory pointer to highest physical address in the machine, closes any FORCEd ot JOINed files, resets all logical device vectors to normal, clears memory, resets 'DO' buffer, and resets ALL user interrupt tasks.

DATA - Whatever you set this parameter equal to, it will use that byte when it is clearing. Otherwise, it will simply use zeros. Only one byte (0-255 or 00H-FFH) is valid.

START - Address at which you wish to begin clearing memory. It cannot be less than hex 5700. If not specified, hex 5700 will be assumed.

END - Address at which you wish to stop clearing memory. It cannot be greater than the top of memory, and if it is not specified, the highest available address will be used.

EXAMPLES -

CLEAR (HIGH=7000H,DATA=140)

CLEAR (START=7000H,END=A000H,DATA=41H)

CLEAR (RESET)

The first will set the high memory pointer to 7000 hex, and fill all memory between hex 5700 and hex 7000 with 140 (hex 8C). Clear will always clear memory (except when clearing a file). There is no option for setting the high memory pointer without clearing out memory.

The second will fill all memory between hex 7000 and hex A000 with 140s (hex 8C). Remember that the data can be either hex OR decimal. Simply append the "H" to hex

values.

The third example will execute a "system reset". (see RESET in the parameter list)

If you specify an optional filespec, CLEAR will clear out a file. This replaces the CLRFILE utility of earlier versions of DOSPLUS. The format is :

CLEAR filespec:d

The only valid parameter when clearing out a file is the DATA parameter.

Remember that all hex values MUST have a trailing 'H' (otherwise CLEAR will assume it to be a decimal value), and remember that if you use CLEAR (RESET), you cannot execute that from a 'DO' file. All other forms of CLEAR are accepted from a 'DO' file, except clearing the file you are executing.

EXAMPLE : CLEAR TEST:0 (DATA=65)

This will fill the file named TEST on drive zero with capital 'A's. Remember that ALL parameters other than DATA are invalid while you are working with a file.

Note that when a file is cleared, the previous data is WIPED. Use extreme caution and forethought before clearing a file.

This command is highly useful, but also extremely complex. Remember that when you use the CLEAR (RESET) option, you literally reset the WHOLE machine. Before you say that you can't get it working and proceed to get upset with us, ask yourself these questions :

1. Have I appended an 'H' onto all hex values?
2. Have I left a space between CLEAR and the parameters and/or the filspec?
3. Have I left a space between the filspec (if included) and the parameter?
4. Have I left out all spaces within the parenthesis?
5. Am I using the RESET parameter incorrectly?
6. Am I using an illegal parameter on a file?

You are not limited to only one or two parameters, though. You can combine the parameters in any legal fashion.

CLOCK -

This command will display the time in the upper right hand corner of the screen. The format is:

CLOCK (parameter)

The only parameters for this command are ON or OFF. However, if a parameter is not typed DOSPLUS will assume the parameter ON.

The real time clock is always running whether it is displayed or not, except during disk and cassette I/O. When the display is turned on, the clock will be displayed and updated every second as long as interrupts are left on. (for an explanation of interrupts, see the technical section)

EXAMPLE : CLOCK (OFF) <ENTER>

NOTE: The clock may lose some time when the disk is accessed. Also, no error checking is done for valid characters input to the clock parameter. If you type in invalid characters, they will be displayed as such.

In the Model III, when the clock reaches "23:59:59", it will reset to "00:00:00" and the date will be incremented. In the Model I, the clock simply resets itself to "00:00:00" and the date is unaffected.

CONFIG (Set non-automatic system parameters) -

This command allows you to custom configure your system to your specific needs. Using this command, you can optimize your disk access procedures. You can modify the following items:

1. Track to track step rate.
2. Double sided drive operation.
3. Eight inch drive operation.
4. 40 track operation in 80 track drives.
5. Hard disk parameters (4.0 only).
6. Number of drives (Model III only).
7. Master drive.
8. System drive (4.0 only).
9. Automatic mode activation (Model I only)

********** IMPORTANT NOTE !!! **********

To make a permanent configuration change, you must now type :

CONFIG (SAVE)

Otherwise, the change you have made affects ONLY the system in memory and will be removed when you reboot.

Please do NOT attempt to permanently configure your system disk when it is write protected. If you do, this will produce a "Disk write protected" error. When you save a configuration, the disk is being written to, and it can't very well do that when a write protect tab is in place.

****

The format is:

CONFIG :dn (par=exp,par=exp....) <ENTER>

":dn" is the drive number that you wish to configure.

"par" is the parameter that you are configuring. A list of configurable parameters appears below. If you specify an incorrect parameter, the error "Parameter error" will occur.

"exp" is the expression to be evaluated by the DOS in altering the specified parameter. If it is an invalid expression, the error "Specification error" will result.

Where you can have the following parameters:

DRIVES (Model III only) - Sets the number of drives in the system. This is primarily intended to prevent system lockup on Dosplus 4.0 when a hard disk controller is not on line. If the hard disk controller is NOT on line and you do a global search for a file, the system will hang. If the controller isn't on line, then configure the system for no more than four drives. Also, it will speed up a global search on systems with less than four drives if you configure for the exact number of drives. The DOS will not waste time attempting to access non-exsistent drives.

EXAMPLE -

CONFIG (DRIVES=3)

MASTER - Sets the default master drive. This allows you to route unspecified disk I/O to a default drive. The "master" drive is defined as being the drive the DOS seeks to access first (i.e. the first drive checked during a global search). This is NOT a software write protect. The specified drive will always override the MASTER parameter. The master drive will be displayed by an "*" next to the corresponding drive number.

EXAMPLE -

CONFIG (MASTER=4)

SYSTEM - Allows you to define which drive in the system contains the operating system. This is valid on Dosplus 4.0 only. Its intended usage is to set the hard drive as the system drive. You would first HFORMAT the drive, then SYSGEN it and configure it as the SYSTEM and MASTER drive. After that you only need the floppy disk to boot up. All other system needs will look to the hard drive. (see also : SYSGEN)

EXAMPLE -

CONFIG (SYSTEM=4)

Note : The system drive is designated by a "$" next to the correct drive number. If the system and the master drive are the same, only the "*" will be displayed.

MODE - (Model I only). This allows you to set an integer value between 0 and 255 that, upon powerup, will be output to port 254. It replaces the SPEED parameter from earlier versions of Dosplus. In the Model I many aftermarket modifications use port 254 for their various modes of operation. In the past, you were forced to go to BASIC, output to port 254, and then return to DOS. This allows you to set the value you wish to use and have that output upon powerup, thereby avoiding the need to go to

BASIC. Also, because CONFIG now affects the system in memory immediately, you may use this parameter to do the output directly from DOS in a temporary application (again freeing you from having to load BASIC).

Example -

CONFIG (MODE=1)

STEP - Sets the drive head step rate. The faster the head moves, the faster your disk I/O. The rate must be compatible with your disk drives. Some drives allow much faster access than others. The allowable step rates are 6, 12, 20 and 30 (FDC 1791) or 40 (FDC 1771) milliseconds. For a hard disk, they would be 1 through 7 milliseconds.

EXAMPLE -

CONFIG :2 (STEP=6)

SIZE - When this parameter is set to eight (8), the DOS will attempt to activate eight inch operation when accessing that drive. Special hardware is required for this feature. You must be using a Dosplus compatible eight inch disk controller and drive.

EXAMPLE -

CONFIG :3 (SIZE=8)

SIDES - This allows you to configure a drive as double or single sided. Using a double headed drive may require a special drive cable. Consult the manual that came with your drive. Drive zero may not be configured as (SIDES=2). SYSGEN will set that parameter for you when create the double headed system diskette (see SYSGEN).

Technical note : When you configure your system for double headed drives, Dosplus now sees both sides of that drive as one drive (single volume). You may use a double headed drive like a single headed one, but you may not use a single headed drive as double headed. For example, you could BACKUP a single headed drive to a double headed one, but not vice versa (re-configuration would be necessary). When you configure any drives on the Model I as double headed, you lose the drive three position (that select line is used for the side select). The Model III does not have that limitation.

EXAMPLE -

CONFIG :1 (SIDES=2)

SKIP - This will allow you to read a 40 track disk in an 80 track drive. If the SKIP is set to 'Y' for a drive, the DOS will step that drive twice when it would

normally step it once. This will allow 40 track operation in 80 track hardware.

EXAMPLE-

```
CONFIG :2 (SKIP=Y)
          or
CONFIG :2 (SKIP)
```

In the second example, the "Y" expression on skip is assumed. You only have to specify "SKIP=N".

********** SKIP WARNING **********

Due to the fact that not only is the distance between the tracks smaller on a eighty track drive then on a forty, but also the tracks themselves are slighty narrower, there are certain distinct limitations to the 'skipped' drives.

Any time that a diskette is written to by a forty track drive, and then written to again by an eighty track drive, the forty track drive will no longer read it.

The track written by the forty track drive is slighty wider, and so when the eighty track drive writes to the disk, it only overwrites part of the track. Then the forty track drive not only reads the new data, but it gets part of the old data as well. This WILL produce read errors.

Unless you are SOLELY reading from a diskette, we do not recommend the constant passing of one disk back and forth between a 'skipped' eighty track drive and a regular forty track drives. Note the following rules of thumb :

1. When receiving a forty track disk for your eighty track drive, use the skip parameter to MAKE A BACKUP. Do not write to the original disk, and you will not spoil it for standard operation.

2. When preparing a diskette for a forty track drive in your eighty track machine, BULK ERASE THE DISK FIRST! Do NOT use FORMAT. The new format will not wipe out any old, wider tracks.

3. After a disk has been written to by a forty track drive, and then by a 'skipped' eighty, DO NOT TAKE THAT DISK BACK TO A FORTY TRACK DRIVE! As a rule, use the skip parameter to make copies of the forty track disks that you want.

****

NOTE : The following parameters, "PC", "TS", "GS", and "DS" pertain ONLY to the hard drives. Please consult the section "Introduction to hard drives" at the beginning of this manual.

PC - Platter count. This should be set equal to the number of platters in your particular hard drive. Most drives are either two or three platter. This number could be anywhere from 1 to 4.

EXAMPLE-

CONFIG :4 (PC=3)

TS - Track size. This should be set equal to the number of sectors that you want the DOS to format each track to. One cylinder may not exceed 256 sectors. For a detailed explanation of cylinders, please consult the section "Introduction to hard drives" in the beginning of this manual. A track must be greater than nineteen sectors.

EXAMPLE -

CONFIG :5 (TS=32)

GS - Granule size. This determines the minimum allocation of space to any one file when writing to the disk. No granule may be longer than 32 sectors, and the Granule Size must divide equally into the number of sectors in a cylinder. The maximum number of granules on any one cylinder is eight.

EXAMPLE -

CONFIG :6 (GS=20)

DS - Directory Size. This allows you to set how many sectors go into the directory. This must be a value between 3 and 34 sectors in length and may not exceed the length of a track. Ideally, the directory should be the same size as a granule (i.e. the directory would only take up a granule on the disk.)

EXAMPLE -

CONFIG :7 (DS=33)

If you type CONFIG without any parameters you will get a display of the current system configuration.

EXAMPLE : CONFIG <ENTER>

```
*:0 STEP=30,SIDES=1,SIZE=5,SKIP=N
 :1 STEP=30,SIDES=1,SIZE=5,SKIP=N
```

```
:2 STEP=30,SIDES=1,SIZE=5,SKIP=N
:3 STEP=30,SIDES=1,SIZE=5,SKIP=N
:4 STEP=3,PC=2,TS=33,GS=22,DS=33
:5 STEP=3,PC=2,TS=33,GS=22,DS=33
:6 STEP=3,PC=2,TS=33,GS=22,DS=33
:7 STEP=3,PC=2,TS=33,GS=22,DS=33
```

This is the way you received the original diskette (if it was 4.0). If it was 3.4, it looked like this :

EXAMPLE : CONFIG <enter>

```
*:0 STEP=30,SIDES=1,SIZE=5,SKIP=N
 :1 STEP=30,SIDES=1,SIZE=5,SKIP=N
 :2 STEP=30,SIDES=1,SIZE=5,SKIP=N
 :3 STEP=30,SIDES=1,SIZE=5,SKIP=N
```

Technical note : Some of the hard disk drive parameters will vary on Dosplus 4.0 depending on which controller board and disk drive combination your system was designed for. Dosplus 4.0 comes assembled for several different types of hard disk units. The letter following the version number indicates to us which version you have. The settings on your original disk are, in our estimation, the optimum settings for your hardware and should not be changed.

The following are some CONFIG examples :

EXAMPLE : CONFIG :2 (SIZE=8,SIDES=2,STEP=20) <ENTER>

With this example you just set Drive #2 as an eight inch, double headed drive and the STEP rate is now 20 milliseconds. Remember you must first have the hardware necessary in order to operate eight inch drives.

Technical note : When working with eight inch drives, the step rate you set will be cut in half in actual usage. Therefore, the drive we just set would really be stepping at 10 ms.

Example - CONFIG :1 (STEP=6,SKIP=Y) <ENTER>

This would have set drive one to step at six milliseconds and enabled the skip parameter (see above : SKIP).

Example - CONFIG :4 (STEP=3,PC=2,TS=32)

With this example, you have just configured the hard disk that is on position four to step at three milliseconds, and have it set as two platter with thirty-two sector tracks. For a complete explanation of the new 4.0 track/cylinder relationship, consult the "Introduction to hard drives" section in the front of the manual.

PLEASE be aware that the way your Dosplus comes configured for the hard drive is the optimum settings for MOST of the hard drives it will be sold with.

Technical note : If a diskette is formatted to 40 or 80 cylinders, the DOS will know this automatically when it goes to access the disk. One easy way of converting a disk for more tracks is to simply format a destination disk with the desired number of tracks (to the limit of your hardware, of course). When DOSPLUS makes a backup, it never loses tracks on the destination disk. Hence you can backup a 35 track disk to a disk that is formatted to 40 tracks, and it will make the backup, copying over the first thirty five tracks but NOT ALTERING THE DESTINATION TRACK COUNT, thus leaving you with a 40 track disk.

COPY -

This command allows you to copy one device/file to another device/file

The format is:

```
COPY source-file TO destination-file
          or
COPY device/file TO device/file
          or
COPY filespec:d :d
```

"source-file" is be the specification of the file to be copied from.

"destination-file" is the name of the file you to be copied to. Can be replaced by a drive/side specification. If so, the same name will be kept.

":d" is the drive specification. If it is not there, Dosplus will search through all available disk drives.

When copying between two files, you may replace the second filspec with a simple drive specification if you are not changing the name during the copy.

```
EXAMPLE :          COPY TST1/BAS:0 TO TST1/BAS:1  <ENTER>
                             or
                   COPY TST1/BAS:0 :1 <ENTER>
```

These will both create a file named TST1/BAS on drive 1. The only time that you DO have to assign a name to the second file is when you are changing the name of the file during the copy process.

You now have the option of copying a logical device to or from a file by replacing either filespec with a device specification.

```
Example :          COPY MODEM/DAT:0 *RO
```

This would copy all the data in the file MODEM/DAT to the RS232 output.

This has several extremely powerful applications. For example, you can now COPY TEST/BAS TO *RO, and it will send the file TEST/BAS out the RS232 board. Or, on the other hand, you could COPY *RI TO TEST/BAS, and store the data coming in the RS232 board in the file TEST/BAS.

When copying to/from a device to/from a file, you must specify both the file and the device. When copying between

two devices, both must be specified.

When it is finished transmitting, it will send an "end of text" control code (CHR$(4) or CTL D) which will tell the other system to close the file and return to DOSPLUS. Your system will also return to DOSPLUS when the file is copied. You can abort a device COPY function by hitting <BREAK> (CHR$(1) or CTL A).

This will ONLY work with files that have been saved in ASCII format. Standard BASIC and machine language files cannot be copied to a device. Also, when using the device copy, you may not use any shorthand save the possible omission of the delimiter word "TO".

You can use this command to copy the keyboard to the printer, thus turning your printer into a typewriter (more or less, depending on the type of printer). You can literally copy to or from any logical device.

Be careful of the order that you use (what you are copying from - to). If you copy from an inactive device (such as *RI) to an active device (such as *KI), the machine will lock up. The computer scans the keyboard constantly, and if there is no input, it simply loops. However, if it were trying to copy the RS232 to the keyboard, it would sit and wait at the RS232 for some kind of input from that board before proceeding.

WARNING -

The applications of this command are limitless, but the shortcomings are large also. The command gives you enormous flexibility, but it also gives you enough rope to hang yourself. This facet of the COPY command was included primarily for the experienced user/programmer. Do not attempt ANY device routing or copying without first carefully thinking through the logic of it.

CREATE -

This feature is used to create a file and preallocate disk space. The advantage of using CREATE is that it will speed up file access time during a write. This is possible because DOSPLUS does not have to periodically allocate space on the disk as it writes. Files that use preallocated disk space are generally less segmented on the disk and the disk drive head does not have to travel as much during disk I/O.

The format is:

CREATE filespec:d (NRECS=n1,LRL=n2)

"filespec" is a standard Dosplus file specification.

":d" is the drive specification. If omitted, Dosplus will seek the first available drive.

"NRECS" is the number of records to preallocate to the file. It can be any value the disk has space for. If not specified, 0 will be used.

"LRL" is the logical record length of the file being created. It can be between 1 and 255. If omitted, 256 will be used.

Be certain that LRL mutiplied by NRECS does not equal more space then you have on the disk.

EXAMPLE : CREATE ATS/DAT:1 (NRECS=100,LRL=128)

Creates a file named ATS/DAT on drive one with a logical record length of 128 and with 100 records allocated.

If the filespec already exists, CREATE will abort and return to DOSPLUS. CREATE is not mandatory. If you don't preallocate disk space, DOSPLUS will do it automatically as more space is needed.

This allows faster disk access (because Dosplus will not have to periodically allocate more space), and it allows you to know in advance whether or not a diskette will have enough room to contain a specific data file.

LEGAL -

CREATE PAYROLL/DAT.PAYDAY (LRL=128)

Creates a 128 byte long file named PAYROLL/DAT, protected by the password PAYDAY, and allocates 0 records to it.

```
CREATE DUMMY:0 (NRECS=100)
```

Creates a 256 byte long file named DUMMY on drive zero, and allocates 100 records to it.

ILLEGAL -

```
CREATE BADFILE/BAS:0 (NRECS=1000)
```

More sectors than disk contains. (If you have drives larger than 40 tracks, this figure will change)

```
CREATE WORSEFIL/BAS (LRL=256,NRECS=10)
```

Specifying 256 for LRL. You cannot specify the default parameter. If you desire it, omit it.

Once created, the file can and will be extended automatically if Dosplus needs more space than is allocated.

DATE -

This feature allows you to display and set the current date. The format is:

DATE <ENTER>

This will display the current date and it requires no parameter.

DATE mm/dd/yy <ENTER>

This will set the date to a specified date where mm is a two digit month, dd is a two digit day and yy is a two digit year. You must add a zero to all single digit numbers.

EXAMPLE : DATE 11/05/80 <ENTER>

This will set the date to November 5, 1980.

Once the date is set, DOSPLUS will use that date anytime it would normally request you to enter a date from the keyboard. For example, when making a backup copy of a disk, if the date is set, the date question will be skipped. Once the date is set, it will remain set until you reboot the system by pressing <RESET> or by turning the computer off.

Please note that no error checking is done for the validity of the date past simply making sure that all input is numeric and that it is in the correct "mm/dd/yy" format.

DEBUG -

DEBUG is a powerful disk based monitor. With it you can examine any memory location, in both RAM and ROM, or any CPU register. You may also change the content of a RAM location or register. DEBUG is so powerful that it should be used with caution, because it is easy to accidentally destroy a program.

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 three ways to call DEBUG when it has been enabled. They are:

1. Pressing <SHIFT><BREAK> at any time
2. Automatically after a machine language program has been loaded and before the first instruction has been executed
3. Called automatically when an error occurs during program execution

The format for enabling DEBUG is:

DEBUG (parameter) <ENTER>

The only parameters are ON and OFF. If no parameter is specified, DOSPLUS will asume ON.

Once DEBUG is called, you have the following commands:

Command	Operation performed
A	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 aborts, space bar records change and moves to next address.
O	Exit to DOSPLUS (DEBUG still engaged)
Rpr aaaa	Alter register pair (pr) to aaaa. Space between register pair and value is required.
S	Set full screen memory display mode
U	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  = 00FF SZ1H1PNC
BC  = 3880:  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
DE  = 4015:  01 32 4C 00 01 07 00 FF  07 73 04 AF 3C 20 8C 00
HL  = 4300:  FE 41 20 13 CD EC 51 7E  FE 20 38 04 FE C0 38 02
AF' = 0CE9 SZ1-1--C
BC' = 0000:  F3 AF C3 15 30 C3 00 40  C3 00 40 E1 E9 C3 12 30
DE' = 3F40:  20 20 20 20 20 20 33 43  39 30 3A 20 20 33 33 20
HL' = 6046:  0C 08 7E 3C E6 1F 77 08  FD E1 C9 21 36 40 01 01
IX  = 4296:  06 3E 4D 00 00 50 52 01  1E 30 00 00 52 49 02 21
IY  = 7B6A:  00 1E 00 20 00 0D 11 8F  05 00 3C 45 3C 01 00 1D
SP  = 41D9:  3F 3F 4C 00 E3 05 00 43  60 4E 2D 40 01 1E 30 00
PC  = 0698:  C1 C9 AF 32 9F 40 16 FF  C3 8D 2B E6 FD 32 9F 40
     3C80:  44 45 20 20 3D 20 34 30  31 35 3A 20 20 30 31 20
     3C90:  33 32 20 34 43 20 30 30  20 30 31 20 30 37 20 30
     3CA0:  30 20 46 46 20 20 30 37  20 37 33 20 30 34 20 41
     3CB0:  46 20 33 43 20 32 30 20  38 43 20 30 30 20 20 20
```

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 useless 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 (S) :

```
3C00:  20 20 20 20 20 20 33 43  30 30 3A 20 20 32 30 20
3C10:  32 30 20 32 30 20 32 30  20 32 30 20 32 30 20 33
3C20:  33 20 34 33 20 20 33 30  20 33 30 20 33 41 20 32
3C30:  30 20 32 30 20 33 32 20  33 30 20 32 30 20 20 20
3C40:  20 20 20 20 20 20 33 43  31 30 3A 20 20 33 32 20
3C50:  33 30 20 32 30 20 33 32  20 33 30 20 32 30 20 33
3C60:  32 20 33 30 20 20 32 30  20 33 32 20 33 30 20 32
3C70:  30 20 33 32 20 33 30 20  32 30 20 33 33 20 20 20
3C80:  20 20 20 20 20 20 33 43  32 30 3A 20 20 33 33 20
3C90:  32 30 20 33 34 20 33 33  20 32 30 20 32 30 20 33
3CA0:  33 20 33 30 20 20 32 30  20 33 33 20 33 30 20 32
3CB0:  30 20 33 33 20 34 31 20  32 30 20 33 32 20 20 20
3CC0:  20 20 20 20 20 20 33 43  33 30 3A 20 20 33 30 20
3CD0:  32 30 20 33 32 20 33 30  20 32 30 20 33 33 20 33
3CE0:  32 20 32 30 20 20 33 33  20 33 30 20 32 30 20 33
3CF0:  32 20 33 30 20 32 30 20  32 30 20 32 30 20 20 20
```

The left hand column contains the hexidecimal 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) :

```
6900:  T h i s   i s   a n   e x a m p
6910:  l e   o f   A S  C I  I   t e x t
6920:  . C H A R A C T  E R S   F R E E
6930:  :   . D O C U M  E N T   L E N G
6940:  T H :   . V I D  E O   L I N E
6950:  W I D T H :   . P A R A G R A P
6960:  H   I N D E N T  :   . C U R S O
6970:  R   L I N E   N  U M B E R :   .
6980:  C U R R E N T  F I L E   N A M
6990:  E :   . D O C U  M E N T   H A S
69A0:    N O T   B E E  N   N A M E D .
69B0:  S P E C I A L  C O M M A N D ?
69C0:    . P R E S S  E N T E R   T O
69D0:    C O N T I N U  E . H O T   Z O
69E0:  N E   ( 2 - 1 0  ) ?   . H Y P H
69F0:  E N A T I O N   C O M P L E T E
```

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 function will display the Directory of the files on the specified disk. To read the directory of the disk in drive 0, simply type DIR and press <ENTER>. This will give you a listing of all the visible, active files presently on the disk in drive 0. To obtain the directory of a disk other than that in drive 0, you must specify the drive. The format is:

DIR :d <ENTER>

Please note that there is a mandatory blank space between the R in DIR and the colon (:). If this blank space is omitted, the directory of the disk in drive 0 will be displayed.

If you wish to see the directory of files other than the visible files, you must specify a parameter for the type of file you wish to see. The format is,

DIR :d (parameter) <ENTER>

Where the parameter can be:

(S) - System files as well as visible files

(I) - Invisible files as well as visible files (System files will not be displayed)

(D) - All files marked for deletion as well as the visible files.

(P) - Print directory on printer

You can specify more than one parameter.

EXAMPLE : DIR :1 (S,I,D,P) <ENTER>

This will print on the printer a directory of all the files on the disk in drive 1.

The directory gives you a great deal of information about the disk files automatically. DOSPLUS has a unique format. When you call up the directory, you will see a display that looks like this :

```
DIRECTORY DRIVE: 0     DOSPLUS -    02/18/81 - 3.4
FILENAME       ATTRIB  LRL   #LOG   #PHY   #GRN   #SEG   EOF
BOOT     SYS    ISB7   256      5      5      1      1     0
DIR      SYS    ISB7   256     10     10      2      1     0
BACKUP   CMD    I*U6   256      9      9      2      1   248
FORMAT   CMD    I*U6   256      9      9      2      1   248
ACCTS    DAT    N*A4    63    600    150     15      1     0
GETTAPE1 BAS    D*X0     1    966      4      1      1   198
SYS0     SYS    ISB7   256     12     12      3      1     0

DOS PLUS
#
```

When the screen is full, the listing will pause until you press either <ENTER> to display the next full page or the <SPACE BAR> to single step through the files. The <BREAK> key will return you to DOSPLUS.

The first information you will see in the Directory is the drive number that the disk is in and the date the disk was created. The last item on the first line is the version of the DOS used to create (format) the disk. The second line contains the column headings for the directory. The filename column gives you the name of the file and the extension, if any. The next column is the attribute list and is broken up as follows:

The first character is:

N - Normal (visible)
I - Invisible
D - Deleted file

The second character is:

* - User file
S - System file

The third character is the password attribute and is as follows:

X - No password currently set
A - Access password set
U - Update password set
B - Both access and update passwords set

The last character is the level of protection afforded by the access password. For a listing of these protection levels, see the ATTRIB command.

The LRL column gives the logical record length of the file. This is the logical record length specified when the file was first created or opened. Because Dosplus allows you to open a file with a different logical record length than it was created with, the LRL of a file is really only for display convenience. For example, the 1 in this column

for the program GETTAPE1/BAS allows you to see the exact byte count of the file simply by looking at the "#LOG" column.

The LOG column tells the number of records in the file. This allows you to see, at a glance the number of logical records in a given file. "#LOG*LRL" should equal the amount of data in that file.

The PHY column shows the number of sectors used for the file. In the above example, the 966 bytes in the file GETTAPE1/BAS used four disk sectors. This column also gives you the information you need to calculate the number of bytes in a file. The formula "((#PHY-1)*256)+EOF" will give you the length of a file.

The GRN column displays the number of granules used by the file. The sector counts of the various granules (single density, double density, 8", 5.25", etc.) are described in the section "Introduction to hard drives".

The SEG column gives an idea of how the file is distributed on the disk. The 1 in the above examples indicates that the files are written on consecutive tracks and sectors. The more highly segmented a file is, the more head travel that is needed for the system to access it. The DOS will only segment when it has to.

The final column is the end of file marker for that file. This tells you how many bytes of the last physical record this file actually uses. This will be one higher the the number of that byte since we start numbering bytes within a sector at 00H.

DO

This feature begins automatic execution of a file created with the BUILD command. The format is:

DO filespec (HIGH=address) <ENTER>

"HIGH=address" is the location that you want the DO buffer to locate itself. It will set this as the top of memory, and locate itself starting at this address and working down. If it is omitted, DO will assume that you wish to start at the top of memory.

The DO command will automatically append the extension /BLD to the filespec if no other extension is specified. To DO a file with another extension, you must specify the extension.

Example -

DO START

This will begin execution of the file START/BLD.

The logical record length of a BUILD file is 64. However, this is arbitrary. You can DO a file of any logical record length as long as it is stored on the disk in ASCII and contains strings no longer than 63 characters followed by a carraige return.

You can combine the AUTO and the DO to power up directly into your program. The format for this is:

AUTO DO filespec <ENTER>

You may execute another DO from a DO file, as long as you make it the LAST statement in the DO file. You may also abort execution of a DO by holding down the <BREAK> key.

The BUILD and DO combination is very simple to implement once you understand how the command works. What you are doing when you create a BUILD file and then execute it; is substituting a file for keyboard input. Anything that you enter with DO will be accepted and executed as if it were actual keyboard input.

This is very useful, but it also has some limitations. Such as when the program requires operator input (DISKDUMP, DISKZAP, etc.), it will go to scan the keyboard to see what you are typing and DO will be giving it input. That is why you cannot execute anything from a DO that will need

operator input to function (with the exception of FORMAT, BACKUP, and BASIC. These have been set up to check for an active DO and compensate. You may use BACKUP and FORMAT within a DO file. If your program set the date, all you would have to do is have a statement that went "BACKUP :0 :1" in your DO file and you could backup drive zero to drive one inside a DO. BASIC looks for an active DO and stops scanning the keyboard. You may now use DO from BASIC effectively to chain commands).

You can use PAUSE to halt a DO long enough to switch disks, turn on the printer or modem, or whatever is needed along those lines, but you CANNOT interrupt a DO and try to input from the keyboard.

When you specify the (HIGH) parameter, DO will first adjust high memory to the address specified, and then locate itself accordingly. This allows you not only to relocate the DO buffer, but it also provides a means of automatically protecting memory for the routines that DO will subsequently load.

EXAMPLE : DO TEST (HIGH=A000H)

This will set high memory to A000 hex, and then locate the DO starting at that address and working down. The DO buffer is 288 bytes in length (256 byte I/O buffer + a 32 byte DCB). Please do not forget to append an "H" for any hexidecimal values when specifying the (HIGH) parameter address.

DUMP

This function allows you to down load a section of memory directly to a disk as program. The format is:

DUMP filespec:d (START=aaaa,END=bbbb,TRA=cccc)

"filespec" is a standard Dosplus file specification.

"START=aaaa" is the start address of the memory block. 'aaaa' must be a value no less than 5700H. Remember to append a trailing "H" for hex values.

"END=bbbb" is the ending address of the memory block.

"TRA=cccc" is the 'transfer' address. This is the address where program execution starts when the file is loaded. If this option is omitted, the transfer address will default to the start address.

EXAMPLE -

DUMP TEST (START=7000H,END=9000H,TRA=71E5H)

This will create a file on the disk called TEST/CMD containing the program in addresses 7000H to 9000H, and when loaded, will begin execution at 71E5H.

Please note that you must append a trailing "H" to the address in each case if the address is in hex notation. Otherwise the DOS will assume a decimal address.

If no other extension was specified when the filespec was entered DOSPLUS will automatically add the extension /CMD.

To execute the program once it has been dumped to the disk, simply enter the filespec from DOS, or you can activate DEBUG, then load the program in and execute it from there.

After a file has been dumped to disk, you can use the Dosplus TAPE utility to relocate the file to wherever you want it in memory (see TAPE).

FORCE -

This function allows the user to route I/O between various devices and/or files. The format is :

```
FORCE *device TO *device
      or
FORCE *device *device
```

The parameters are:

```
DO - Display output (video monitor)
KI - Keyboard input
PR - Printer
RI - RS232 input (Model III only)
RO - RS232 output (Model III only)
```

EXAMPLE : FORCE *PR TO *DO <ENTER>

This would have forced the printer output to the display. Instead of a device specification, you have the option of specifying a filename.

EXAMPLE : FORCE *PR TO PRINTER/PRT:0

If you had not specified an extension, FORCE would have appended a /TXT (like the spooler) to the filename.

You can now route the printer to a disk file. This means that you can run reports now and list them later with the PRINT and CTL parameters (see LIST), and have the printout whenever you want it. You can force the video to a disk file, although joining it might be a more pleasant alternative (see JOIN).

Typing FORCE by itself will get you a display of the current logical devices and their driver vector addresses (which is the address of the driver itself, NOT the DCB). It will also display any forced status. The device specification will be replaced by the word "FORCE" and the device or file specification that it has been forced to.

To reset a forced device, you can either force it back to itself (selective reset) or use the "RESET" parameter on CLEAR (total reset; see CLEAR).

EXAMPLE : FORCE *PR *PR

Would set the printer back to itself.

EXAMPLE : CLEAR (RESET)

Would reset all devices in the system. Also closes all forced files and clears memory (for a more detailed explanation of the CLEAR parameters, see CLEAR).

EXAMPLE : FORCE

Would display all current driver vector addresses.

Technical note : FORCE disables the device being forced. This means that ALL I/O normally going to that device is suspended and re-routed to the new device/file. If you FORCE the screen to another device, no video output will be seen. To avoid this (if it is not desirable for the present application), use JOIN (see JOIN).

FORMS -

This function controls the printer driver and will allow you to direct the output to either the serial board or the parallel printer port as well as tell the printer the size of the paper you are using. It will allow you to use narrow paper in a wide printer without the printer trying to print past the right edge of the paper. The format is:

FORMS (par=exp,par=exp...)

"par" is the parameter to be altered. A list of parameters appears below.

"exp" is the expression which gives the altered value. If it is not within an acceptable range, the error "Specification error" will result.

The parameters for the FORMS COMMAND are:

P - Maximum Number of lines per page. Standard 11 inch paper has 66 lines per page
L - Number of print lines on a page
W - Number of characters on a line
S - Directs all output to serial printer.
LF - Auto linefeed on carriage return. Will generate a linefeed on a carriage return for printers that do not have this feature.
T - Performs a top of form on the printer.
NULL- Suppresses line feed on nul/carriage return
EP - Configures forms to allow for EPSON printer
SAVE- Writes current configuration to disk

The default values for the parameters are (for a more detailed explanation of the parameters, read on) :

P - 66 (66 lines per page)
L - 66 (print lines to be used per page)
W - 0 (infinite characters per line)
S - OFF (output to parallel printer)
LF - OFF (NO auto linefeed generated)
T - No default. If included, top of form
NULL- ON (NO line feed on nul/carriage returns)
EP - OFF (Standard TRS-80 graphic codes)
SAVE- No default. If included, writes to disk

DOSPLUS will power-up from a cold boot with the FORMS command set to these default values. Typing FORMS with no parameters will display current settings.

In order to obtain pagination (paging control), you must first change the parameters from their shipping status. For sake of compatibility with other systems, this

version of DOSPLUS comes with the forms control already defeated. This is new from versions past.

The (P=aa) parameter will set the number of lines that are on the page physically. This does not mean that the system will print on every line of it (that is controlled by the "L" parameter), it is merely informing the system how many potential lines there are. Standard 11 inch paper has 66 lines on it (6 per inch). Some paper may have more, some less. For most applications, "P" will remain 66.

The (L=bb) parameter will set the number of lines on the page that the system will actually print. In most cases 60 lines on a page will skip the perforations of standard fanfold paper nicely. If you for some reason wish to print on short sheets of normal width paper though, you would lessen the "L" parameter accordingly. When P and L are equal to each other (i.e. "P=66,L=66"), FORMS does no paging because you have in effect told it that you wish to print on every available line.

The (W=cc) parameter tells the system how many characters can fit on a line. Using normal sized (non compressed or expanded) print, 11 inch paper can hold 80 columns or characters across while 15 inch paper can hold 132. When the line reaches the limit of the "W" parameter, the DOS will cut the line off, generate a carriage return, and continue with a new line. Some varieties of compressed print are capable of printing more than 132 characters on a line. If this is true, adjust your parameters to allow for it. The default value of 0 tells the system not to do any line termination. If you output EXACTLY the number of characters specified in the "W" parameter, you may experience double carriage returns (line feeds). This is due to the fact that you generated one line feed when you finished printing the line, and FORMS generated one when the limit of the "W" parameter was reached. To solve this, increase the "W" parameter by one.

The preceding three commands were all set to equal a certain number. The following parameters must merely be included in the parameter list to be turned on, and must be set equal to the word "OFF" to be disabled.

EXAMPLE : FORMS (S,LF,NULL=OFF)

This would have enabled the serial driver and auto line feed options, and disabled the null buffer option.

The (S) parameter tells the printer driver to route all printer output to the RS232 out (*RO). This is for use with serial printers. When using this parameter, please make sure that you have consulted the manual with your printer and set the RS232 board to the proper settings (see RS232). The use of this parameter means that you no longer have to locate serial printer drivers up in high memory

where they take up space and get in the way.

The (LF) parameter is for use with printers that do not generate a line feed with a carriage return. This is rare, and when it does occur it is usually in serial printers. For example, every printer that Radio Shack sells produces a line feed when it receives a carriage return. If your printer does not have this as standard, doing something as simple as a line listing will be disastrous, with each line being printed over the top of its predecessor.

The (T) parameter, if included in the list, will cause the line printer to generate an immediate top of form on the line printer. Its primary use is for paging up on the printer after printing a word processor file and exiting, after a screen print, or upon system startup. Whenever you desire a top of form from DOS, this is the method to use.

The (NULL) parameter is used in essentially the same manner as Radio Shack's LPC/CMD. After the old LP I series printers were upstaged by the LP V and VI, the programs written for the earlier printers would not run properly. This is because the older printers, when they received a carriage return by itself (with no preceding text to be printed, i.e. typing "LPRINT" and pressing ENTER), would NOT generate a line feed. This meant that the statement LPRINT CHR$(138) would produce one line feed (for the 138). However, the new "intelligent" printers DID do a line feed when they received a carriage return on an empty buffer. Therefore the statement LPRINT CHR$(138) would produce TWO line feeds (one for the 138 and one for the LPRINT). This threw the printouts off. So they came out with LPC/CMD. And our NULL parameter does the same thing. When NULL is on (which is the standard), if the printer driver gets a carriage return without any text, it will supress sending a line feed to the printer. Programs written with earlier versions of DOSPLUS may need NULLs turned OFF to run properly. This is because NULL is now the EXACT OPPOSITE of what it did in earlier DOSPLUS'.

The (EP) parameter is used to implement the special graphics capacity of the EPSON series printers. EPSON printers have the capacity to use a special graphics set so that you have more interesting custom features. If EP is on (not the default), then all printer codes will be intercepted if they are beteween 128 and 255, and 32 (decimal) will be added to them before they are output to the printer. This is a feature unique to EPSON printers and is of absolutely NO use to Radio Shack printer owners. For a Radio Shack printer to operate correctly, EP must be OFF (the default setting). EPSON owners, consult your manual before using this parameter. This automatically offsets the characters 32 decimal (20 hex), so if you have the switch inside the printer set to do this also, it will be done twice.

The (SAVE) parameter allows you to change the default settings. Every time you type FORMS, it will display the current configuration. If this is different from the default settings (what you get on power-up), simply type FORMS (SAVE) and the DOS will write the current configuration to the disk. These will then become the new default parameters. Please do NOT have your system disk write protected when you attempt to use this parameter. If you do, this will produce a "Disk write protected". The information is being written to disk, and it can't very well do that if there is a write protect tab on the diskette.

FREE -

This function will display a free space map of a disk and tell you the number of free granules/kilobytes. The format is:

FREE :d (P) <ENTER>

":d" is the drive to be displayed.

"P" tells Dosplus to send the information to the printer. If omitted, the display will be to video only.

Example -

FREE :1

Free space for drive one.

The screen will clear and display the following:

```
          Free space drive: 1  - 24g/36k
00-06:  XXX ! XXX ! XXX ! XXX ! X.X ! ... ! ...
07-13:  ... ! ... ! ... ! XXX ! XX. ! XXX ! XXX
14-20:  XXX ! XXX ! XXX ! DDD ! XXX ! XXX ! XXX
21-27:  XXX ! XXX ! XXX ! XXX ! XXX ! XXX ! XXX
28-34:  XXX ! XXX ! XXX ! XXX ! XXX ! XXX ! XXX
35-39:  XXX ! XXX ! X.. ! ... ! .L.
```

An X indicates that a granule has been used, a period (.) indicates that it is free, a D indicates that the granule has been allocated for the directory, and an L indicates that the granule has been locked out due to a flaw when the disk was formatted.

DOSPLUS will detect and display all formatted tracks. It will handle 35 to 96 track disks, depending upon your hardware.

You will notice that the example is for a double density diskette. Single density granules are five sectors long (1280 bytes), and double density granules are six sectors long (1536 bytes). Had this been a single density disk, you would have seen only two granules on a track.

If you are using Dosplus 4.0 and you request the free space for one of the hard drives, obviously a free space map is a shade impractical. Therefore, you get a concise readout of the free space that looks like this :

Free space drive: 4 - 731g/4020k

Remember, every 1000K is 1 megabyte. That drive has a little over four MEGABYTES free on it.

JOIN -

This command allows the linking of two logical devices and/or a logical device and a file. Basic operation of the command is similar to FORCE (see FORCE), but it does not totally re-route the I/O to that particular device or file like FORCE. It merely enables you to duplicate whatever I/O would normally go to one device or file to another device or file. The format is :

```
JOIN *device TO *device
          or
JOIN *device *device
```

You may specify a filespec in place of either device specification, however, remember not to precede a filespec with an asterisk. Only device specifications are preceded by an asterisk.

Your legal devices are :

```
*KI - Keyboard input
*RI - RS232 input (Model III only)
*DO - Display output (video monitor)
*RO - RS232 output (Model III only)
*PR - Printer
```

You can specify any legal filename (see FILESPECS).

JOIN is an extremely powerful and useful command. You can join the video and the printer for hardcopy of what goes to your screen. You can join the video and a diskfile for a visual record of what was typed into the machine (be certain to list it back with the "CTL" option; see LIST).

EXAMPLE : JOIN *DO VIDREC/SCR:0

This would duplicate every byte of information that goes to the screen and put it in a file called "VIDREC/ASC". This enables you to do much. For example, if you had a very difficult or critical stage of a BASIC program that the user was prone to destroying, you could CMD"JOIN *DO VIDREC/ASC:0" just before you get there, and then if the user types "NEW" instead of "NO", you would be able to list the file back and see EXACTLY where the error was. And if you list it back using the (CTL) parameter, you will see it EXACTLY as it appeared to him. This is invaluable in debugging. Some other potential applications are :

EXAMPLE : JOIN *PR TO *DO <ENTER>

This would have joined the printer output to the display. Instead of a device specification, you now have the option of specifying a filename.

EXAMPLE : JOIN *PR TO PRINTER/PRT:0

If you had not specified an extension, JOIN would have appended a /TXT (like the spooler) to the filename.

You can now link the printer to a disk file. This means that you can run reports now and list them again later with the PRINT and CTL parameters (see LIST), and have the printout whenever you want it.

Typing JOIN by itself will get you a display of the current logical devices and their driver vector addresses (which is the address of the driver itself, NOT the DCB). It will also display any joined status. The device specification will be followed by the word "JOIN" and the device or file specification that it has been joined to.

To reset a joined device, you can either join it back to itself (selective reset) or use the "RESET" parameter on CLEAR (total reset; see CLEAR).

EXAMPLE : JOIN *PR *PR

Would set the printer back to itself.

EXAMPLE : CLEAR (RESET)

Would reset all devices in the system. Also closes all joined files and clears memory (for a more detailed explanation of the CLEAR parameters, see CLEAR).

KILL -

This feature will allow you to delete a file from the disk directory and de-allocate the space. The format is:

KILL filespec <ENTER>

If the file is password protected, you must use the password to KILL the file. To remove a password protected file that you do not want (such as a DOS utility like DISKDUMP/CMD), simply use PURGE (see UTILITIES).

Example -

KILL TESTFIL/CMD:2

This will kill the file TESTFIL/CMD on drive two.

Specifying a drive number kill cause KILL to search ONLY the specified drive for the file to be killed.

If the file is on a drive other than zero, simply typing KILL followed by the filename will cause the KILL command to do a global search of all available drives until it locates a program by that name. However, this has it's drawbacks. If you wish to kill a file named TEST on drive one, and a file by the same name exsists on zero, you MUST specify the drive number. If you do not, KILL will encounter the file on drive zero FIRST, and kill that one. Then not only will you have killed the wrong file, but the file that you wanted to kill in the first place is still there.

You will note that you only delete the file from the directory, and do not actually zero the entry. So, if you kill a file accidentally, you can recover the file using the RESTORE utility (see UTILITIES). If you have written to the disk since you killed the file, and the area formerly used by that file has been overwritten by new data, then the RESTORE command may not work. However, if the disk space has NOT been overwritten, and the file entry has NOT been physically zeroed from the directory, then RESTORE will be able to re-build the entry and re-allocate the disk space, thus restoring the file exactly as it was.

LIB

This function will list on the screen all of the library commands. The format is:

LIB <ENTER>

There are no parameters for this function.

A library command is defined as something that is a part of the actual DOSPLUS system (i.e. contained within a system file). These commands can all be used from BASIC and have your control returned to BASIC. Library commands are not the same as system utilities. A library command will not be lost unless you purge a file with the "/SYS" extension.

What this means to you is that should you want a disk with the ABSOLUTE maximum space on it that a "full system" systems disk can have, you may use PURGE to remove any file with the "/CMD" extension. And you can do this without fear of losing any function listed in this library of commands.

LIST -

This function will list the contents of a file on the screen or to a printer. The file is displayed in ASCII code with the unprintable characters replaced by a period (.) unless otherwise designated. The format for this command is :

LIST filespec (parameter) <ENTER>

The parameters you have a choice of are as follows.

CTL - Output control codes
PRINT - Send output to printer

When you include the (CTL) parameter, the file listed will not have the control codes translated into periods. This will, under most circumstances, produce some rather wild results. If listed to the screen, things will start to happen based on what the contents of the file do when they are interpreted as video control codes.

Where the (CTL) parameter is vital is in listing a file that is the result of a FORCE or JOIN of the screen. This file will contain all the data that would have gone to the screen, including the control codes for clear screen, etc., etc. When you list this file back, if you want to see the file EXACTLY as it appeared on the screen, you must use the (CTL) parameter.

Including the (PRINT) parameter will send all the output to the printer. Once again, using the (CTL) parameter too quickly could cause problems. The control codes sent would be interpreted as printer control codes, and could produce extremely erratic behavior. "PRINT" may be abbreviated as "P".

Where the (PRINT,CTL) combination is needed is if you have created a disk file or printed output by forcing or joining your printer to a diskfile, and then you want to print this file. When you want the printout to be formatted nicely, including the "CTL" with the "PRINT" will cause the DOS to output the printer control codes (needed for paging) along with the text.

Under most circumstances, you will find that the ASCII list of DOSPLUS is a great advantage when listing files.

EXAMPLE : LIST filespec <ENTER>

This enables you to see a orderly printout of the file on the screen instead of a wild, unintelligible dump of data to the screen. By including the (CTL) parameter, we have not taken away the old style of listing, we have merely expanded your options.

LOAD -

This feature loads a machine-language program or data from disk into memory and returns control to either Dosplus or BASIC. Its primary use is to LOAD a file created by the DUMP command. The format is:

LOAD filespec:d <ENTER>

"filespec:d" is a standard Dosplus file specification.

If the file is password protected, you will have to enter the password before you can load the file.

Example -

LOAD RSTERM/CMD:2

This will load the file RSTERM/CMD from drive 2.

Using CMD"LOAD filespec" from Extended Disk BASIC is the equivalent of CMD"L" in TRSDOS Disk BASIC.

When a program is loaded in through LOAD, execution must be intialized through DEBUG or some other means. CPU control will be returned to the DOS rather than sent to the program.

PAUSE -

This function stops execution of a program awaiting operator action. The format is:

PAUSE message

"message" is the message to be displayed on the screen during execution of the pause. If omitted, PAUSE will be displayed by itself.

This command was designed primarily to halt execution of a DO file so that Dosplus can display a prompt or reminder to the operator.

To continue after the PAUSE, press the <ENTER> key.

Example -

```
PAUSE Insert PAYROLL data disk.
```

Because Dosplus allows you to preface a DOS command line with a period (.), making it a comment, you can display several lines of text before entering the PAUSE command.

Example -

```
.Locate disk number 21
.Place it in drive one
PAUSE Press ENTER to continue...
```

This will display the first two lines but TAKE NO ACTION, and then pause when it encounters the third line.

Remember, when the first character of Dosplus command line is a period (.), Dosplus will treat the remainder of that line as a remark and will NOT execute it.

When PAUSE continues, it will proceed to the next statement. It will NOT attempt to execute the trailing message. PAUSE also continues only when the ENTER key is pressed, any other key is ignored.

PROT -

This feature will allow you to alter the disk's protection status using the disk master password. With this, you can lock/unlock all the files on a disk, change the master password (provided of course, you know the old master password), rename the disk itself, or clean the directory of deleted entries. The master password was set when the disk was created, using the FORMAT or BACKUP commands.

The master password for the DOSPLUS disk is not set, pressing ENTER will suffice.

The format is:

PROT :d (option) <ENTER>

The options are:

NAME - Assign new disk name.
LOCK - Protect all visible user files by assigning the disk master password as the access and update password.
UNLOCK - Remove password protection from all visible user files. Will not un-protect system files.
PW - Change the master password.
CLEAN - Zero deleted directory entries.

DOSPLUS will then ask you for the current master password and then prompt you to input the necessary data.

EXAMPLE :

```
PROT :0 (NAME) <ENTER>
Master password ? <ENTER>
New diskette name ? GEOFUN <ENTER>
```

The options can be used in any combination.

Example -

```
PROT :1 (NAME,PW,LOCK,CLEAN)
Master Password ?
New diskette name ? TESTDISK
New Master password ? TESTTEST
```

This would have renamed the disk TESTDISK, assigned it the new master password TESTTEST, assigned all visible user files on the disk the password TESTTEST for both the access AND update passwords, and then would have zeroed all deleted entries from the directory.

When a directory is CLEANed, you can no longer restore any of the dead files on the disk with the RESTORE utility. The entries actually physically disappear.

RENAME -

This function will allow you to assign a new name or extension to a disk file. Only the name and extension of the file are affected by RENAME. File content and protection status remain unchanged. If the file is password protected, RENAME will require the password. The format is:

RENAME filespec TO filespec <ENTER>
or
RENAME filespec filespec <ENTER>

RENAME can not be used to change the password. If the new filespec already exists, DOSPLUS will print an error message on the screen and return you to the DOSPLUS prompt.

Examples -

RENAME PAYROLL:0 PAYROLL/BAS

Tells Dosplus to add the extension /BAS to the file PAYROLL.

RENAME TEST/BAS:2 TEST1/BAS

Changes only the filename of the program on drive two.

RENAME FILENAM1 FILENAM2

Searches for FILENAM1, and when it locates it, renames it FILENAM2.

You cannot use RENAME to change a file's protection level, use ATTRIB for that.

RS232 -

This command will allow you to display or alter the settings on the RS232 serial interface board. The format is :

RS232 (par=exp,par=exp,parl=expl,parl=expl...)

"par" is a parameter that you will set equal to a specific value. BAUD is an example of this. You will set BAUD equal to a specific rate.

"exp" is the value that you are adjusting "par" to.

"parl" is the second type of RS232 parameter. These parameters are set equal to either "ON" or "OFF". There is no specific value. When displayed, if they are active, they will be shown. It is not neccessary that we say either "ON" or "OFF". If they are there, they are on. PARITY is an example of this.

"expl" is the second type of expression. It is either "ON" or "OFF".

Briefly, the parameters are :

BAUD (set/display BAUD rate)
WORD (set/display WORD length)
STOPS (set/display number of STOP bits)
PARITY (enable/disable parity error detection))
EVEN or ODD (set parity mode)
WAIT (set WAIT mode - Model III only)
BREAK (disable serial communication)
SAVE (permanent configuration - alters defaults)
DTR (Data set signal)
CTS (Printer handshaking signal)
DSR (Printer handshaking signal)
CD (Printer handshaking signal)
RI (Printer handshaking signal)

For an explanation of these parameters, read on.

You have the following as your FIRST type parameters (those that are set equal to a specific value) :

BAUD is the rate of transfer allowed by the RS232 board. Allowable speeds are : 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, or 19200.

Most modems are not capable of speeds faster than 300 BAUD. This is the default value. If it is omitted from the command line, it will be unchanged from its current value. Please be careful when attempting to exceed 300 BAUD as the data transfer is not always reliable at speeds higher than that when you are using standard telephone equipment and the average modem.

Example -

```
RS232 (BAUD=110)
```

WORD is the number of bits per byte. This can be either 5, 6, 7, or 8. Set this according to your particular needs. The default value is 7. This is sufficient for transmitting letters. For sending graphics, you may need to change this to 8. If omitted, it will remain unchanged. If the value for WORD is different between two systems attempting to communicate, unpredictable results are guaranteed. It is not so much important WHAT you set WORD equal to as it is that both machines be set the same.

Example -

```
RS232 (WORD=8)
```

STOPS is the number of stop bits. Can be either 1 or 2. Set it according to your needs. It affects the length of the terminating bits sent after each character transmitted. Default value is 1. If omitted, it will be unchanged. Again, it is not so much important WHAT it is set to, as much as it is important that both communicating machines be set the same.

Example -

```
RS232 (STOPS=2)
```

The following parameters are of the SECOND variety, that is, they are set equal to either the word "ON" or "OFF". "ON" is the default value. To turn one of these on, all you must do is mention it in the command line. However, in order to turn it off, you must specify "OFF".

PARITY indicates that parity is enabled. To disable parity, set "PARITY=OFF". Default value is inhibited (OFF). If it appears in the display line, it is enabled. If it does not, it is inhibited. Having your system set for no parity (PARITY=OFF - the default) disables parity error detection.

Example -

```
RS232 (PARITY=ON) or RS232 (PARITY=OFF)
```

EVEN or ODD. If parity is enabled (ON), and you see the word PARITY in the display line, you will also see either the word 'EVEN' or the word 'ODD'. This indicates whether the parity is even or odd. This should be set according to your needs. The default is ODD. Once again, it is not so much important as to whether or not you use even or odd parity as it is that you use whatever parity the guy at the other end of the phone is set for. To set even or odd, simply use the word. You don't set these equal to anything. If your system is set for no parity, this disables the parity error detection.

Example -

RS232 (EVEN) or RS232 (ODD)

WAIT tells the RS232 driver to wait for a completion of character I/O before returning. If transmitting, it will not return without sending a character. If receiving, it will not come back from the RS232 without a character. WAIT will default to ON. If it is omitted from the command line, its current status will be left unchanged.

Example -

RS232 (WAIT=ON) or RS232 (WAIT=OFF)

BREAK interrupts all RS232 operations. As long as BREAK is on, the RS232 will not receive or transmit characters. BREAK defaults to off. If omitted from the command line, its current status will be left unaltered.

Example -

RS232 (BREAK=ON) or RS232 (BREAK=OFF)

Typing "RS232 <enter>" with no parameters causes the computer to display the current settings. For example, if you examined the default settings (as we shipped the disk) you would see this :

RS232 <enter>

BAUD=300,WORD=7,STOPS=1,DTR,DSR,WAIT

These are the default parameters, and unless you change them, they will remain in effect. To make a permanent change in the parameters (i.e. alter default settings), after you get the parameters set like you want them, type :

RS232 (SAVE)

This will cause the default parameters to be written to the disk. A word of caution; do NOT attempt this if the

diskette is write protected.

These last parameters are the "handshaking" parameters. The first, DTR, is used to indicate that your terminal is ready. The remaining (CTS, DSR, CD, and RI) are used for handshaking by our serial printer driver. They are set equal to either "ON" or "OFF".

DTR (Data Terminal Ready). This signal is used by the "Data set", i.e. your modem. It is a logic signal, not transmitted data. For the most part, you should leave it on for compatibility's sake. The default value is on.

Example -

RS232 (DTR=ON) or RS232 (DTR=OFF)

CTS (Clear To Send). This signal is used by our serial printer driver. If enabled, it will handshake to the printer on pin 5. If your serial printer handshakes on pin 5 of the RS232 cable, then this is the parameter you should enable.

Example -

RS232 (CTS=ON) or RS232 (CTS=OFF)

DSR (Data Set Ready). This signal is also used by the serial printer driver for handshaking. When enabled, it will handshake on pin 6. If your serial printer handshakes on pin 6, then this is the parameter for you. Your Dosplus 3.4/4.0 comes with DSR on.

Example -

RS232 (DSR=ON) or RS232 (DSR=OFF)

CD (Carrier Detect). This is another signal used by our serial driver. If this is enabled, it will handshake on pin 8. If your serial printer handshakes on pin 8, then set this parameter.

Example -

RS232 (CD=ON) or RS232 (CD=OFF)

RI (Ring Indicator). This is still another signal that you can use for handshaking with a serial printer. When enabled, it will handshake on pin 22. If your serial printer uses pin 22 for handshaking, then this is the parameter for you to set.

Example -

RS232 (RI=ON) or RS232 (RI=OFF)

Removing handshaking -

You can disable any serial printer handshaking by simply setting all four above mentioned parameters to "OFF". Since only DSR is set when you receive the disk, all you would have to do then is set DSR equal to off. Simply make sure that none of the names (CTS, DSR, CD, or RI) appear in the display line (i.e. RS232 <enter>).

Technical note : Without handshaking, Dosplus has no way of knowing whether or not your printer is ready to receive data. If your printer is not ready, the DOS will continue to send data. This can cause loss of characters. If this is happening, try slowing the BAUD rate down.

A final word -

You can combine any of the parameters given above within one command line as long as they are each individually legal and seperated inside the parenthesis by a comma only!

Example -

```
RS232 (BAUD=600,WORD=8,DSR=OFF,CD=ON)
```

This would increase the BAUD rate to 600, set WORD length to 8 bits, disable serial handshaking on pin 6 and enable it on pin 8.

If these were the parameters that you would be using 99 percent of the time, you should then make them a permanent change. You would type :

```
RS232 (SAVE)
```

And it would write the current settings to disk as the new default parameters.

TIME -

This feature allows you to display and set the time. The time is set to 00:00:00 when the system is powered up or on a cold boot. This is how you set the time for the CLOCK command. The format is:

```
TIME  <ENTER> (to display the time)
          or
TIME hh:mm:ss <ENTER> (to set the time)
```

All DOSPLUS commands and utilities (except DISKZAP) will re-enter the system without your having to re-set the DATE and TIME.

It should be remembered that the internal clock is a twenty-four hour clock, and will be displayed as such. For example, after it runs past 12:59:59, 1:00:00 P.M. will be displayed as 13:00:00.

No internal error checking is done. If you input characters that are beyond the valid number range (i.e. 99:77:88), these will be accepted and incremented as normal. This will produce some rather strange results.

In the Model III, if the clock is allowed to run past 23:59:59, it will reset to 00:00:00 and the date will be incremented. In the Model I, the date will simply reset itself to 00:00:00 and the date will be unaffected.

VERIFY -

This function tells DOSPLUS to do an automatic read after each write to the disk. The verify function is recommended when you are saving programs and data of great importance. The format is:

VERIFY parameter <ENTER>

The parameters for the VERIFY command are ON and OFF only. If no parameter is specified, DOSPLUS will assume ON. On system power-up, the VERIFY command is OFF unless you turn it ON.

If verify encounters a CRC error (or a read error of ANY type), it will abort with the proper message. A CRC error occurs when the CRC value the DOS gets by reading back the data does not match the value for the data it thought it just wrote.

A CRC (cyclic redundacy check) is calculated by running all 256 bytes of information in a sector through an algorithm which produces a two byte value for that 256 bytes. This value is stored with the sector ID. If the value calculated on a read does match the stored value, then an error has occured. Due to the way the CRC is calculated, it gives you an exact check of your data. If even ONE BIT has changed, the CRCs will no longer match, and an error will result.

UTILITIES

The following utilities are included with DOSPLUS. They are executed from the DOSPLUS command mode.

BACKUP	- Duplicate a disk
COPY1	- Single drive copy
CONVERT	- Convert Model III TRSDOS or single density
CRUNCH	- BASIC program compression utility
DISKDUMP	- Display/Modify a disk file sector
DISKZAP	- Floppy disk editor
FORMAT	- Formats a disk and writes system data
MAP	- Map out diskette, showing file locations
PURGE	- Deletes all unwanted files from a disk
RESTORE	- Recovers inadvertently deleted files
SPOOL	- Printer spooler
SYSGEN	- Create non-standard system diskette
TAPE	- Tape/Disk - Disk/Tape utility (relocator)
TRANSFER	- Copies all visible files to another disk

Dosplus 4.0 includes these -

HCOPY	- Hard disk file offload/reload program
HFORMAT	- Hard disk format utility
HZAP	- Hard disk editor

HCOPY has its own section and the other two utilities are dealt with in the same section as FORMAT and DISKZAP.

BACKUP -

This program will make an exact duplicate of a disk. It will work with either a single or multiple drive system. The format is:

BACKUP <ENTER>

DOSPLUS will then prompt for additional input from the keyboard:

Source drive number ?
Destination drive number ?
Backup date (MM/DD/YY) ?

DOSPLUS will also allow the following formats:

BACKUP :d TO :d <ENTER>
or
BACKUP :d :d <ENTER>

DOSPLUS will then skip the first two questions. If the date has been set using the DATE command, it will also skip the date question. If the date has not been set, the DOS will prompt you for the date, and pressing ENTER will default to 00/00/00.

If you specify the same drive number for both the source and destination drive, DOSPLUS will automatically do a single drive backup. You will be prompted to insert the source, destination, and system disks, as required. Be sure that you insert the disk it is prompting you for. If it gets the wrong diskette at the wrong time, it could fail to make a proper backup.

If the DESTINATION disk is ALREADY formatted to a GREATER number of cylinders than the SOURCE disk, BACKUP will backup the source disk but leave you the extra cylinders. Which means you can backup a 35 cylinder disk to a 40 cylinder disk without it re-formatting and losing the last five cylinders.

Backup will only re-format the disk under very specific circumstances. They are :

1. Blank (unformatted) diskette
2. Unable to read Directory
3. Source cylinder count higher than destination
4. Densities differ
5. Operator intervention

If the diskette contains data, DOSPLUS will display the message "Diskette contains data, use or not ?".

Answer this with either Y (Yes), or U (Use), or F

(Format). If you enter the Y or U, it will seek to copy the disk contents without first re-formatting the destination disk.

If you answer "F", it will reformat the destination disk first. This is recommended in instances where you are having "Destination disk read errors" that are preventing you backing up onto a previously formatted disk.

PLEASE, do NOT switch the source disk after the question "Diskette contains data, use or not ? " appears. The GAT (Granule Allocation Table) for it has been read at this point. To change disks would cause a seemingly good backup to have an invalid GAT on it. This WILL cause unreliable system operation further on. You may switch the destination disk if you wish. It is not, however, a practice recommended by us. It is our recommendation that if you are not going to backup to and from the two disks CURRENTLY in the machine, you load BACKUP into memory without answering the source and destination drive number questions, remove the disks and set the machine up for the backup. THEN answer the source and destination drive questions (once you have the disks you are going to backup to and from in the machine).

Backup will ONLY back up allocated granules. This will make the speed of the backup seem irregular. Do not be alarmed. It merely saves time to only read, write, and verify what you have to. Why copy the entire cylinder when only the center granule is allocated. This is especially true for double headed double density drives which could have six sectors on a cylinder.

If the directory cylinder of the source disk is on a different cylinder than it is on the destination disk or the cylinder zero (bootstrap) densities differ, the Dosplus system will re-format THAT CYLINDER ONLY; in order to move or correct it. Remember, it will seek to create a "mirror image" of the diskette. If for any reason a granule is locked out on the destination disk that is allocated on the source disk, the BACKUP program will abort with the message "BACKUP rejected, Too many FLAWS!". At that point you may attempt to re-format the disk and try again, if you like.

When verifying a read during format, Dosplus will retry 5 times before locking out the cylinder. It will home the head between tries. It will try to read the source disk during backup 5 times before aborting. It will only verify the write to the destination disk ONCE before aborting. Its main goal is to produce you a good backup. If you cannot verify a write immediately after writing it, a problem exsists and you should be aware of it.

Please also be aware that with the new "single volume" approach to double headed disks, you can no longer make a normal backup between a single headed disk and double

headed one. If you wish to backup a double headed disk, you either need another double headed disk drive or you must use a single drive backup.

You may use a double headed drive as a single headed unit for the purpose of backing up a single headed disk from another drive. However, before you do this, you must re-configure the double headed drive as if it were a single headed unit (see CONFIG).

Backup is an extremely intelligent program and under most circumstances requires minimal operator input. However, if you that the diskette in the destination drive is going to be unreadable to that drive (i.e. an 80 track disk in a 40 track drive), we suggest that you do utilize the "F" (re-format) option at the "Diskette contains data, use or not ?" prompt. Otherwise the system will re-try multiple times before re-formatting the unreadable disk on its own.

To "backup" a hard disk drive, you must use TRANSFER of COPY to remove any files that will fit on a single disk. For files that are too large for a single disk, there is a utility called HCOPY/CMD that will perform this function.

COPY1 -

This is a single drive copy utility. It is to be used by single drive users to copy a file from one disk to another. The format is:

COPY1 filespec <ENTER>

"filespec" is a standard Dosplus file specification.

If the program is password protected, you will be required to use the password before you can access the file.

DOSPLUS will prompt you when to insert the source, destination, and system disks into the drive. You can use COPY1 to copy a file from a non-system disk to another non-system disk by following the on-screen prompts.

It will also serve to move programs between single and double density on single drive systems.

To copy a file on a drive other than 0, specify the drive number after the filespec.

If a file is too long to be read into memory at one time, the message "File too long" will appear and the program will return to to DOS. The user must then use a multi-drive copy to copy the file.

CONVERT -

This is a dual function utility. First, it will read a file from a diskette formatted by Model III TRSDOS to a diskette formatted by Dosplus. Second, it will convert a Model I single density disk to Model III single density format (Model III Dosplus) or Model III single density to Model I (Model I Dosplus). This allows transfer of programs between the Model I and III without a double density adapter for the Model I.

First mode -

CONVERT filename :sd :dd (V13)

"filename" is the wild card mask that you can use to limit yourself to a particular file or type of file.

":sd" is the source drive number.

":dd" is then destination drive number.

"(V13)" is the optional parameter that indicates to Dosplus that you are converting from a disk formatted by TRSDOS ver 1.3. If you are converting from TRSDOS 1.1 or 1.2, omit this.

The wildcard filename option allows you to convert a single file from a Model III TRSDOS disk. Or you can convert all the "/CMD" files. Or all the "/BAS" files. Or whatever, it is very flexible. For example, let's say you want to convert Scripsit to Dosplus. First, you would boot up Dosplus. Then place the diskette containing Scripsit in drive one. Just to be certain that the disk IS readable to the drive, do a "CAT :1 (T)". Besides, this lets you make sure that Scripsit is on that disk. The file catalog tells you that the file "SCRIPSIT/CMD" does indeed exsist on the disk. At this point, you type :

CONVERT SCRIPSIT/CMD :1 :0
or
CONVERT SCRIPSIT/CMD :1 :0 (V13)

Depending on which version of TRSDOS the disk was made under. Because the only file that matches the wildcard is "SCRIPSIT/CMD", it is the only file converted. This is as picky or as accepting as you make it. For example, if you were to :

CONVERT :1 :0 (V13)

It will read EVERYTHING on drive one and put it on drive zero. If you :

```
CONVERT /CMD :2 :1 (V13)
```

It would read all the files with the "/CMD" extension off drive two (which was created under TRSDOS 1.3) and place them on drive one. You could even go so far as :

```
CONVERT ??B/?XT :3 :0
```

It would then scan drive three (which contains a disk made under TRSDOS 1.1 or 1.2) and read any files that have a "B" in the third character of the filename and have an extension ending in "XT". This illustrates that a question mark may be used in place of a character to indicate that you don't care what appears there. A question mark matches ANYTHING.

Remember, if you omit the wildcard, it will convert the entire disk. If you specify an exact filename, it will convert that file. If you give it a piece of a filename, it will convert whatever matches that piece. And a question mark will match anything.

Two things to note. First, convert in this mode requires two drives. Single drive user's do NOT have the option of reading Model III TRSDOS disks. Second, if a file already exsists on the destination disk, convert will ask "Overwrite ?". At this point, you may :

* Press enter, skipping the file.
* Enter "N", skipping the file.
* Enter "Y", copying over the exsisting file.
* Press break and abort the copy.

Technical note : When convert moves a file from Model III TRSDOS to Dosplus, all passwords are removed. The file is treated as a standard non-protected entry. Invisible files will be converted as will programs containing system attributes. However, they will convert as standard "/CMD" files with no protection. The actual Model III TRSDOS system programs do not appear in the directory so they will no be copied. Do NOT overlay TRSDOS' BASIC/CMD and CONVERT/CMD on top of the correspondingly named Dosplus utilities.

Second mode -

```
CONVERT :td
```

":td" is the target drive number. This can be drive zero as this mode of convert will work with a single drive.

This will take the single density disk in the target

drive and alter it to Model III format (Model III Dosplus) or Model I format (Model I Dosplus). Model III Dosplus can only read Model III single density. Model I Dosplus can read either Model I OR Model III single density.

The only Model I operating system that cannot read Model III single density is TRSDOS 2.3. That (TRSDOS' inability to read Model III single density) is really the only reason for this facet of convert on the Model I Dosplus. It (Model I Dosplus) itself can read the disk fine. But TRSDOS cannot. That means if it was a TRSDOS system diskette, it can no longer even boot up. Therefore, you must convert it BACK to Model I format for it to resume normal operation. The same is true for a data diskette that you hope to have TRSDOS 2.3 read. It too would have to be converted back to Model I format.

Lets say you wanted to read over an old diskette from the Model I that had lots of games on it. First you would boot up Dosplus. Then you would place the diskette to be altered in drive one and type :

```
CONVERT :1
```

The drive would run for a few seconds and the DOS PLUS prompt will be re-displayed, signalling a completed conversion. At that point the diskette is 100% readable to the Model III. It is STILL single density, it is just readable to the Model III. You may leave the disk single density if you wish or you can move the files to a double density disk. You may move the files via a standard COPY or with TRANSFER.

Note : Every time the diskette is written to in the Model I, it must be re-converted. When the Model I writes to the disk, it alters it back to Model I format.

You can use single density as the medium of transfer between the two machines. The two single densities are not quite the same, and the Model I can read either where the Model III can only read its own.

General information -

If the Model I is equipped with a double density adapter in the expansion interface and is running Model I Dosplus double density, this disks created are 100% compatible with the Model III. Although our Model III DOS won't boot in a Model I and vice versa, the double density formats are identical. The ONLY difference is that a Model I system disk will have a single density cylinder zero.

Model III Dosplus system disks function as Model I double density data disks and conversely, Model I double density system disks function as Model III data disks.

BUT, the disks can be walked back and forth all day long. No conversions or glitches. This is true Model I/Model III compatibility.

To take a file back to Model III TRSDOS (any version), do the following :

1. Format a 35 cylinder, single density disk.
2. Copy or save the program to it.
3. Boot up TRSDOS, and use their CONVERT.

This will read the file over onto Model III TRSDOS. Please be certain the disk is 35 cylinder, AND single density (FORMAT gives you the options of specifying number of cylinders and density.).

If the file is too big for a 35 cylinder, single density disk, then you cannot convert it back to Model III TRSDOS.

CRUNCH -

This is a utility that will remove unnecessary blank spaces and remarks from a BASIC program. This function is executed from the DOSPLUS command mode or from BASIC. The format is:

```
CRUNCH filespec:d TO filespec:d  <ENTER>
                or
CRUNCH filespec:d filespec:d  <ENTER>
```

The program will read the BASIC program file and write it back to the disk. The source and destination filespecs MUST be the same if the file is to be written back on top of the source.

Example -

```
CRUNCH TSTPGM:0 TSTPGM:0
```

This will compress the file TSTPGM, and write the compressed file back on top of the original.

Also, unlike other compression programs, CRUNCH will not remove blank spaces from DATA statements or those enclosed within quotation marks.

Example -

```
CRUNCH TEST1/BAS:1 TO TEST1/COM:0
```

This will compress the BASIC file TEST1/BAS on drive 1 and create a new file named TEST1/COM on drive 0.

In some programs there are REMARK lines that are referenced by GOTO or GOSUB statements. When a program has been CRUNCHED these lines will be deleted, causing the program to crash. To avoid this, type (LINE) after the CRUNCH command line. CRUNCH will then delete the REMARK but NOT the line number of the REMARK statement.

Example -

```
CRUNCH TEST1/BAS:1 TO TEST1/BAS:0 (LINE)
```

The program :

```
10 'This is a remark
20 PRINT"This is a program statement"
30 GOTO 10
```

Would now be :

```
10
20 PRINT"This is a program statement"
30 GOTO 10
```

and would still run.

DISKDUMP -

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

DISKDUMP filespec <ENTER>

If you didn't enter a filespec when you executed DISKDUMP, DOSPLUS will ask:

File :

Enter the file name and include all extensions and passwords, if any. Once you have displayed the first sector of a file, you have the following options :

; - Advance one sector
- - Go back one sector
+ - Advance to end of file
= - Go to beginning of file
G - Go to specified sector
M - Enter modify mode

Note : When typing the letters "G" and "M", you must be using capital letters. Non-capital letters will be ignored.

A sector display will look like this :

```
000000: FF7B 6F0A 003A 93FB 2020 2020 2050 6174 .{o..:..     Pat
000010: 6368 2070 726F 6772 616D 2066 6F72 2044 ch program for D
000020: 6F73 706C 7573 204D 6F64 656C 2049 4949 osplus Model III
000030: 009C 6F14 003A 93FB 2020 2020 2050 726F ..o..:..     Pro
000040: 6772 616D 6D65 7220 3A20 4D52 4C2F 4D53 grammer : MRL/MS
000050: 5300 3F70 1E00 3A93 FB20 2020 2020 506C S.?p..:..     Pl
000060: 6561 7365 206D 616B 6520 6365 7274 6169 ease make certai
000070: 6E20 7468 6973 2069 7320 7573 6564 206F n this is used o
000080: 6E20 7468 6520 7072 6F70 6572 206D 6163 n the proper mac
000090: 6869 6E65 0A09 0909 2020 2020 2020 2020 hine....
0000A0: 284D 6F64 656C 2049 202D 204D 6F64 656C (Model I - Model
0000B0: 2049 4949 292E 2020 416C 736F 206D 616B  III).  Also mak
0000C0: 6520 6365 7274 6169 6E20 7468 6174 2074 e certain that t
0000D0: 6865 0A20 2020 2020 2020 2076 6572 7369 he.         versi
0000E0: 6F6E 206E 756D 6265 7273 206D 6174 6368 on numbers match
0000F0: 202E 2E2E 00A3 7028 003A 93FB 2020 2020  .....p(.:..
```

The left hand side of the screen has six digits that indicate what you are looking at.

The first digit is the drive number.

The next three digits are the sector number.

The last two digits are the beginning byte number for each row of data. For example, if you wanted to find byte "E4", you would find byte "E0" in the left hand column and count over to

byte "E4". Remember, we are dealing with hex notation. You will count from 0-9 and then go A-F before starting over at 0.

To go to a specific sector, type :

G num <enter>

If "num" is not a valid sector for that file, it will ignore you, otherwise it will go to the specified sector. Please note that there is no space between the "G" and the number of the desired sector. You must type a "G" followed immediately by the desired sector number.

Example -

DISKDUMP PATCH/BAS

the display would look like the example.

G 23

it will ignore this because Patch/Bas doesn't HAVE a sector 23 hex.

G 3

it will advance to sector three.

Modify mode -

Once you are in the modify mode, a cursor will be displayed in the upper left hand corner of the screen. You may then :

* Type in valid hex characters
* Move the cursor with the arrows
* Enter ASCII mode by pressing CLEAR
* Zero from cursor on by pressing "@"

You may enter the ASCII modify mode by pressing CLEAR. The cursor will move into the ASCII area. At that point, you may :

* Type in ASCII data
* Move the cursor with the arrows
* Enter HEX mode by pressing CLEAR
* Zero from cursor on by pressing "@"

Once you have completed your modifications, you can :

* Exit saving changes by pressing ENTER
* Exit aborting changes by pressing BREAK

This will return you to the display mode. Once there, pressing BREAK will return you to the "File :" prompt. Pressing BREAK there will return you to Dosplus.

Technical note : When in the ASCII mode, you can enter all ASCII characters EXCEPT : Control codes, any of the arrows, or a "@". These all must be imbedded from the hex if desired. Consult your Level II manual for the proper codes. Also, it you exit the modify mode back to the display mode and then you re-enter the modify mode without exiting Diskdump, you will be in the same mode that you last left (ASCII/HEX). Don't be startled, this is normal.

To obtain a printout of the sector you are now viewing, use the screen printer (see BUILT IN FEATURES).

DISKZAP -

INTRODUCTION

DISKZAP is a powerful disk sector editor. You should use extreme caution while using DISKZAP because it can destroy a disk in a matter of seconds if not used correctly. Micro-Systems Software, Inc. assumes no responsibility for damages that may occour due to the use or misuse of this utility. HZAP users, proceed to the end of this section and read te information on HZAP.

THINGS TO REMEMBER:

1. <BREAK> is the universal abort in DISKZAP. It will halt whatever operation is taking place at that moment and return you to the DISKZAP MENU. If you are in the Modify mode, you will exit that mode and return to the MENU without writing the changes to the disk.

2. DISKZAP is written in machine-language. This means that it will acccess a disk faster than one written in BASIC. But, it also means that mistakes happen faster, too. Remember, DISKZAP doesn't destroy disks, people do. So use caution when making entries from the keyboard.

3. DISKZAP will access any sector of a disk regardless of protection status. It will disregard passwords and protection levels. This is both good and bad. There are few programs that DISKZAP can not look at, modify, or copy. But, it is also easy to change a system file by accident.

4. There are two types of data address markers. They are:

* Data mark (type 0)
* Deleted data mark (type 1, used to indicate directory in Dosplus)

Data address marks are stored as part of the sector ID. They are not visible to, nor alterable by, the user. The sole reason for even mentioning them is so that you are aware when Diskzap pauses for a "Type 1 Data address mark", it is simply telling you that it has encountered the directory.

OPERATION

The format for entry into DISKZAP is "DISKZAP <enter>".

You will then see the question "How many tracks ?". This is the number of tracks that you specified when you formatted the disk. If you press <ENTER>, DOSPLUS will assume 35 tracks. To reset the track count, press <BREAK> and you will be returned to the "How many tracks ?" prompt. After you have made your track count entry, the screen will clear and display:

```
Model III Diskzap utility - Ver 3.1
Copyright (c) 1981, Micro-Systems Software

*   Mode
    Zero
    Copy
    Print
    Verify
    Format
    Display
```

By pressing the up and down arrows keys, you can move the pointer until it points at the option you desire. Then press <ENTER> to engage the option indicated. Pressing <BREAK> will return you to the MENU.

MODE

Mode has to do with the diskette that you are currently addressing. It is concerned with three areas :

1. Density (Double or single)
2. Size (Five or eight inch disk)
3. Starting sector number (number of first sector on each track. TRSDOS uses 1-18. Most other systems, including Dosplus are 0-17)

DISKZAP powers-up in the double density mode. However, when you select this option from the MENU it will ask "Single or double", "Five or eight", and then "Sector offset". Pressing <ENTER> will return you to the MENU leaving the MODE unchanged. Do not assume things when dealing with the differences between single and double density. If you don't know for sure, leave it alone! The format is:

Single or double ? parameter <ENTER>

The parameters are:

S - Single density
D - Double density

The next question asked is "Five or eight". Enter a numeric five (5) or a numeric eight (8), depending on what the drive you are accessing is. Please note that the parameter, if you just press ENTER, remains unchanged. If you set the mode at eight inch, and then come back later to a five inch, you MUST reconfigure the Diskzap for eight inch. Using eight inch operation on a five inch drive is disastrous.

Then it will ask you "Sector offset ?". If you are using DISKZAP on a DOSPLUS disk, this value should be 0, if you are using it on a TRSDOS disk the value should be 1. This number is the starting sector number on each track. After the MODE has been set, you will be returned to the MENU. The mode must be reset every time the diskette parameters change for the disk you

are working on.

ZERO

This utility will set any part of a disk that you specify to HEX 00. When you select this utility from the MENU, you will be prompted as follows:

```
Drive ?  (drive to be zeroed)
Track ?  (track to start at)
Sector ? (sector to start at)
Sector count ? (number of sectors to be zeroed)
Type data address mark ? (should be a one if you are
                          zeroing a directory track.
                          otherwise set it to zero.)
```

NOTE: Pressing <ENTER> will give the default value (lowest possible value for each prompt.

The track and sector numbers are given in hex, but the sector number is decimal. THIS IS IMPORTANT.

COPY

This utility will allow you to do a sector for sector copy of a disk, or BACKUP a whole disk. When you select this utility from the MENU, you will be prompted as follows:

```
Source mode ? (is the disk you are copying from single
               or double density)
Five or eight ? (is the disk five or eight inch)
Drive ? (which drive are you copying from)
Track ? (track to start copying at)
Sector ? (sector to start copying at)
```

You will then see:

```
Destination mode ? (is the disk you are copying to
                    single or double density)
Five or eight ? (is the disk five or eight inch)
Drive ? (which drive are you copying to)
Track ? (track to begin at)
Sector ? (sector to begin at)
Sector count ? (number of sectors you wish to copy)
```

The source question establishes where you want to begin copying, the destination question establishes where you want to write and the sector count question ascertains how many sectors you wish to copy. If you wanted to copy an entire 40 track disk, your sector count would be 720.

NOTE:

DISKZAP will copy any program as long as the tracks and sectors

are numbered exactly as you specified. However, it does not create Directory entries. Without a Directory entry, your program is useless. Do not expect an entry to appear in the Directory just because you copied some files from another disk.

PRINT

PRINT will give you a hard copy of what the display option prints on the screen. When you select this option, you will get the following prompts:

```
DRIVE ? (default value = 0)
TRACK ? (default value = 0)
SECTOR ? (default value = 0)
SECTOR COUNT ? (default value = 1)
```

For the default value press, <ENTER> when each prompt appears on the screen. After you have answered the prompts, DISKZAP will display the first sector on the screen and at the same time send it to the printer. It will then clear the screen and display the next sector on the screen while sending it to the printer. It will continue in this fashion until the requested number of sectors have been printed.

The printed copy will look like this:

```
000000: FF87 6D01 0084 3A99 49CE 4E3A 4824 D522 ..m...:.I.N:H$."
000010: 3031 3233 3435 3637 3839 4142 4344 4546 0123456789ABCDEF
000020: 2200 B06D 0200 B222 4D6F 6465 6C20 4949 "..m..."Model II
000030: 4920 5041 5443 4820 7072 6F67 7261 6D20 I PATCH program
000040: 2D20 5665 7220 332E 3322 00E7 6D03 00B2 - Ver 3.3"..m...
000050: 2243 6F70 7972 6967 6874 2028 6329 2031 "Copyright (c) 1
000060: 3938 312C 204D 6963 726F 2D53 7973 7465 981, Micro-Syste
000070: 6D73 2053 6F66 7477 6172 6520 496E 632E ms Software Inc.
000080: 2200 F06D 0400 4931 D531 0014 6E05 008B "..m..I1.1..n...
000090: 4124 2849 3129 3A8F 4124 2849 3129 D6D4 A$(I1):.A$(I1)..
0000A0: 222A 22CA 4931 D549 31CD 313A 8D35 0032 "*".I1.I1.1:.5.2
0000B0: 6E06 00B2 4032 3536 2CF7 2833 3129 3B22 n...@256,.(31);"
0000C0: 5072 6F67 7261 6D73 3A22 3AB2 004F 6E07 Programs:":..On.
0000D0: 0081 49D5 31BD 4931 CE31 3AB2 492C 4124 ..I.1.I1.1:.I,A$
0000E0: 2849 293A 873A B23A B200 776E 0800 B2F7 (I):.:.:..wn....
0000F0: 2832 3729 F728 3330 2922 5365 6C65 6374 (27).(30)"Select
```

The first column is the address of the displayed sector. The first two digits are the track, the middle two are the sector, and the last two digits denote the HEX address of the first byte on the displayed line. The next eight columns are the contents of the eight bytes, starting at the addressed byte, displayed in HEX. The ASCII translation of these eight bytes are displayed immediately to the right. The unprintable characters are replaced by a period (.).

VERIFY

This option will check any or all parts of a disk to make sure that it is readable. When you select this option, you will get the following prompts on the screen:

```
Drive ? (default value = 0)
Track ? (default value = 0)
Sector ? (default value = 0)
Sector count ? (default value = 1)
```

To get the default values, press <ENTER> in response to the prompts as they appear. Remember that the first three prompts are for hex information, while the last prompt (Sector count) is in decimal.

If for any reason DISKZAP can not read a sector, it will halt and display an error message and the track and sector on which the error occurred. If you press <ENTER>, the verify function will continue. If it has paused on a read protected sector, pressing ENTER will cause it to verify that sector after pausing. Press <BREAK> to return to the MENU.

NOTE:
VERIFY will always stop when it encounters a deleted data address mark. Also, while every other option will try five times to read a sector before it gives up, VERIFY will try only once. The mere fact that you are

checking a disk indicates that it is suspect and the least error should be reported.

FORMAT

This utility is a high speed formatter. It is very fast. However, it gets its speed by not verifying its format or writing system data to the disk. It is NOT intended to replace the FORMAT command of DOSPLUS. With this option, you can format a track anywhere on the disk. This is useful for fixing CRC errors on unused tracks. When you select this option, you will get the following prompts on the screen:

```
DRIVE ? (default value = 0)
TRACK ? (default value = 0)
TRACK COUNT ? (default value = 1)
```

Press <ENTER>
To get the default value or enter the desired parameters.

NOTE: This formatter is intended to be used within DISKZAP only. It does not write a bootstrap or a directory to the disk. Without this system information, the disk can not be used by the operating system. Also note that the track count is in decimal, where all other data is in hex.

DISPLAY

This option is the heart of DISKZAP. This is the option that will allow you to display and modify any sector on a disk. The display is identical to that of the PRINT option. When you select this option, you will get the following prompts on the screen:

```
Drive ? (default value = 0)
Track ? (default value = 0)
Sector ? (default value = 0)
```

As usual, press <ENTER> to get the default value when the prompt is displayed. After the first sector is displayd, you have the following options:

```
; - Display next sector
+ - Display same sector on next track
- - Display prior sector
= - Display same sector on prior track
M - Enter MODIFY mode
```

When you select the MODIFY mode, you will get a block cursor over the first nibble of the first byte in the upper left hand corner of the screen. You can move this cursor on the screen using the up, down, left, and right arrows keys. You can zero from the current cursor position on by pressing the Z key.

When you are in the MODIFY mode, the keyboard will accept valid hexidecimal characters only, all other entries will be ignored. The input is nibble oriented and you must completely modify the nibble you are on before DISKZAP will recognize any other key. This includes <ENTER> and <BREAK>.

After you have finished your modifications to the displayed sector, you may either press <ENTER> to write the modified sector back to the disk or <BREAK> to exit, leaving the sector unchanged on the disk. In both cases you will be returned to the DISPLAY mode. To return to the DISKZAP MENU, press <BREAK>.

HZAP -

The HZAP hard-disk editor is a program designed to be used ONLY by the experienced programmer/computer operator. It may NOT be used if the hard disk controller board is not attached to the data bus.

HZAP will power up with the selection switch in the "set" position. This is where you will set up the parameters for your hard drive. Please note that HZAP will not work on your floppies. DISKZAP is still there for that.

After you press ENTER to select "set", HZAP will ask you for :

Drive number.
Platter count.
Cylinder count.
Sectors per track.
Step rate.
Write precompensation.

Drive number is obvious. This is the drive with which you will be dealing. Drive numbers 0-3 are valid. Anything else will cause a lockup or be rejected. Drive zero is drive four, drive one is drive five, etc., etc. Again, please do not attempt to use HZAP unless your hard disk controller is attached.

Platter count is also obvious. You should know the number of platters in your hard drive. If you do not, consult your owner's manual or call the dealer who sold you te drive

Cylinder count is the number of cylinders your drive was formatted to. Again, consult your owner's manual for the drive to determine how many this should be. The actual formatting of the cylinders will be taken care of by the program HFORMAT described later.

Sectors per track is the same thing as track size. For a more detailed explanation of this parameter, consult the section "Introduction to hard drives".

Step rate will also be determined by your hardware.

Consult your owner's manual for the step rate that your drive is capable of. It should be a value between 1 and 7 milliseconds.

Write precompensation is seeking the cylinder number that you wish to engage it at. We recommend cylinder 100 (hex 64). Please be cautious. Engaging write precom too soon or too late are both equally bad ideas. They can cause all sorts of unusual read errors.

When configuring HZAP, please don't contradict the information that you inputted at CONFIG. All of these parameters except cylinder count are also set in CONFIG. Cylinder count is determined at format time. Since HFORMAT pulls all the other information from CONFIG, whatever is set there is how the drive is formatted. Using different parameters in HZAP is an extremely poor idea.

The only other major difference between HZAP and DISKZAP is that instead of the option "zero" it has the option "fill". These two commands will function in the same manner with two exceptions.

First, "fill" will ask your for the byte you wish to fill with where "zero" always uses zeros. Second, "fill" does not give you the option of selecting the data address mark where "zero" used to. This was deemed unnecessary and has been removed (hard drives don't use data address marks).

Finally, HZAP will always ask you for "cylinder" and "sector" as opposed to DISKZAP's "track" and "sector". For a detailed explanation of the relationship between tracks and cylinders, consult the section "Introduction to hard drives" elsewhere in this manual.

To exit HZAP, from the menu press "O" (for out).

All other areas of HZAP are virtually identical to DISKZAP, and the DISKZAP portion of this manual should serve as sufficient instruction to become proficient in their usage.

FORMAT/HFORMAT -

This utility formats a new disk and prepares it to receive data. It also initializes the bootstrap and directory system information for the operating system. The format is:

```
FORMAT  <ENTER>
          or
FORMAT :d  <ENTER>
          or
HFORMAT <ENTER>
          or
HFORMAT :d <ENTER>
```

":d" optional drive number. If omitted you will be prompted for it later.

Depending on which program you want. The computer will then prompt:

Which drive is to be used ?

Answer this question with the drive number that contains the disk to be formatted. Be certain that if you are going to use drive zero, you remove the Dosplus system disk at this time. If you specify the drive number when entering FORMAT, this question will be skipped. Pressing ENTER will default to drive zero.

Format date (MM/DD/YY) ?

Input the date at this point. If the date has already been set via the DATE command, FORMAT will skip this question. If the date is not set, the question will be asked. If it is not important, you can press ENTER and the date will default to 00/00/00.

Password ?

Answer this with the password that you wish to assign to that diskette. This will be the "Diskette master password" that is used with the PROT command (see library commands). If no special protection is needed, we recommend a null password. This is accomplished by answering this question with pressing <ENTER>. This is the way you received your Dosplus diskette. Whatever you set your password to, don't forget it.

*** FORMAT ***
Number of cylinders (35-96) ?
*** HFORMAT ***

Number of cylinders (100-200) ?

Answer this question with the number of cylinders you wish to have formatted. This is limited by your hardware. Consult the owner's manual with your disk drives if you are uncertain. The standard Radio Shack drives are 40 track (although some of the older R/S drives are 35 track only). If you press ENTER, it will default to 40 cylinders. For a complete explanation of the new 3.4/4.0 cylinder/track relationship, please consult the section "Introduction to hard drives" in the front of the manual.

*** FORMAT only ***
Single or double density ?

Unless you are going to take this disk back to a Model I, answer this question with 'D' (for double). If you wish to create a single density diskette, type 'S' (for single).

*** HFORMAT only ***
Sector interleave factor ?

Answering this question by pressing ENTER will default to a sector interleave of one. We have found a sector interleave factor of 8 to be optimum for speed of verification. Where access times are concerned, the difference between one sector interleave and another is so slight as to be insignificant.

If the diskette contains data, you will see the message :

Diskette contains data, Use or not ?

Answer this question with a 'N' (for no) if you do not wish to use it, and the format command will be aborted. Answer it with a 'Y' (for yes), or a 'U' (for use) if you do. If you tell the machine to use the disk, it will erase all previous data. Please be careful.

After a few moments, the screen will flash :

Insert SYSTEM disk <ENTER>

At this point make sure that the Dosplus system diskette is in drive zero, and then press ENTER. The format is complete.

After you respond to all of the questions, DOSPLUS will format the disk and write the system data to the disk, check the disk for errors, lock-out any defective granules,

and write the directory and bootstrap to the disk.

You may also re-format a disk that has been used (previously formatted). However, any data on this disk will be lost. If you re-format a disk, the data on the disk will be lost. There is no way to recover this data once it has been erased.

The Model III IS capable of single density operation. However, there are some differences in the Model III single density and the Model I single density. Diskettes formatted in the Model I CANNOT be read without being converted to Model III format (see CONVERT).

Diskettes formatted in the Model III single density can only be read in the Model I by the more intelligent operating systems on the market. All of the 3.3 or later series Model I Dosplus' will read Model III single density.

The only difference in FORMAT and HFORMAT has to do with the drive that it deals with. They are in effect the same program, but HFORMAT is geared for the hard drive.

MAP -

This utility will map out all the files on a diskette, showing on what cylinders and sectors each file is located. The format is:

MAP filename :td (P)

"filename" is the wildcard mask that you can use to limit yourself to a particular file or type of file.

":td" is the target drive. If omitted, drive zero is assumed.

"(P)" optional printer output. If you specify this, the data will go both to the screen AND the line printer. If it is omitted, output will go to the screen only.

Your output should look something like this :

```
DOSPLUS  03/25/81
PURGE      CMD      021,012 - 021,017
ACCREC     BAS      005,013 - 005,016
                    010,000 - 010,005
SYS8       SYS      019,000 - 019,005
CRUNCH     CMD      019,006 - 019,011
```

The first line is the diskette name and the date it was last formatted or backed up. The three columns are:

Column one - File name
Column two - Extension
Column three - Beginning cylinder, sector
to
Ending cylinder, sector.

This information is in decimal. In order for the tracks and sectors listed in MAP to work with DISKZAP, you must convert them to hex. The blank space under the filename "ACCREC/BAS" indicates that the next extent belongs to it also.

The wildcard filename option allows you to map a single file. Or all the "/CMD" files. Or all the "/BAS" files. Or whatever, it is very flexible. For example, let's say that while loading Scripsit, you hear the drive retry. You know that the file is marginal, and you'd like to find out where it is and correct it with DISKZAP. First, boot up Dosplus. Then place the disk with Scripsit on it in drive one (or it could be the same disk, if your Scripsit is on Dosplus 3.4/4.0). Then type :

```
MAP SCRIPSIT/CMD :1
```

This will scan drive one and report the location of Scripsit. Convert the addresses to hex and you can access that file via DISKZAP. Enter the modify mode and re-write the sector without changing anything (i.e. "M" <enter>). Nine times out of ten, that is enough. You can also do a :

```
MAP :1
```

And it will report on all files. Or you could :

```
MAP /CMD :2
```

And it would map out all the files with the "/CMD" extension on drive two. You could also :

```
MAP ??B/?XT :3 (P)
```

This will display the location on drive three of any files that have a "B" for the third letter of the filename and have an extension that ends in "XT". It will duplicate that output to the line printer.

PURGE -

This utility will call up the directory of a disk and ask if you want to delete the files, one at a time. To delete a file, you answer Y for YES and press <ENTER>. Pressing <ENTER> alone will tell DOSPLUS to keep the file on the disk. The format is:

PURGE filename :td (par)

"filename" is the wild card mask that allows you to purge only files that meet your specifications.

":td" is the target drive number.

"(par)" is your optional parameter.

The legal parameters are:

S - System files
I - Invisible files
A - Absolute purge

S (System). If you include this as a parameter, even system files will be called into the purge. Otherwise, it will work only with the visible user files.

I (Invisible). If you include this, purge will also consider the invisible files. Otherwise, it works only with the visible user files.

A (Absolute). This parameter assumes that you have set up your mask to screen for certain files, and you really don't want to be asked whether or not to purge the file for each one of programs. If this parameter is included, the question mark will not be displayed. It will simply go in and purge each file that matches the mask, displaying the filename as it kills it. Use this parameter with care.

Example -

PURGE /CMD :1 (I)

This will call up all the user files on drive one that have the "/CMD" extension and ask you whether or not you wish to purge it. It would look something like this :

```
DOSPLUS - Model III PURGE utility - 3.4
Copyright (c) 1981, Micro-Systems Software Inc.

RTEST/CMD       ?
SCRIPSIT/CMD    ?
TAPE/CMD        ? Y             *** PURGED ***
```

FS/CMD ?

When you see the question mark, you may :

* Enter "Y" and purge file.
* Enter an "N" and skip file.
* Press <enter> and skip file.
* Press break to abort the purge.

Example -

PURGE :2 (S)

This example would call up all system and visible user files from the disk in drive two and ask if you wish to purge them.

Example -

PURGE :3 (A)

This would kill all visible user files on drive three.

Example -

PURGE PR4??B/DAT (S,I,A)

This would purge any file on drive zero that matched the mask (i.e. Begins with PR4, has two unspecified characters ending in a "B", and has a "/DAT" extension.) It would ask no questions and would consider even system files.

System files are those that are essential to the proper operation of Dosplus and its library commands. They are denoted by the "/SYS" extension and by the "*" in the ATTRB column of the DIR function. (see FILESPECS and DIR)

If you wish to make the smallest possible "working" diskette, purge off all the utility files. That is, any of the files with the "/CMD" extension. You may want to leave, say, FORMAT and BACKUP and kill all else.

The general layout of the system files is :

SYS0-SYS4

These systems are VITAL to the operation of Dosplus. If they are not present, the DOS will NOT function properly to even load and execute programs.

SYS5

Error messages. This system also cannot be safely purged from the disk.

SYS6

Debug system. If not being used, it can be purged.

SYS7-SYS15

These contain the library commands. They are interactive (call one another). If you are going to remove one for the sake of room, you may as well just purge them all and forfeit your library commands.

Therefore, a minimum system disk to load and execute machine language programs would be systems 0-5.

Deleting a system file will cripple DOSPLUS, since all system files are interactive. If you do delete a system file, there is no telling what odd things DOSPLUS may do. However, if you delete a file you didn't want to delete, the RESTORE command may recover it.

Purge does not need a password. It will delete any file from a disk regardless of its protection level. If you press <BREAK> before the last file has been called up, you will exit PURGE and leave the remaining files on the disk unchanged.

RESTORE -

This utility will bring a file "back from the dead". However, RESTORE will not recover a file that has been over-written by another file. Its primary function is to recover a file that has been deleted by accident. The format is:

RESTORE filespec:d <ENTER>

Please note that the drive specification on RESTORE is NOT optional. If you leave it off, RESTORE will scan for the file on drive zero ONLY! It does NOT do a global search of all drives when looking for a file.

And also note that RESTORE will restore the first occurrence of that file in the directory. If that is not the particular one you want, rename the restored file to something else, and try again.

RESTORE will rebuild the file's entry in the GAT (Granule Allocation Table), and the HIT (Hash Index Table). It will bring the file back with the same attributes and protection levels it had before you killed it.

Example -

RESTORE TSTPGM:1

This will seek to restore the file TSTPGM on drive one.

RESTORE PAYDAT

This will look for the file PAYDAT on DRIVE ZERO ONLY!

SPOOL -

Spool printer text to memory or disk.

SPOOL filename (par=exp)

"filename" is the optional disk overflow filename. If you tell the spooler to go to disk and don't give it a filename, it will default to "SPOOL/TXT". If you give it a filename and no extension, it will use the "/TXT" extension.

"par" is the control parameters to set up the spool buffers.

"exp" is the value you will set it equal to.

The following are your valid SPOOL parameters :

"DISK" can be anywhere from 0 to 400 kilobytes. If not specified, it defaults to zero.

"MEM" can be from 1 to 32 kilobytes. If not specified, it defaults to four.

The number is assumed to be in kilobytes, so enter them that way. (i.e. MEM=12 for 12K memory buffer area)

The spooler enables you to set up areas in memory and on disk that the computer can send data to at a high rate of speed, thereby enabling the program currently running to proceed to the next step, and the text will be printed as the printer is available. Note: The spooler is interrupt driven. What this means to you is that it will never print any faster than 80 cps no matter how fast your printer is. Also, printing will be temporarily suspended during disk I/O. The spooler may seem to run slower and then faster. This is due to the program you are spooling from, the printer you are using, and the way that the interrupts are being used. Don't be alarmed at this. If the spooler is printing, it is working. It speeds up when it can and slows down when it has to.

You spool two ways :

First, you can spool to memory only. You set up an area in memory for the spooler to use without specifying a disk file for text overflow. This is the fastest and most common method used in spooler operation. To do this, type SPOOL (MEM=however many K you want it to use). When it fills up memory, it will usurp control of the CPU and function as normal during a print operation until enough

memory is freed for it to continue spooling. If this happens, don't be alarmed. The spooler is still in control, and will return the CPU to you when it is done with it.

Example -

```
SPOOL (MEM=5)
```

This will set up a 5K spool buffer in RAM. The "DISK" parameter defaults to zero and leaving it unchanged tells the spooler to spool to memory ONLY!

Or second, you can spool to memory with optional disk overflow. This method is used when you will be potentially spooling enormous amounts of text. This is not quite as fast, because the spooler must interrupt the CPU when turning on the drives to either send text to the disk, or when pulling text off the disk.

If the filename is already in existence, the spooler will open the file and write text to it. It starts fresh every time. Don't give it the wrong name, or it might write your payroll data on top of your favorite game.

Example -

```
SPOOL PAYDAT/PTR:2 (MEM=20,DISK=20)
```

This will spool up to 20K into memory and if that gets full, it will use up to 20K of disk overflow area in a file called "PAYDAT/PTR" on drive two.

Please make certain that the disk you are spooling to has enough space to handle all the data. "Disk space full" error messages in the middle of printing a file can be distressing.

Remember, once the spooler is loaded, it will remain in effect until the system is re-booted. It also will not interfere with the DO buffer in high memory, so you can activate it from a BUILD/DO file.

You can activate the spooler from BASIC via CMD"SPOOL", but be CERTAIN that you have protected all of the high memory that the spooler is going to use. Otherwise you will LPRINT into the middle of BASIC's high memory and LPRINT your way to real problems.

DO NOT remove the disk containing the spool file while the spooler is on. The spooler will write to the new disk at the sectors it thinks the spool file is occupying, overlaying everything in its path. Be careful!

SYSGEN (Generate non standard system diskette) -

The format is :

SYSGEN :td

":td" is the target drive number.

Technical note : You may NOT have any files on a drive that you wish to sysgen. The drive must have just been formatted.

The primary usage of this utility is twofold. One, to create double headed system diskettes. Two, it allows you to place Dosplus (4.0 only!) on a hard drive and use that drive as the system drive. You must create double headed system disks using the sysgen method due to Dosplus 3.4/4.0's new "single volume" addressing of double headed drives. You no longer have a side "A" and side "B" which are viewed as two seperate drives. If you wish double headed operation, you must create a double headed disk.

While a single headed diskette will boot in a double headed drive, the converse is NOT true. Double headed disks will not boot up in single headed drives. Once booted up, Dosplus will run ANY combination of cylinder counts. However, it has to boot initially or it can do nothing. This utility was created to circumvent these problems.

SYSGEN will only work if your system has more than one drive. If you have one single headed and on double headed drive, you can boot off the single head and sysgen the double head. If you have only one drive or if for some reason your double headed drive HAS to function as drive zero, we will prepare a double headed disk for you. Return your master diskette to us (with a description of your drive) and for a small service charge, we will create the new disk for you.

Remember, if you are not using this drive as a system drive, then you do not need to sysgen it. If your double headed or hard disk drive is only functioning as a data drive, then simply configure it correctly, format it, and go.

Procedure for configuring a hard disk -

First, make certain that the hard disk CONFIG parameters are as you want them to be. Then, HFORMAT the drive (see HFORMAT).

Then type :

SYSGEN :td

where ":td" is the target drive number.

When the DOS PLUS prompt is re-displayed, signalling completion of the sysgen, type :

TRANSFER :0 :td (I)

where ":td" is the target drive number.

Transfer will appear on the screen and you should see the names of the various utility files as they are copied. When it is finished, type :

CONFIG (SYSTEM=td)

where "td" is the target drive number.

The type "CONFIG" and press ENTER to display system parameters. Make certain that the "$" is next to the specified drive. Then type :

CONFIG (SAVE)

and reboot. Once the system is booted, if you wish to have the hard drive defined as the master drive as well (see CONFIG), do the following. Remember, this is OPTIONAL. We do recommend it in most cases, though, so that the operation of the system is similar to that of the floppy equivalent (i.e. DIR <enter> defaults to the system drive, etc. It prevents a lot of waiting while the DOS scans through four "empty" drives.). The procedure is :

CONFIG (MASTER=td)

where "td" is the target drive number.

Again, type :

CONFIG (SAVE)

From now on, you only need your floppy disk long enough to boot up. After that, all system operation will be from the hard drive. To get to drive zero, you will have to ask for it.

Technical note : It will be necessary to re-load the floppy whenever you press the reset button. This is due to the fact that the ROM looks for the bootstrap on drive zero. Some hard disk systems implement a "direct to hard disk" ROM alteration. Dosplus may or may not be configured to operate with this. If your computer has such a ROM modification, contact the drive manufacturer (or Micro-Systems Technical Support Division) to see if their version of Dosplus has this option. Also, any programs

that exit by resetting the system (i.e. a jump to memory address 00H) will require the floppy to be re-loaded.

To remove the configuration, after pressing reset, hold down the break key. Press reset first, otherwise you will jump to ROM BASIC. This will cause the DOS to override the system parameter and boot off the floppy. At that point you may re-configure the system drive as drive zero, save it and then change the master drive as well (if desired). This allows you to use your Dosplus 4.0 with a non-sysgened hard disk. Also, if your hard disk needs repair, you may temporarily return to floppy operation.

Procedure for creating a double headed floppy disk -

First, make certain that your drive configuration is correct. Then FORMAT the drive (see FORMAT). FORMAT will pick up the double headed configuration and format both sides of the disk. After that type :

SYSGEN :td

where ":td" is the target drive number.

When the DOS PLUS prompt is re-displayed, signalling the completion of the sysgen, type :

TRANSFER :0 :td (I)

where ":td" is the target drive number.

You will see the names of the various utilities appear as they are copied over to the new diskette. When TRANSFER has finished, you are done. The diskette you have just made is now a double headed system diskette.

Technical note : As mentioned in the CONFIG section, the procedure just completed has set your SIDES parameter for drive zero equal to 2. You do not have to enter this from CONFIG and if you try it will produce an error. SYSGEN is the only method of doing this.

Furthermore -

SYSGEN can be used ANY time that you wish to make a diskette different than the one that you originally received. Any diskette that you can format, you can make into a DOS. This becomes highly convenient when a user has more that one machine and they have differing drive zeros.

By using the procedure described in the section on producing a double sided floppy disk, you can create 80 track DOS' from 40 track versions and vice versa. All that is needed is to have the correct hardware available.

Technical note : SYSGEN will (when sysgening floppies) pick up the configuration currently in memory. This means you do not have to permanently configure the master diskette you are sysgening from in order to create a non-standard disk. You may make all needed changes in memory and proceed with the sysgen without ever having to write to the master diskette.

Restrictions -

Although sysgen will attempt to compensate for locked out granules, there are two major restrictions.

First, the diskette that you are using in the sysgen must be freshly formatted or have had all files purged save "BOOT/SYS" and "DIR/SYS". If there are any other files present, sysgen will abort.

Second, the system diskette that you are sysgening from must not have any of the system files purged on it. If some of the utility files are missing, TRANSFER won't care. But SYSGEN will expect all the system files to be there.

As long as you keep to the described procedures and beware of the above mentioned restrictions, you should be able to use SYSGEN to transport your Dosplus to whatever type of media you desire.

TAPE -

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 at this point.

Once the tape utility is loaded in, the program header will be displayed and you will have the command prompt (an asterisk). Type "H" and press ENTER for a "help" command.

Example -

```
TAPE <enter>

DOSPLUS - Disk/tape utility - 3.4/4.0
Copyright (c) 1981, Micro-Systems Software Inc.

*H
B,H,I,L,M,O,Q,R,S,W
```

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.

"H" prints the list you have already seen. That is all that it does.

"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 seperately. 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 I offset it 1000 hex bytes ("O 1000"), it will now load in at hex 9C00. Please note that tape works only with hexidecimal 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.

General notes -

The tape utility can be used to relocate programs and then dump them back to disk or tape. There is nothing that says you HAVE to go from one media to another.

It can also be used to simply interrogate a program as to where it loads in memory. In a lot of de-bugging applications for a mchine language programmer, it is nice to know that there are no "hidden" segments giving you fits. Tape will map them out for you.

We feel that it is by far the most superior program of its variety included standard with a DOS.

TRANSFER -

This utility will copy all user files from one disk to another. It will only work for the multiple drive user.

TRANSFER filename :sd :dd (I)

"filename" is the optional wildcard mask that you use to control which files are transferred and which are not. If omitted, the entire disk will be transferred.

":sd" is the source drive number.

":dd" is the destination drive number.

"(I)" tells TRANSFER that you wish to copy invisible files also. Otherwise it will only copy visible user files. TRANSFER does not move system files. Use SYSGEN for that.

With the new wildcard option, you transfer only certain files, such as all the data files (/DAT) or all your payroll journals (PAY/JNL). You can mask for a filename, a piece of a filename, or an extension. This makes TRANSFER useful as being the "BACKUP" utility for the hard drives.

Suppose we have started off all of our payroll files with the notation "PR" as in "PRCHK", "PRTAX", "PREARN/DAT", etc. When we have finished payroll, we would insert the backup floppy in one of the drives, say drive two, and type :

TRANSFER PR/DAT :4 :2

This will copy over all the payroll data files to the floppies. Programs you can reconstruct, but data files and indexes shpuld be backed up. There are more examples of this later.

If TRANSFER encounters a non-protected file of the same name on the destination disk, it will pause and display :

Overwrite ?

If you respond to this question with a "Y", it will proceed. If you respond with a "N" or <enter>, it will skip this file and proceed to the next one. If you press <break>, it will abort the transfer.

TRANSFER will keep the attributes of the file being

copied the same. This is new from earlier Dosplus'. Earlier TRANSFERs would strip passwords and protection levels. This version will not. Although it will copy a protected file, it will NOT copy OVER a protected file. If it encounters a protected file on the destination disk with the same name as the file it is moving, it will generate a "File access denied due to password protection" error and abort.

TRANSFER will serve to move files between single and double density disks and vice versa. After you convert your Model I diskettes for Model III, you can transfer all the programs to double density with a single command.

Examples -

```
TRANSFER /CMD :1 :2
```

This will copy all the visible user files that have the extension "/CMD" from drive one to drive two.

```
TRANSFER :1 :0 (I)
```

This will attempt to copy all user files from the drive one disk to the disk in drive zero.

```
TRANSFER ??PR4/DAT :4 :3
```

This will copy all the visible user files from drive four that have a "PR4" beginning at the third character and have the extension "/DAT" and put them on drive three.

Note : The drive four is reserved for hard disks in Dosplus 4.0. If you are not using Dosplus 4.0 with a hard drive attached, this will produce a "Illegal drive specification" error.

HCOPY (Offload or Reload a file) -

This utility allows you to copy a file off the hard drive that is too large for a single floppy to hold. The format is :

HCOPY <enter>

there are no parameters at this point.

After HCOPY has loaded itself in, it will ask you :

<O>ffload or <R>eload file ?

Respond with the corresponding letter. If you wish to move a file FROM the hard disk to floppies, then you are offloading. Conversely, if you wish to move a file TO the hard drive from several floppies, you are reloading.

It will then prompt :

Source file :

Answer this with the filespec of the file you are moving. It will then prompt :

Destination file :

Answer this prompt with the filespec of the file you wish to copy the data into. It will then begin the copy. When any one floppy disk is either filled up (Offloading) or all the data has been copied from it (Reloading), the program will prompt you for the next floppy.

When the program has completed the copy the message "File copied" will be displayed and the program will return to DOS.

Restrictions -

First, the file you are attempting to reload MUST have been created by HCOPY. This utility knows its own files and will not reload what it has not offloaded.

Second, all data diskettes must be formatted in advance. You do not have the option of formatting a disk in the middle of a copy. To avoid problems, make certain that you have plenty of disks ready to recieve data.

Third, the last diskette copied to must be the last diskette copied from. It has special data pertinent to correct program operation on it. For example, let us suppose you HCOPY the file "SUPER/DAT" and it fills up five floppy disks. When you are loading that file back onto the

hard drive, the fifth disk copied to MUST be the last diskette placed in the machine. However, the order of the first four does not matter.

As long as you are careful to keep the last disk seperate, you should have no problem. Inserting the last disk early will cause a premature abort of the program.

Technical notes :

No matter what logical record length the file being offloaded is, it will be set to 256 when it is reloaded. This will not affect program operation at all inasmuch as you may open a file, under Dosplus 3.4/4.0, with any logical record length. The LRL is there for display's sake only. If you wish to avoid this, make certain that you CREATE (see CREATE) the destination file first, giving it the desired logical record length.

When copying a file, HCOPY writes until the diskette is filled before asking for the next disk. This means it will automatically use all available disk space on 80 track or double headed disk drives.

HCOPY may be used from Extended Disk BASIC from within a program (if desired).

Disk BASIC

Introduction -

Disk BASIC is simply that, a set of enhancements to the Model III 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. Disk BASIC is completely memory resident on the Dosplus system diskette, and comes in two forms - BASIC and TBASIC.

BASIC loads in at hex 5700 (dec 22272).
TBASIC loads in at hex 5200 (dec 20992).

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 commands & DOS commands. Less memory effecient.

TBASIC - memory effecient. Lacks Dosplus system commands and extended features.

You must specify, upon calling BASIC, the number of files 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 0 file buffers.

You use the following syntax :

BASIC filespec-F:#buf-M:#mem

filespec	an optional Dosplus file specification. BASIC will load and execute this file after loading itself.
-F:#buf	"#buf" is the number of files you expect to be using at one time (can be from 1-15). You may not specify a higher buffer during an "OPEN" than you specify here.
-M:#mem	"#mem" 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 highest address physically in the machine.

For example :

BASIC

Load BASIC with no files or 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.

BASIC filespec

Load BASIC and load and run the specified BASIC program.

BASIC -F:3

Load BASIC and allocate 3 files.

BASIC -M:61000

Load BASIC and protect Memory above 61000.

BASIC filespec-F:3-M:61000

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

NOTE: There is a mandatory blank space between BASIC and any parameter. Once BASIC is loaded, there is no way to set files or protect memory. You must return to DOS and re-enter BASIC using the proper syntax.

EXAMPLE : BASIC PAYROLL/BAS-F:4

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

Once you enter Disk BASIC, you will see the header :

```
DOSPLUS - Extended Z80 Disk BASIC - Ver 1.6
Copyright (c) 1981, Micro-Systems Software Inc.

READY
```

or

```
DOSPLUS - Tiny Disk BASIC - Ver 1.5
Copyright (c) 1981, Micro-Systems Software Inc.

READY
```

depending on which version you summoned.

Once you have entered Disk BASIC, you can return to DOS by simply typing "CMD" and pressing ENTER. For TRSDOS compatibility, CMD"S" will also work. If you have done this in error, you may immediately re-enter BASIC with "BASIC *" and not harm your program or data in RAM. As a rule, be sure to save off any program in memory before returning to DOS.

Disk BASIC features -

Disk BASIC contains several enhancements to the Model III ROM resident BASIC that have little to do with actual disk I/O. We will cover these first. They appear in alphabetical order.

&H (hex constant) -

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

&Hdddd

where "dddd" is a one to four digit hexidecimal value. (0,1,..9,A,B,..,F)

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

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

Hexidecimal values may not be typed in in response to an INPUT statement or included as numeric data in a DATA statement.

Examples -

```
PRINT &H5200
```

This will print the value 20992 on the screen.

```
POKE &H3C00,65
```

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

```
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.

```
DEF FN var1(var2)=exp
```

"var1" is the variable that will be used as the name of the function.

"var2" is used in describing what the function does.

"exp" is the expression that manipulates "var2".

Once a function is defined, you may call it simply by referencing "var1" prefixed with "FN".

The type of variable specified in the "var2" position determines the type of variable that will be returned by the function. For example, if "var2" was "RN!", the value stored in it would always be single precision even if the function involved only integers.

For example :

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

will ALWAYS return a double precision value in no matter what the precision of the variables used in the actual calculation. 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 you use to pass information to the function or what variable you end up with the data in. For example :

```
V#=FN TV#(D!)
```

would have worked just as well.

The function must be defined with at least one arguement, even if this arguement is not actually used to pass a value to the function.

DEFUSR (define entry address for USR routine) -

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

DEFUSRn=address

"n" is the USR routine number. It is a number between 0 and 9. If it is omitted, 0 will be assumed.

"address" is the address of the machine language routine (entry address).

For example :

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.

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 a Editor/Assembler to create a diskfile from source code and then load it into RAM via CMD"LOAD filespec".

In either case, make certain that you protect the area of memory that the routine is destined for by using the "-M:address" syntax. If you do not, then BASIC will destroy the routine by using the area for string storage or some such function.

Refer to the section on USR routines.

INSTR (string search function) -

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

INSTR(n,strl,str2)

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

"strl" is the name of the string to be searched.

"str2" is the sub-string you want to search for.

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.

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 value	Result
INSTR(A$,"ONE")	6
INSTR(A$,"123")	0
INSTR(A$," ")	5
INSTR(3,"123123","12")	4

LINE INPUT (input a line of text from keyboard) -

This function allows you to input an entire line of text from the keyboard. It functions almost the same as INPUT except that only ENTER serves as a delimiter.

LINE INPUT"Prompt";var$

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

"var$" is the name of the string you wish line input to return you the information in.

It differs from INPUT in that :

* When the computer is 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 is to press ENTER.

Line input is used when you wish to input a string and not be worried about the accidental entry of delimiters. If you want to allow anybody to input data into your program without special instructions, use line input and then analyze the resultant string.

Line input will serve well in the event that you need to input a string that includes leading blanks, commas, and quotes. For example :

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.

MID$(var$,n1,n2)=rep$

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

"n1" is the starting position for the replacement.

"n2" specifies how many characters will be replaced. If omitted, LEN(rep$) or LEN(var$)-n1+1 is used, whichever is smaller.

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

Note : The length of "var$" is never changed. If "rep$" is longer that "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.

Examples :
(assume A$="ABCDEFG")

Expression	Resultant A$
MID$(A$,3,4)="12345"	AB1234G
MID$(A$,1,2)=""	ABCDEFG
MID$(A$,5)="123"	ABCD123
MID$(A$,5)="01"	ABCD01G
MID$(A$,1,3)="***"	***DEFG

This demonstrates the replace 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$="BCD" without affecting A$ 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.

USRn (call to user's external subroutine) -

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

USRn(exp)

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

"exp" is an integer in the range -32768 to +32767. It is passed as an arguement to the USR routine.

When a USR call is encountered in your program, control goes to the USR routine at the address specified in your DEFUSR statement. This address specifies the entry point to your machine language routine. A "RET" or a "JP 0A9A" instruction in your subroutine returns control to the USR call in your BASIC program.

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 arguement directly to and from the routine via the USR call itself, and others can be POKEed into RAM and then PEEKed out.

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. It returns a value in "A" if you return with the proper jump (see below) or else it mirrors the value of "X".

To pass arguements :

POKE values into reserved RAM and the your machine language routine call pull them out when it needs them. Have the machine language routine store its results in another area of RAM 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 arguements back and forth from a USR routine.

You can pass one arguement as the value from the USR routine, then use special ROM calls to get this arguement and return a value to BASIC. This method is limited to sending one variable to and from the routine (both must be integers).

ROM calls -

==========

CALL 0A7FH	Puts the USR arguement 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 arguement 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 to areas to this. The first is file manipulation. The second is file access. 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 in RAM.
RUN"progname"	Load and execute a BASIC program stored on the disk.
SAVE	Write the BASIC program currently in RAM to disk.

Statements and functions under file access
==========

Statements:

OPEN	Open a file for access (create the file if necessary).
CLOSE	Close access to the file (update DIR entry and EOF).
INPUT#	Read from a disk in sequential mode.
LINE INPUT#	Input a line of data from the disk in sequential mode.
PRINT#	Write to disk in sequential mode.
GET	Read from disk in random access mode.
PUT	Write to disk in random access mode.
FIELD	Assign field sizes and names to a random access 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 disk.
CVS	Restore single precision number to numeric form after reading from disk.
CVI	Restore integer to numeric form after reading from disk.
EOF	Check to see if end of file was encountered during read.
LOF	Return number of logical records in a file.

MKD$	Convert double precision number to eight byte string so that it can be written to disk.
MKI$	Convert integer to two byte string so it can be written to disk.
MKS$	Convert single precision value to a four byte string so that it can be written to disk.

KILL -

This command will kill a program or data file from the disk. It does not delete the directory entry and RESTORE can bring it back.

KILL"filespec"

"filespec" is a standard Dosplus file specification.

This command functions essentially the same as the Dosplus library command KILL. 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.

Example -

KILL"PAYROLL/BAS"

will search for the first time that the program "PAYROLL/BAS" occurs and will delete it.

KILL"ACCREC/DAT:2"

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

Note : Do not KILL an open file. Although Dosplus will correct for this and the diskette will not be destroyed, it is NOT a good programming practice and repeated instances of it will result in unreliable system operation.

LOAD -

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

LOAD"filespec",R,V

"filespec" is a standard Dosplus file specification.

",R" is an option that in this case means to RUN the program after loading. Equivalent to RUN"filespec".

",V" is an option which instructs BASIC to load the new program but preserve the current variables.

When you load a file without the ",R" option, it wipes out any BASIC program currently resident, destroys all set variables, and closes the files. To load a file with the ",R" option still destroys 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 detroy all currently set variables.

What this means is that a standard load and a load with the ",V" option both return to the command mode when done. Any time a ",R" is specified with these, the files are not closed (if any are even open) and the new program is RUN after loading. LOAD"filespec",R is the same as RUN"filespec",R.

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). After a LOAD or RUN you will need to re-field the files (if you kept them open with a ",R").

If you attempt to load a non-BASIC file the statement "Direct statement in file" or "Attempted to use non-program file as a program" 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 seperately. 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.

Examples -

```
LOAD"ACCREC/MOD"
```

will load the first occurence of "ACCREC/MOD" that it finds.

```
LOAD"ACCREC/MOD",V
```

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

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

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

MERGE -

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

MERGE"filespec"

"filespec" is a standard Dosplus file specification defining a file saved in ASCII format (i.e. SAVE"filespec",A).

Merge is similar to load except that the resident program is not wiped 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 be overlaid by the new 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.

Note : MERGE closes all files and clears all variables. When done, it returns to the BASIC command mode. It may not be used in run time from a program and should be restricted to development.

RUN -

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

RUN"filespec",R,V

"filespec" is a standard Dosplus file specification.

",R" is the optional syntax to indicate that you do not want the files closed when this program is loaded.

",V" is the optional syntax to indicate that you wish to have the variable area preserved when the new program is run.

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 program one will not be cleared, nor will the files opened by program one be closed when program two is run.

SAVE -

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

SAVE"filespec",A

"filespec" is a standard Dosplus file specification.

",A" is the optional syntax to indicate that you wish to save the program in ASCII format.

The compressed format uses less disk space and both loads and saves 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.

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

For most applications, you will find that saving BASIC programs in compressed form is the superior method. That is why this is the default.

Examples -

SAVE"PAYROLL/BAS:2"

will save the program "PAYROLL/BAS" in compressed format on drive two.

SAVE"TOBECOMP/ASC",A

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

Note : Upon completion of the command SAVE, BASIC returns to the command mode. However, you may use SAVE from a BASIC program. The program will then save itself and continue. This is useful for self-modifying programs.

FILE ACCESS -

This section is dedicated to those commands that are involved in the actual handling of the files. Where file manipulation is involved in allowing you to load and run various BASIC programs, file access is involved in handling your own data files. It will deal primarily in these areas.

INITIALIZING FILES -

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 file buffers that program is going to require. The number you specify when entering BASIC is the highest allowed buffer number.

Each buffer 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 file buffers 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.

All of these things are handled by opening and closing a file.

OPEN -

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

OPEN"mode",bufnum,"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.

"bufnum" is the buffer 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 intializing the buffer for.

"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 intialize a file I/O buffer for disk access. Let's examine each one in more detail.

"mode" 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

If a file opened for random access (R) does not exist, BASIC will create it. If it does exist, merely opening it does not destroy existing data. The file could be closed again right away 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 is in there. If you try and read a record from a random access file and that record is not there, it will return a buffer full of zeros. If you attempt the same thing with 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. If the file does exist, the current contents are lost as soon as it is opened. Even if you close it 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, it places a pointer at the end and all new data is added from that point on. 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.

Technical note : Dosplus also allows you to use an OPEN"D" statement. It is the exact equivalent of the OPEN"R" statement. It is there purely for compatibility with Model II programs.

"bufnum" is the 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. To change the default number of buffers (currently 0), consult the part of the Extended Disk BASIC section labeled "File zaps".

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

"filespec" is the standard Dosplus file specification (see FILESPECS) that you wish to open for input/output.

"lrl" is the logical record length of the file to be accessed. A physical record is defined as being one disk sector or 256 bytes. This is displayed in the "# PHY" column in the directory. 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 sub-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. The technical aspect of the logical record's relation to physical records is explained in the technical section of the manual, but suffice it to say here that (#LR=INT(256/LRL)). That formula will give you the number of logical records in any one physical record. Logical records can and do span sectors and tracks.

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 transfered to disk and the buffer cleared for more input/output. This means 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"
```

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. Buffer one will be assigned for access.

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

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

```
OPEN"R",1,"DATABASE/ASC:3",24
```

will open for random access the file "DATABASE/ASC" on drive three. 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 buffer one for access.

```
OPEN"E",1,"INVDATA:3"
```

will open the file "INVDATA" on drive three 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 buffer number. The filename itself appears only when you open the file.

CLOSE -

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

CLOSE bufnum,bufnum,bufnum...

"bufnum" is the buffer number you wish to close. You may specify up to 15. Bufnum may be between 1 and 15. If bufnum is omitted, all files will be closed.

If you attempt to close a buffer that was not opened, the statement will simply ahve no effect.

Examples -

CLOSE 1,2,3

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

CLOSE

will close all currently open files.

Note : Do NOT, under ANY circumstances, remove a diskette that has a file open on it. This is disastrous to correct system operation. The current buffer has not been written to the disk and the directory entry has not been updated until the file is closed.

The following actions will cause the files to close :

Function	Action
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.

or
Adding/Deleting program lines.
Use of certain extended Disk BASIC CMD"" features.

INPUT # -

Read from a sequential disk file.

INPUT# bufnum,var,var,var...

"bufnum" is the file buffer number of the file that has been opened for sequential input.

"var" 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.

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

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

Un-quoted string terminators
==========
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.

If a carriage return is followed by a linefeed, the linefeed will NOT be considered part of the terminator and will be included in the next input. This is different than TRSDOS Disk BASIC. Dosplus Disk BASIC does not output a linefeed after every carriage return during sequential I/O.

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 from.

Examples -

```
SI        INPUT# 1,A
```

will attempt to read the numeric variable "A" from the file that is defined for buffer number one.

```
INPUT# 2,A$,B#,C%,D$
```

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 # -

Inputs a complete line of text sequentially from the disk.

LINE INPUT# bufnum,var

"bufnum" is the file buffer number of the file that has been opened for sequential input.

"var" 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 a "line" of string data into the specified variable. This comes in handy when you want to read in text of some variety and ignore the usual terminators.

Line input will read everything from the first character of the file up to either :

* 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 into memory as data.

Example -

LINE INPUT# 1,D$

will read from the file assigned buffer one and store the data read in the variable in "D$".

Technical note : 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. This is different than TRSDOS. Dosplus does not output a linefeed after every carriage return during sequential I/O. Other than this small difference, the two system handle sequential disk I/O in exactly the same manner, thus assuring program compatibility.

PRINT # -

This command will do a write to a sequential disk file.

PRINT# bufnum,USING format$,var,del,var...

"bufnum" is the file buffer number of the file that has been opened for sequential output.

"USING format$" 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.

"var" is the variable that contains the data you wish to write.

"del" is the delimiter that must be placed between each variable that is to be written. This should 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 works to the disk in a manner identical to that of the PRINT command that you use to write to the screen. As a matter of fact, the code that handles PRINT# is the same code as handles PRINT, the I/O DCB is simply re-routed to a disk file. To illustrate this, open the screen for sequential output (this is allowed in Dosplus because the screen is viewed as simply another logical device. i.e. OPEN"O",1,"*DO"). When you do a PRINT#, you will see that in operates in a manner exactly the same as a standard PRINT to the screen.

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 only one space in between. That is why semi-colons are the preferred delimiter.

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 #1,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 filed to be used for the output data format. For example (when A=123.456) :

```
PRINT #1,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 -

This will allow you to organize a random file buffer into fields defined by variables so that I/O to the file can begin.

FIELD num1,num2 AS var1$,num3 AS var2$...

"num1" is the file buffer number specified at time of OPEN.

"num2" defines the field length for the first field.

"var1$" is the variable name for the first field.

"num3" is the field length for the second field.

"var2$" is the variable name for the second field.

You may specify as many fields as you have need for. Before you can field a file, you must first open it via OPEN"R". This assigns it the buffer number specified during a field (i.e. num1). 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 appear in the standard string variable area and consequently does not use up any string space. The pointer for these 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 say 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.

Examples -

```
FIELD 1,23 AS A$,24 AS B$,132 AS NAME$

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 in from disk.

GET num1,num2

"num1" is the file buffer number.

"num2" 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 file. After all this, you may get a record and make effecient 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. This is convenient for when you know that you are going to read every record of a random access file in order, you may use a FOR-NEXT loop until you have read up to the end of file.

Example -

```
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 -

Write a record to disk.

PUT num1,num2

"num1" is the file buffer number.

"num2" is the specific logical record number that you wish to write to. 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. In the case, this feature could be used for when you wish to pre-allocate and clear out space in a particular file.

Example -

PUT 1,2

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 in a random file buffer.

LSET/RSET var$=exp$

"var$" is the variable that has been defined via the field statement.

"exp$" is the expression for evaluation.

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 both, if the string is too large, it will be truncated on the right hand side.

The way that they pad the data into the buffer is the ONLY difference between LSET and RSET.

Example -

LSET A$=FD$

this takes the data currently in FD$ and left justifies it into the file variable A$.

RSET A$=FD$

this will do the identical thing 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$.

MKI$, MKS$, and MKD$ -

Place numeric data into a string for entering into a random file buffer.

MKI$(exp), MKS$(exp), or MKD$(exp)

"exp" is the expression to be evaluated for conversion.

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

Example -

```
LSET A$=MKI$(12)
```

"A$" will now contain a two byte representation of the integer value twelve.

```
LSET 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, the way to recover it is via the conversion statements CVI, CVS, and CVD.

Note : In the case of ASCII data, no conversion is needed going in or coming out. For example, you would simply say "LSET A$=ZP$" (assuming ZP$ to be a standard string variable), and then PUT the buffer to disk.

CVI, CVS, and CVD -

Restore string data to numeric form.

CVI(var$), CVS(var$), or CVD(var$)

"var$" is the string variable that the data was placed into.

These functions are used to retrieve data after it has been stored into string form by the commands just covered. They are the exact inverse of their 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 they are greater, extra characters will be ignored.

Function	Length of string
CVI	Two byte
CVS	Four byte
CVD	Eight byte

They can have more characters, but not less.

Examples -

In the previous section we showed the example "LSET A$=MKI$(12)".

```
DEC%=CVI(A$)
```

the variable "DEC%" would become equal to twelve.

```
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$". It is an eight byte double precision value. You then have a single precision value for the year to date taxes in the variable "TT!". If you try to "NET!=YTD$-TT!", you will get a "Type mismatch" error. After all, YTD$ IS a string.

However, you can say "NET!=CVD(YTD$)-TT!" and get a

perfectly acurate figure.

You can also print the conversion value directly or assigning the value to a temporary variable and print that.

Note : In the case of ASCII string data, no conversion is needed going in or coming out. You would simply say "ZP$=A$" (if A$ was an ASCII file variable).

EOF -

End of file detection

EOF(bufnum)

"bufnum" is the file buffer number of the buffer 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 those "Input past end" errors in sequential input.

As long as "bufnum" is the buffer number for an active (open) file, then EOF will return a 0 if the last record has not been read and a -1 if it has.

For example, you could have a program that looks like 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. If you were to break the program and "PRINT EOF(1)", the corresponding value (0 or -1) would appear on the screen.

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 -

Get highest logical record number.

LOF(bufnum)

"bufnum" 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 has no way of knowing how the data is on the disk. Because of this, LOF simply returns the number of physical sectors.

Often in random access, you wish to look at every record in a file. That program would look something like this :

```
10 OPEN"R",1,"NAME/INX:2",12
20 FIELD 1,10 AS NAME$,2 AS REC$
30 FOR I=1TOLOF(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 -

Returns actual record number of record last accessed.

LOC(bufnum)

"bufnum" 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 arguement.

You can also say "PRINT LOC(1)" as you are accessing file buffer one and have a dynamic display of what records the program is actually reading from the file.

Example -

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 accessed in file buffer number two.

Note : This command is only accurate when working with random files. It deals with logical records.

Disk BASIC error codes -

The Dosplus Disk BASIC error codes are, for compatibility's sake, the same as those of the other operating systems. They are as follows :

Code	Error
98	Undefined USR function
100	Field overflow
102	Internal error
104	Bad file number
106	File not found
108	Bad file mode
110	File already open
112	Unprintable error
114	Disk I/O error
116	File already exists
118	Unprintable error
120	Unprintable error
122	Disk full
124	Input past end
126	Bad record number
128	Bad file name
130	Mode mismatch
132	Direct statement in file
134	Too many files
136	Disk write protected
138	File access DENIED

Use of the above codes will enable you to "error-trap" your programs. The Level II manual documents the "ON ERROR GOTO" function and lists several codes you can use to catch and handle errors, thereby preventing program interruption. These codes should be added to that list.

There is a function in ROM BASIC called ERR. There is another one called ERROR. In order to make the codes returned by ERR produce the correct result when they were used with ERROR, you had to divide ERR by two and add one.

This practice has carried over into Disk BASIC. Be advised that ERROR will not return a Disk BASIC error.

Extended Disk BASIC features -

This section will cover the extended Disk BASIC features. The previous section dealt with those commands of Disk BASIC that are pretty much universal to all TRS-80 Disk BASIC's. The commands found in this section will not operate with other BASIC's (or at least not in the same exact manner). The commands covered are :

Command	Function
CMD"	DOS commands from BASIC
DI	Delete and Insert (BASIC program line)
DU	Duplicate (BASIC program line)
Edit	Shorthand editing 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)
ZAP	Change default files for BASIC and TBASIC
TBASIC	TBASIC error messages

CMD (Use DOS command from BASIC) -

CMD"DOS command"

"DOS command" is any legal syntax for any library command and some of the utilities.

The CMD function has been changed 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.

Example -

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.

Note : Use of the Dosplus library command CLEAR from BASIC is not recommended, and could cause erratic system operation. If not used carefully, you could clear out part of your program or part of BASIC itself.

You may use the following Dosplus utility programs from BASIC :

* CONVERT
* CRUNCH
* DISKDUMP
* MAP
* PURGE
* RESTORE
* SPOOL
* TRANSFER

Example -

CMD"CLEAR "+FS$+" (DATA=E5H)"

Assuming FS$="TEST/DAT", this statement will fill the fil "TEST/DAT" with "E5". 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.

Example -

```
CMD"FREE :2 (P)"
```

This will output a free space map of drive two to the line printer.

Example -

```
CMD"LOAD SUBROUT/OBJ:3"
```

This will load the file "SUBROUT/OBJ" and return control to BASIC. This is the equivalent of TRSDOS' CMD"L".

Example -

```
CMD"JOIN *DO *PR"
```

This will duplicate video output to the line printer. It is the equivalent of TRSDOS' CMD"Z".

DI (Delete and Insert BASIC program line) -

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) -

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.

EDIT (Shorthand EDITing commands) -

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:

```
; (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)
A              - Abbreviation for AUTO (A10,5)
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 error.

You may not use the abbreviations in a program line. They are only valid when they are the first character typed on a line.

Technical notes : If you type a character (other than a shorthand command) and then backspace, the shorthand commands will not work. They must be the FIRST character typed on a line, not just the beginning character. If you DO type another character and then have to backspace, press ENTER to receive a new line prompt before attempting a shorthand command. Also, use of the arrow keys to LIST lines mat result in erratic behavior if the lines exceed 240 characters. This is rare, but if it does happen, don't worry. The line is fine, but simply too long for the shorthand LIST to accurately display it.

RENUM (Renumber BASIC text) -

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-exsistent line number. When it finds it, it will print the error message. This feature checks the text only! It will not renumber it.

Example -

CMD"RENUM",!

This will check the text for any errors.

Example -

CMD"RENUM",10,5

This will renumber the entire text, calling the first line 10 and incrementing by fives. (10, 15, 20, 25, etc.)

Example -

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. The 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 was I was using, I could stop there.

TAB (TAB to the line printer) -

LPRINT TAB(210);A$

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)-

TRON or TROFF

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 the <ENTER> 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.

REF (Reference BASIC text) -

CMD"REF",par,par

"par" can be any of the parameters legal for this command.

Your legal parameters are :

Parameter	Function
S=	Single variable, line, or keyword
V	All variables
L	All line numbers
K	All keywords
P	Printer output

This will allow you to reference your BASIC program for line numbers (L), variables (V), or keywords (K). For example :

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.

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 occurences 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.

Technical note : The referencer will display all variables specifying variable type and whether or not it is an arrayed variable. If it occurs mutiple times in a line, it will reference it multiple times. Also, any ASCII number in text will be treated as a line number during a reference.

CMD"M" (Dynamic variable dump) -

This feature will instantly display all the simple variables currently allocated, and what they are set to. All you have to do from the BASIC command mode is type :

```
CMD"M"
```

The computer does the rest. 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. If, for example, A$ is equal to CHR$(23) and you do a CMD"M", the screen will go into the 32 character mode as CHR$(23) will be printed during the dump.

Typing :

```
CMD"M",P
```

will dump the variables to the printer.

If the screen seems to go crazy, it is probably due to the fact that there are control codes imbedded in a string variable somewhere.

It will not display arrays. Simple variables only.

SR (Global editing of BASIC text) -

CMD"SR",sexp,rexp,sln-eln

"sexp" is the search expression. This can be any valid string expression, a literal, or a combination of both string variables AND literals.

"rexp" is the replace expression. This can also be any valid string expression, literal, or combination.

"sln-eln" starting line number 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. All you have to do is 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. Note : The starting line number is inclusive, the ending line number is not. That means that this utility will check the starting line number but will stop at the line BEFORE the ending line number.

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.

PHYSICAL DEVICES -

You may now specify a device instead of a file when doing a LOAD, a SAVE, an OPEN"I", or an OPEN"O". This means that you can type:

SAVE"*RO" or LOAD"*RI"

and load and save in and out the RS232. You can also do an :

OPEN"I",1,"*RI" or OPEN"O",1,"*RO"

and do sequential I/O in and out the RS232.

You may only open a device for sequential I/O. Random I/O to a logical device is impossible. When you have completed all I/O and you CLOSE the device, Dosplus sends a control<D> (ASCII 04) for end of text.

Your list of devices is detailed under the DOS library commands FORCE and JOIN. As an example, suppose one has a BASIC program and wish to transfer it to a friend who also has Dosplus 3.4/4.0. All he must do is call his friend and establish modem carrier. Once the computers are linked, he would simply :

SAVE"*RO"

while his friend had already done a :

LOAD"*RI"

This would, assuming no transmission problems, send the program directly into the other machine's memory.

CMD"O" (Sort BASIC arrays) -

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 indiactes 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 appended 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 arrrays.

Exception to the "+" or "-" appendage is the PRIMARY KEY ARRAY. Primary key array is seperated 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 ney 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.

Where we differ from TRSDOS is capacity and order. In Dosplus' CMD"O", you may indicate ascending or descending order on a tag array. Also, Dosplus' CMD"O" allows you up to five key arrays (counting primary key array) and fifteen tag arrays for a total of TWENTY arrays.

Technical notes : This sort use memory from hex 5200 to hex 56FF. This is the DOS command overlay from Disk BASIC. That means you cannot operate this sort from TBASIC. This sort will function with any version of Model III Dosplus (not just 3.4/4.0). 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.

Application -

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$,20 AS NAME$
30 EF=LOF(1):DIM A$(EF),RN%(EF)
40 FOR I=1TOEF
50 GET 1,I
60 A$(I)=NAME$: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.

INPUT@ (Controlled screen input) -

INPUT@<pos>,pr$,fl,"it";var$

"<pos>" is the screen position you wish to input at. It will begin here with the prompt string if one is specified.

",pr$" is the prompt string you wish to have displayed on the screen in front of your input field. This must be a literal.

",fl" is an integer expression that defines field length.

',"it"' item type flag. Should be "$" for alphanumeric or "#" for numeric. If you append an asterisk to this, you set the "return on full field" mode. This may be a literal OR an expression.

";var$" string variable that data typed into the input field is passed to BASIC in. Must be a string even if input was restricted to numeric only. Note that this option is seperated 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. This will banish that.

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.

",pr$" (Prompt string). This must be a literal string. 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 there 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.

',"ft"' (Field 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. This preserves the form you have created on the screen by stopping the line feed after input. If ENTER is pressed alone, the return variable will be ASCII 13 and if it was CLEAR, ASCII 31. You can check this by asking "IF var$=CHR$(13)" or "IF var$=CHR$(31)".

";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.

Technical notes : This is really more of a LINE INPUT@ than an INPUT@, but LINE INPUT is a longer and more cumbersome keyword. What this means is that we will be dealing with all STRING data. This is preferable anyway for control of input. With INPUT@, you can limit only to alphanumeric or numeric. This is useful, but you are still responsible for the bulk of the error checking. Also, while inputting data, you may use the following :

* Repeating keys.
* Backspace.
* Erase line (shift back arrow).

TBASIC (Tiny Disk BASIC error codes) -

TBASIC has simplified two-letter error messages in addition to the LEVEL II error messages. They are:

AD - File access denied (protected file)
AO - File already open
BM - Bad file mode
BN - Bad file number
DF - Disk full
DS - Direct statement in file
EF - End of file encountered
FE - File already exists
FF - File not found
FL - Too many files (you need to open more files)
FO - Field overflow
IE - Internal error
IO - Disk I/O error
MM - Mode mismatch (sequential/random files)
NM - Bad file name
RN - Bad record number
UE - Undefined error
UF - Undefined user function
WP - Disk write protected

TBASIC gives you the maximum BASIC compatibility that we can offer. "TBASIC filename-F:3" will run almost ANYTHING. When you are having compatibility problems, try that solution before calling in the troops.

NUMBER OF FILES -

The following information is pertinent to those of you who wish to change the number of default files that you use coming into BASIC.

These changes are good for Dosplus 3.4/4.0, BASIC 1.6, and TBASIC 1.5 ONLY!! Using them on Dosplus 3.3 or BASIC 1.4 is a sure-fire way to destroy your program.

For BASIC 1.6 :

1. Type DISKDUMP BASIC/CMD.XANTH

2. Type "G A" <enter>.
(No space between the "G" and the sector number)

3. Press "M" for modify.

4. Examine relative byte B2 hex (Model III)
or
9F hex (Model I)

   It should be 00. If it is not, you are not at the right location and you need to examine carefully the sector number.

5. Change it to the value you wish (00-0F).

   Remember that you are dealing with hexidecimal numbers. You can only have a maximum of fifteen files altogether. (For example, 3 files would be 03, and 10 files would be 0A, etc., etc.)

For TBASIC 1.5 :

1. Type DISKDUMP TBASIC/CMD.XANTH

2. Type "G 6" <enter> (Model III)
or
"G 5" <enter> (Model I)
(No space between the "G" and the sector number)

3. Press "M" for modify.

4. Examine relative byte 06 hex (Model III)
or
EF hex (Model I)

   It should be 00. If it is not, you are at the wrong location and you need to go back and carefully examine the sector number.

5. Change it to the value you desire (00-0F).

   Remember that you are dealing with hexidecimal numbers. You can only have fifteen files altogether. (For example, 3 files would be 03, 10 files would be 0A, etc., etc.)

Please do not attempt this if you are unfamiliar with machine language and modifying programs. This is NOT mandatory. You can enter BASIC with the before specified syntax (see Entering BASIC). This should only be done by the experienced user.

Patch program instruction sheet

1. Getting started -

To enter the program type the following :

```
BASIC PATCH/BAS-F:1
```

The screen will clear and the program header will display. Make sure the Model number matches the machine you are working on.

2. Operation -

There will be a menu of programs that you can patch. Beside each choice there will be a number. This is the number that you enter for the "Select ?" prompt. If you enter a zero, the program will end and return you to Dosplus.

When you select the number of the option that you want, the word "Select ?" will be replaced by "Filespec ?". This is looking for the filename of the program that you want to patch. (i.e. SCRIPSIT/CMD, VC/CMD, INIT, etc., etc.) It will NOT copy the file over, nor is this filspec the name of what you would LIKE to call it. We patch the program leaving the filename and program location alone.

If you have forgotten what you have named the program, you may type in "DIR" when it asks for "Filespec ?". This is ONLY valid under Extended Disk BASIC, it is NOT allowed under TBASIC. It will ask "Which drive ?". You respond with the drive number, and it will respond with the proper directory.

When it returns to the "Select ?" prompt and prints the message "Patch complete.", you have installed the patch. You may, at this point, exit and run your patched program

NOTE :

Please make certain you patch the INIT file in PROFILE Model III and not PROFILE/CMD. This is the only file that needs a patch.

Patches to MicroSoft's BASIC and FORTRAN compilers are available on request. Patches are also available for other programs. Scripsit Ver 3.2 does not need a patch to run with Dosplus. Only use the Scripsit patch on Scripsit Ver 1.0 or 3.1.

Many times the version of the program that you have will run just fine. If this is the case, don't worry about it. Often only certain versions of a particular program are incompatible. MOST programs WILL run "as-is". The ones that don't usually are incompatible in either the end of file handling, or in calling a directory from program.

Before you assume anything, give us a call. We have people available from 10 in the morning till 4:30 at night (Eastern Standard Time) to help you. Call us at (305) 983-3390.

Please don't call us and ask for a "list" of patches. There is no such thing. More and more programs every day are becoming compatible with Dosplus. We feel that to publish such a list would be a Herculean feat inasmuch as we would be updating it daily. If you want a patch for a specific program, let us know. Maybe it has already been changed (by the author!) to run.

Also, many times we have a version of the program here that has been patched for operation with Dosplus before we started recording patches. In that case we will require that the a copy of program be sent in for replacement (so that we do not violate any copyright laws). We regret the extra inconvenience, but it is necessary.

Thank you,
Micro-Systems Software Inc.
Technical Support Division.

```
0 '       TESTPGM - Variable length record example
          Created : 05/14/81  MRL/MSS

10 CLEAR 500:DEFSTR A,Z:DEFINT I,N
20 Z1=CHR$(30):Z2=CHR$(31):Z3=CHR$(27):Z4=CHR$(140)
30 CLS:PRINT"Testpgm Ver 1.0":PRINT"By Micro-Systems
Software Inc."
40 '
50 '      Main menu
60 '
70 PRINT@45,Z1;"Main menu";:PRINT@256,Z2;
80 PRINTTAB(10)"1. Input new records":PRINTTAB(10)"2.
Review old records":PRINTTAB(10)"E. End session":PRINT
90 PRINTTAB(10)"Selection ? ";:LINE INPUT A
100 IF A="E" OR A="e" THEN CMD
110 IF A<>"1" AND A<>"2" THEN PRINT Z3;Z1;:GOTO 90
120 ON VAL(A) GOTO 160,550
130 '
140 '     Add new records
150 '
160 OPEN"R",1,"TEST/INX",4
170 OPEN"R",2,"TEST/DAT",10
180 FIELD 1,2 AS A1,2 AS A2
190 FIELD 2,10 AS A3
200 GET 1,1:I1=CVI(A2)
210 '
220 '     Input data
230 '
240 PRINT@45,Z1;"Add records";:PRINT@256,Z2;
250 '
260 LINE INPUT"Record number ? ";Z
270 IF LEN(Z)>4 THEN PRINTZ3;Z1;:GOTO 260
280 IF Z="" THEN LSET A1=MKI$(0):LSET A2=MKI$(I1):PUT
1,1:CLOSE:GOTO 70
290 '
300 '     Search index
310 '
320 LQ=I1+1:XQ=VAL(Z):FF%=0:GOSUB 940
330 IF FF%=1 THEN PRINT"Number already in
file.":PRINTZ3;Z3;Z1;:GOTO 260
340 '
350 '     If not in file, bump index
360 '
370 FOR I=I1+1 TO MQ STEP-1:GET 1,I:PUT 1,I+1:NEXT I
380 '
390 '     Put index record
400 '
410 I1=I1+1:LSET A1=MKI$(VAL(Z)):LSET A2=MKI$(I1):PUT 1,MQ
420 '
430 '     Input data to be filed
440 '
450 LINE INPUT"Remark (10 char. max.) ? ";Z
460 IF LEN(Z)>10 THEN PRINTZ3;Z1;:GOTO 450 ELSE LSET A3=Z
470 PUT 2,I1
480 '
490 '     Loop back for next
```

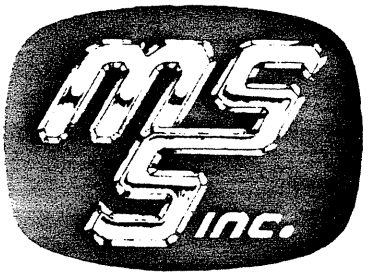
```
500 '
510 PRINT@256,Z2;:GOTO 260
520 '
530 '    Review old records
540 '
550 PRINT@45,Z1;"Review records";:PRINT@256,Z2;
560 OPEN"R",1,"TEST/INX",4
570 OPEN"R",2,"TEST/DAT",10
580 FIELD 1,2 AS A1,2 AS A2
590 FIELD 2,10 AS A3
600 '
610 '     Get record number and search index
620 '
630 LINE INPUT"Record number ? ";Z
640 IF LEN(Z)>4 THEN PRINTZ3;Z1;:GOTO 630
650 IF Z="" THEN CLOSE:GOTO 70
660 '
670 GET 1,1:I1=CVI(A2)
680 LQ=I1+1:XQ=VAL(Z):FF%=0:GOSUB 940
690 IF FF%=0 THEN PRINT"Not in file.":PRINTZ3;Z3;Z1;:GOTO
630
700 '
710 '     If found
720 '
730 GET 2,CVI(A2)
740 '
750 '     Display and edit (if needed)
760 '
770 PRINT"Remark --> ";A3
780 LINE INPUT"Change (Y/N) ? ";Z:IF LEN(Z)>1 THEN
PRINTZ3;Z1;:GOTO 780
790 IF Z="Y" OR Z="y" THEN PRINTZ3;Z1;:LINE INPUT"New
remark (10 char. max.) ? ";Z ELSE GOTO 880
800 IF LEN(Z)>10 THEN Z="Y":GOTO 790
810 '
820 '     Save changes
830 '
840 LSET A3=Z:PUT 2,LOC(2)
850 '
860 '     Loop back for next
870 '
880 PRINT@256,Z2;:GOTO 630
890 '
900 ' Subroutine to do HI-LO search
910 ' Inputs LQ (list length) and XQ (search variable)
920 ' Returns MQ (insert slot)
930 '
940 MQ=INT(LQ/2)
950 IF MQ=0 THEN GOTO 1010
960 GET 1,MQ+1:IF CVI(A1)=XQ THEN FF%=1:GOTO1020
970 IF CVI(A1)<XQ THEN GOTO 980 ELSE LQ=MQ:GOTO 940
980 TQ=LQ-MQ
990 TQ=INT(TQ/2)
1000 IF TQ=0 THEN 1010 ELSE MQ=MQ+TQ:GOTO 960
1010 MQ=MQ+1
1020 MQ=MQ+1:RETURN
```

```
10 '     Directory Subroutine
20 '     Created 06/09/81  MRL/MSS
30 '
40 CLEAR 1600:DIM A$(128)
50 '
60 '             Input drive and open dir
70 '
80 INPUT"Which drive
";DR$:DR$=":"+DR$:OPEN"R",1,"DIR/SYS"+DR$
90 '
100 '             Read sector
110 '
120 J=0:FOR I=3TOLOF(1):GET 1,I:FOR II=0TO7
130 FIELD 1,(II*32) AS D$,1 AS A$,4 AS D$,8 AS FS$,3 AS FE$
140 '
150 '             Check for sys and del files
160 '
170 IF NOT(CVI(A$+CHR$(0))AND208)=16 THEN220
180 '
190 '             Build dir array
200 '
210 A$(J)=FS$+" "+FE$:J=J+1
220 NEXT II,I:CLOSE
230 '
240 '             DONE - print results
250 '
260 FOR I=0TOJ:PRINTA$(I),:NEXT
```

```
10 '    Program to input the date from BASIC
20 '    Valid for Extended Disk BASIC ONLY!!
30 '    Created : 09/22/81  MRL/MSS
40 '
50 '
60 '             Input date
70 '
80 LINE INPUT"Input today's date (MM/DD/YY) ? ";DT$
90 '
100 '             Interrogate for correct form
110 '
120 IF LEN(DT$)<>8 THEN GOTO 150
130 IF MID$(DT$,3,1)<>"/" OR MID$(DT$,6,1)<>"/" THEN GOTO
150
140 GOTO 230
150 PRINT"*** Bad format ! ***":GOTO 80
160 '
170 '             Pass date to Dosplus system
180 '
190 '   Please note that the space after the word DATE in
200 '   CMD"DATE " is required by the system and may not
210 '   be deleted for any reason.
220 '
230 CMD"DATE "+DT$
```

MICRO-SYSTEMS SOFTWARE·INC

5846 Funston Street Hollywood, FL 33023
(305) 983-3390



Technical Section Ver T.2
for
DOSPLUS Ver 3.4/4.0
Disk Operating System

by Micro-Systems Software Inc.

I N D E X

Physical and Logical Records -

Dosplus deals with two types of data records :

(1) Physical Records

(2) Logical Records

A physical record is very simply a single disk sector. A sector, and therefore a physical record, is exactly 256 bytes in length.

A logical record may be of length from 1 to 256 bytes. The length of a logical record, called the Logical Record Length or LRL, is established by the user at the time the file is created. If the LRL < 256, Dosplus will automatically perform record "blocking"; i.e. Dosplus will fit as many logical records as possible into a physical record, and even span sector boundaries where necessary.

The relationship between a logical record and the physical record upon which it begins is as follows :

PR = INT ((RN x LRL)/256) + 1

Where PR = Physical Record Number
RN = Logical Record Number
LRL= Logical Record Length of file

Disk Storage Allocation -

The smallest unit of disk space that can be allocated to a file is called a GRANULE, or GRAN. All files occupy a whole number multiple of one gran. The actual size of a granule depends on the type of diskette size (5 1/4" or 8") the recording density (single or double density) and the number of recording surfaces used (1 or 2). The following table describes granule size for various diskette formats :

Disk Format	Granule Size
5" Single Density Single Side	5 Sectors
5" Double Density Single Side	6 " "
5" Single Density Double Side	5 " "
5" Double Density Double Side	6 " "
8" Single Density Single Side	8 " "
8" Double Density Single Side	6 " "
8" Single Density Double Side	8 " "
8" Double Density Double Side	10 " "

Granule size on a hard drive is user-defined.

Device Control Blocks (DCB's) -

A device control block is a section of RAM used to manage data transfer between the CPU and a device (display, keyboard, printer, RS232) or a disk file. The first byte of any DCB contains the DCB TYPE. The DCB type is represented as follows :

Bit	Meaning (when set)
0	Input device
1	Output device
2	CTL. Not implemented.
3	Reserved
4	FORCE in effect
5	JOIN in effect
6	Reserved
7	File DCB and/or FORCE or JOIN active

File DCB's -

The file DCB is a 32-byte area of memory specified by the user. Before OPENing a file and after CLOSEing it, the DCB contains a left-justified representation of the filespec, terminated with any control character (A carriage return, 0DH, is preferred).

Example DCB's -

```
BASIC/CMD.PASSWORD:0 <cr>
FILEDATA <cr>
DIR/SYS <cr>
```

Where "<cr>" denotes a carriage return, 0DH.

After OPENing a file and before CLOSEing it, the DCB contains the following information :

Address : DCB + 0
Length : 1 Byte
==========

DCB type.

Address : DCB + 1
Length : 2 Bytes
==========

Reserved.

Address : DCB + 3
Length : 2 Bytes
==========

Pointer to 256 byte file I/O buffer. The location of this buffer is specified by the user at file OPEN time.

Address : DCB + 5
Length : 1 Byte
==========

Offset to beginning of next logical record within the current physical record. For instance, if a file with a logical record length of 27H is OPENed, and the first record READ, then this byte will be contain a 27H. After the second record is READ, the byte will be 4EH. If the beginning of the next logical record lies at the start of the next physical record, a 0 will be stored in this location.

Address : DCB + 6
Length : 1 Byte
==========

DCB + 6 contains information about the drive on which a file is resident.

Bit	Contents
0-2	Drive #
3	Reserved
4	Side Select. Reset=Side A, Set=Side B (Used in earlier versions of Dosplus. Side select is ignored in 3.4/4.0)
5	Reserved
6	Drive Type. Reset=5 1/4", Set=8"
7	Density Select. Reset=Single, Set=Double

Address : DCB + 7
Length : 1 Byte
==========

Reserved.

Address : DCB + 8
Length : 1 Byte
==========

End-of-file byte. This byte indicates how far the file extends into the last sector allocated to it. For instance, if a file existed with a LRL of 67 and was 125 records in length, the EOF byte would be 183, the byte following the last byte of the file.

Address : DCB + 9
Length : 1 Byte
==========

This byte contains the logical record length that you specified when you OPENed the file. A value of 0 indicates a LRL of 256.

Address : DCB + 10
Length : 2 Bytes
==========

DCB + 10 indicates the record number of the next physical record in the file. This is called the Next Record Number, or NRN. For instance, if a file of LRL=98 was OPENed and logical record 213 READ, the NRN would be 83 decimal.

Address : DCB + 12
Length : 2 Bytes
==========

This location contains the Ending Record Number, or ERN. The ERN is simply the record number of the last physical record in the file.

Address : DCB + 14
Length : 17 Bytes
==========

Reserved.

If the user intends to manipulate a DCB for an OPEN file, great caution should be exercised. Dosplus requires DCB information to properly maintain its files, and altering the DCB contents may cause wildly unpredictable results.

I/O DCB's -

System I/O devices also require DCB's that contain useful information. They are listed below.

Keyboard DCB : 4015H

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver Address
DCB + 3	1	Reserved
DCB + 4	1	Caps lock. Reset=No Caps, Set=Caps
DCB + 5	1	Cursor Blink Count
DCB + 6	1	Cursor Blink Status. Z=Off, NZ=On.
DCB + 7	1	Cursor Blink. Z=Blink, NZ=No Blink

DCB + 6 and DCB + 7 are not used as shown on the Model-I.

Display DCB : 401DH

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver Address
DCB + 3	2	Cursor Position
DCB + 5	1	Cursor Flag. Z=Off, NZ=On
DCB + 6	1	Cursor Character
DCB + 7	1	Spec. Char./Tabs Flag. Z=Tabs, NZ=Spec

DCB + 6 and DCB + 7 are not used as shown on the Model-I.

Printer DCB : 4025H

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver Address
DCB + 3	1	Page length in lines
DCB + 4	1	Line counter
DCB + 5	1	Lines per page
DCB + 6	1	Characters per line
DCB + 7	1	Character counter

RS232 Input DCB : 41E5H (Model-III only)

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver address
DCB + 3	1	Input buffer
DCB + 5	1	Status flag. Bit 1 : Set = Wait Reset = No Wait Bit 2 : Set = Driver Active Reset = Driver Inactive

RS232 Output DCB : 41EDH (Model-III only)

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver address
DCB + 2	1	Output buffer
DCB + 3	1	Status Flag Bit 1 : Set = Driver Active Reset = Driver Inactive

RS232 Initialization DCB : 41F5H (Model-III only)

Address	Length	Contents
DCB + 0	1	DCB type
DCB + 1	2	Driver Address
DCB + 3	1	Baud Rate
DCB + 4	1	Config. Code
DCB + 5	1	Wait flag Z=No wait, NZ=Wait

Operation of FORCE and JOIN -

FORCE and JOIN are two of the most powerful and flexible features of Dosplus. With these commands it is possible to re-direct data flowing between a device (CRT, printer, keyboard, etc.) and another device or file. Dosplus makes use of the DCB type byte in the device or file DCB to flag an active FORCE or JOIN. If the printer (*PR) had been forced to the display (*DO) by typing :

```
FORCE *PR *DO
```

the printer DCB located at 4025H would be altered. Specifically, the DCB type would now have bits 4 and 7 set, indicating that a FORCE is in effect. DCB + 1 and DCB + 2, which usually contain the address of the device driver, will now contain the address of the display (*DO) DCB, 401DH.

In general, a FORCE will set bits 4 and 7 of the source device, and will replace the driver address with the address of the DCB of the destination device.

JOIN operates in a somewhat more complex manner. JOIN creates a "dummy" DCB which stores both the driver address of the source device and the DCB address of the destination device. DCB + 1 and DCB + 2 of the source device point to the dummy DCB, and the source DCB type has bits 5 and 7 set.

For instance, if we were to JOIN *DO PRT/TXT, the DCB type for *DO (located at 401DH) would have bits 5 and 7 set, indicating that a JOIN is active. Locations 401EH and 401FH would contain the address of the dummy DCB; let's say 4296H. In the dummy DCB, DCB + 1 and DCB + 2 (4297H and 4298H in the example) would contain the display driver address. DCB + 3 and DCB + 4 (4299H and 429AH) would hold a pointer to the DCB of the file PRT/TXT.

System Vectors -

This section will provide information to enable the programmer to utilize a wealth of built-in routines in Dosplus. Use of these routines should allow the programmer complete freedom from writing actual disk I/O code, and lighten the burden of writing many other commonplace routines.

File I/O Routines -

INIT : 4420H

INIT will take a valid filespec and search all drives in an attempt to find the file (unless a specific drive spec is supplied). If the file is found, it is OPENed for I/O. If the file is not found, INIT will create it on the first available drive (unless a drive is specified) and OPEN it for I/O.

INIT may also be used to OPEN a device for I/O. In this case, the DCB should contain the device specification (*KI, *DO, *PR, *RO, *RI). The contents of HL and B are ignored. Only character (single-byte) I/O is allowed to/from a device, therefore READ and WRITE are not allowable I/O operations for a device. GET and PUT are to be used for device I/O.

An OPEN or INIT on a device spec will produce a 10-byte DCB. The first 8 bytes of this DCB are a duplicate of the standard DCB (without FORCEs or JOINs) for that device. The last two bytes contain the device name (KI, DO, PR, RO, RI).

ENTRY : HL=> 256 byte file buffer.

DE=> File/Device DCB

B = File LRL.

ENTER : CALL 4420H

EXIT : Z flag status :
SET - No error.
RES - Error has occured. Error code contained in register A.

C flag status :
SET - File was created.
RES - No file created.

OPEN : 4424H

OPEN will search all available drives for a filespec (unless a drive is specified) and if found, OPEN the file for I/O. If the file is not found, OPEN will return an error message. OPEN does not create a file; INIT must be used for that.

OPEN may be used with a device as well as a file. In this case, the DCB should contain a device spec. See notes on device I/O in the INIT section.

ENTRY : HL=> 256 byte file buffer.

DE=> File/Device DCB.

B = LRL

ENTER : CALL 4424H

EXIT : Z flag status :
SET - No error.
RES - Error has occured. Error code is in register A.

CLOSE : 4428H

CLOSE writes any data remaining in the disk I/O buffer to disk and updates the file's directory entry. CLOSE should be called if any data has been written to a file, but it is not necessary if a file has ONLY been read. If a disk I/O error occurs, it is recommended that the file NOT be closed in order to avoid the possibility of writing bad data to the file.

CLOSE may also be used on an open DCB used for device I/O. It is not necessary to CLOSE a device, but the operation is allowable in order to maintain compatibility with file I/O. A CLOSE performed on a device will result in a CTL-D being output to that device.

ENTRY : DE=> File/Device DCB

ENTER : CALL 4428H

EXIT : Z flag status :
SET - No error.
RES - Error has occured. Error code is contained in register A.

After a CLOSE, the DCB once again contains the filename, extension and drivespec of the file, BUT NOT THE PASSWORD.

KILL : 442CH
==========

KILL de-allocates the disk space occupied by a file, and deletes the file's directory entry. KILL cannot be performed on a non-open file.

ENTRY : DE=> File DCB.

ENTER : CALL 442CH

EXIT : Z flag status :
SET - No error.
RES - Error has occurred. Error code is contained in register A.

LOAD : 4430H
==========

LOAD will search for and load into RAM a Z-80 object file stored in load module format, such as those produced by the DUMP command or a Z-80 assembler.

ENTRY : DE=> File DCB.

ENTER : CALL 4430H

EXIT : DE is destroyed

HL= Program entry address

Z flag status :
SET - No error.
RES - Error has occurred. Error code is in register A.

RUN : 4433H
==========

RUN functions in a manner similar to LOAD; It searches for a Z-80 object file and loads it into RAM. RUN then transfers CPU control to the loaded program at the file's transfer address.

ENTRY : DE=> File DCB.

ENTER : JP 4433H

READ : 4436H
==========

This routine will read a logical record from a disk file into RAM. The value of LRL will determine where the record will be placed in RAM. If the LRL < 256, then the record will appear in a buffer area specified by the user at the time of the READ. If LRL = 256, the record will be placed in the 256 byte I/O buffer specified at the time of file OPEN or INIT, and also contained in the file DCB.

ENTRY : DE=> File DCB.

IF LRL < 256, then HL=> User record buffer;
OTHERWISE, HL is ignored.

ENTER : CALL 4436H

EXIT : Z flag status :
SET - No error.
RES - Error has occurred. Error code is in register A.

WRITE : 4439H

WRITE will take a logical record from a RAM buffer and place it into a disk file. Like READ, the area of RAM referenced is dependent upon the value of LRL. If the file LRL < 256, the buffer area is specified by the user on entry to WRITE. If LRL = 256, then the 256 byte I/O buffer created at OPEN or INIT is used. The address of that buffer may be found in the file's DCB.

ENTRY : DE=> File DCB.

IF LRL < 256, then HL=> User record buffer;
OTHERWISE, HL is ignored.

ENTER : CALL 4439H

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Register A contains error code.

VERF : 443CH
==========

VERF performs the same function as WRITE, with one addition. After a physical record is written to a file with VERF, the record will be read back, and the CRC's compared. If the two CRC's are not identical, Dosplus will report an error.

ENTRY : DE=> File DCB.

IF LRL < 256, then HL=> User record buffer;
OTHERWISE, HL is ignored.

ENTER : CALL 443CH

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Error code is contained in register A.

REW : 443FH
==========

REW will position a file's next record pointer to the first logical record in the file.

ENTRY : DE=> File DCB.

ENTER : CALL 443FH

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Register A contains eror code.

POSN : 4442H
==========

POSN will adjust a file's next record number pointer to any desired logical record. POSN allows the user to access file data in a random order, as opposed to sequential access.

ENTRY : BC = Logical record number.

DE=> File DCB.

ENTER : CALL 4442H

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Error code is in register A.

BKSP : 4445H
==========

BKSP allows the programmer to "back up" to the previous logical record in a file. For instance, if the current logical record is #38, BKSP would set the next record to be read or written equal to #37. If the current record the the first record in the file, Dosplus will return an error, and the pointer position will be set to FFFFH, the highest possible record number.

ENTRY : DE=> File DCB.

ENTER : CALL 4445H

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Error code is in register A.

PEOF : 4448H
==========

PEOF positions the next record pointer to the record following the last record currently in the file. PEOF is useful for extending or appending sequential files.

ENTRY : DE=> File DCB.

ENTER : CALL 4448H

EXIT : Zero flag status :
SET - No error.
RES - Error has occurred. Register A contains error code.

Miscellaneous Routines -

EXIT : 402DH

A jump to this address will result in an exit to the Dosplus command level.

ENTER : JP 402DH

ABORT : 4030H

Programs encountering an error may jump to this address instead of 402DH. If ABORT is used rather than EXIT, Dosplus will enter DEBUG if it is activated. Otherwise, ABORT will function as EXIT.

ENTER : JP 4030H

CMD : 4400H

A jump to CMD will result in a normal exit to the Dosplus command level, and is equivalent to the EXIT routine.

ENTER : JP 4400H

CMNDI : 4405H

This is the Dosplus command interpreter. It will allow a user program to execute Dosplus library commands and functions. After the command is executed, CMNDI will return control to the calling program. If an error is encountered during CMNDI execution, the routine will exit to DOS command level through the ABORT vector.

NOTE : Most Dosplus library commands make use of the DOS overlay area located from 5200H - 56FFH. If the user is has located a program in that area, execution of a library command may overlay the user program.

ENTRY : HL=> User command buffer. Should contain a legal Dosplus command line, terminated with a carriage return, 0DH.

ENTER : CALL 4405H

ERROR : 4409H

Prints error message on video display, although the message may be re-routed by use of the FORCE command. Two types of error messages are available :

(1) The "normal" error display.
(2) The "emphasized" error display. In this display, the same error message as in (1) is displayed framed in asterisks. "* * File Not Found * *"

ENTRY : Register A contains a Dosplus error code.

Bit 6 set indicates the normal error message, reset it will produce the emphasized message.

Bit 7 controls the type of exit ERROR will make. Set, ERROR returns to the calling program. Reset, the routine will return to the Dosplus command level, through the ABORT routine, invoking the DEBUG program if DEBUG is on.

ENTER : CALL 4409H

DEBUG : JP 440DH
==========

A JP or CALL to this address will enter the DEBUG monitor, whether or not the DOS "DEBUG" command is active or not.

ENTER : JP 440DH

CAT : 4419H (Model-III ONLY)
==========

This CALL will display a disk directory (consisting of only the filename and extension) on the CRT (or whatever device or file *DO is FORCEd or JOINed to).

ENTRY : C= Drive information
Bits 0-2; Drive number

ENTER : CALL 4419H

NOTE : CAT is not implemented in any version of Dosplus prior to 3.4. 3.3 and 3.38 users cannot call CAT, as an UNKNOWN ERROR CODE will be returned.

FSPEC : 441CH

FSPEC is used to move a filename or device name from an input buffer into a file DCB. FSPEC will scan the filename or device name in the buffer from left to right and copy it character by character into the DCB until it encounters an invalid character. When an invalid character is detected FSPEC will terminate.

For instance :

Input Filespec	DCB contents	
DATE12	DATE12	
MONEY$	MONEY	
ACCREC-P-A	ACCREC	
JULY%SAL	JULY	
12MONTH	----------	Exit with error.
*PR	*PR	
*DO%	*DO	

If an error is encountered, FSPEC does not exit with a specific error code. The user must supply whatever error message is suitable.

Repeated calls to FSPEC will continue to scan the user's input buffer for additional filespecs. A blank space, 20H, or the word TO are acceptable delimiters.

ENTRY : HL=> User input buffer containing file or device spec(s).

DE=> File or Device DCB

ENTER : CALL 441CH

EXIT : BC is destroyed

Z flag status :
- SET - No error.
- RES - Error has occured. No error code returned.

FEXT3 : 444BH
==========

FEXT3 will add a default extension to a filename. For instance, if the file DCB (before OPEN) contains the filename MANUAL.PASSWORD:2, FEXT3 could be used to add the extention TXT. The DCB would then contain MANUAL.PASSWORD/TXT:2. FEXT3 checks for existing extensions, and if one is already present in the DCB, FEXT3 will have no effect.

ENTRY : DE=> File DCB.

HL=> Buffer containing extension.

ENTER : CALL 444BH

EXIT : BC is destroyed

FEXT0 : 4473H
==========

FEXT0 performs the same function as FEXT3. FEXT0 is valid in either Model-I OR Model-III versions of Dosplus. If a program is intended to work on either machine, FEXT0 should be used.

ENTRY : DE=> File DCB.

HL=> Buffer containing extension.

ENTER : CALL 4473H

EXIT : BC is destroyed

MULT : 444EH
==========

This routine will perform a 16-bit by 8-bit multiply.

ENTRY : A= Multiplier

HL= Multiplicand

ENTER : CALL 444EH

EXIT : A= Least signifigant byte of result.

HL= Most signifigant bytes of result.
H= Most sig., L= Next most sig.

DIVD : 4451H
==========

DIVD will divide a 16-bit dividend by an 8-bit divisor.

ENTRY : A= Divisor

HL= Dividend

ENTER : CALL 4451H

EXIT : A= Remainder

HL= Quotient

PARAM : 4454H

PARAM is used to scan a input line for command parameters. For instance, a line such as :

```
RS232 (BAUD=300,WORD=8,RI)
```

contains three parameters; BAUD, WORD, and RI. Each parameter has been assigned a value, in this instance BAUD has a value of 300 (decimal), WORD has a value of 8, and RI has a logical value of TRUE.

PARAM requires that the user set up a "parameter list" in memory containing the name of each parameter (a 6-byte left justified string, padded with blanks, 20H) and a pointer to a 2-byte storage area in RAM for the value of each parameter. The format is as follows :

Address	Parameter	Pointer
START + 0	PARAM1	5000H
START + 8	PARAM2	5002H
START + 16	PARAM3	5004H
START + 24	PARAM4	5006H
START + 32	PARAM5	5008H
START + 40	PARAM6	500AH
Etc., Etc. . . .		

The parameter list is terminated with a 0. In practice, the user will let HL point to the beginning of a command line containing possibly a filespec, and the parameters to be evaluated. DE should point to the beginning of a parameter list such as illustrated above. When PARAM is called, it will scan through the command line, evaluating all parameters, and placing their values in the addresses pointed to in the parameter list. For instance, given the parameter list above, and this command line :

```
LISTER/CMD (PARAM1=66,PARAM2=60,PARAM3=80,PARAM5=Y)
```

After a call to PARAM, the result would be :

Address	Value
5000H	0042H, 66 decimal
5002H	003CH, 60 decimal
5004H	0050H, 80 decimal
5006H	Unchanged from before PARAM
5008H	FFFFH, 255 decimal. Logical True.

Two types of values are recognized by PARAM :

(1) Numeric Values
(2) Logical Values

Numeric values are unsigned integers in the range 0-65535 decimal or 0000H - FFFFH. When PARAM is CALLed, it will place the value into the address pointed to in the parameter list for any given parameter.

Logical values can be represented in many ways, however, their values are always either TRUE or FALSE. A true is stored as a FFFFH. FALSE is stored as a 0000H. Below are listed the acceptable representations for logical values :

Word	Value
Y	TRUE
YES	TRUE
ON	TRUE
N	FALSE
NO	FALSE
OFF	FALSE

In addition, if only a parameter is listed in a command line, such as :

```
DIR :1 (S)
```

the value of S is assumed to be TRUE. Parameters present in the text line BUT NOT IN THE PARAMETER LIST will result in NZ status upon return from PARAM indicating a PARAMTER ERROR.

ENTRY : DE=> Parameter List (see above)

HL=> Command line to be scanned.

ENTER : CALL 4454H

EXIT : BC is destroyed.

Values stored in addresses specified in parameter list. See above.

Zero flag status :
SET : No error
RESET : Parameter error

PARAM0 : 4476H

PARAM0 is the Model-I equivalent of the PARAM call in Model-III Dosplus. PARAM0 is present in both versions of Dosplus, whereas PARAM appears only in Model-III. Programs intended for use on both systems should use PARAM0.

ENTRY : DE=> Parameter list. See PARAM

HL=> Command line. See PARAM.

ENTER : CALL 4476H

EXIT : BC is destroyed.

Values stored in addresses specified in parameter list. See PARAM.

Zero flag status :
SET : No error
RESET : Parameter error.

CKEOF : 4457H
==========

A CALL to CKEOF will return the end-of-file status of a particular file.

ENTRY : DE=> File DCB.

ENTER : CALL 4457H

EXIT : Zero flag status :
SET : No EOF
RES : EOF

The Zero flag will be reset (NZ status) when the NRN (next record number) exceeds the ERN (ending record number). When this is the case, two error codes are possible in the A register :

(1) Error code 28 decimal, 1CH. This indicates that the NRN is pointing to the record immediately following ERN.

(2) Error code 29 decimal, 1DH. This indicates that that the NRN is > ERN + 1.

DSPLY : 4467H

This routine will display a line of text starting at the current cursor position. The routine will return to the calling program when it encounters either a carriage return (0DH) or an end-of-text character (03H). In the case of the carriage return, it will be displayed, moving the cursor to the beginning of the next line on the CRT and scrolling if necessary. If an end-of-text is encountered first, DSPLY will return without displaying it, leaving the cursor at the position following the last character displayed. DSPLY is subject to the re-routing of output via FORCE command.

ENTRY : HL=> Text buffer terminated with 03H or 0DH.

ENTER : CALL 4467H

EXIT : DE is destroyed.

PRINT : 446AH

PRINT functions in a manner similar to that of DSPLY, except that output goes to the line printer instead of the video display. Text is terminated by either a carriage return (0DH) or an end-of-text character (03H). Carriage returns will be outputted to the line printer, end-of-text characters will not. PRINT is subject to re-routing of output via the FORCE command.

ENTRY : HL=> Text buffer terminated with 03H or 0DH.

ENTER : CALL 446AH

EXIT : DE is destroyed.

GTTIME : 446DH
==========

GTTIME will create an ASCII text string representation of the time contained in the system clock in any area of RAM specified by the user. The string is presented in HH:MM:SS format.

ENTRY : HL=> RAM buffer for time string.

ENTER : CALL 446DH

GTDATE : 4470H
==========

A call to GTDATE will create an ASCII string in a user specified area of RAM containing the system date in MM/DD/YY format.

ENTRY : HL=> RAM buffer for date string.

ENTER : CALL 4470H

FDRV : 4479H
==========

FDRV will check for a drive specifier and return the drive number, if found. On entry, HL should point to the filespec containing the drivespec. If a drivespec is not found (meaning no colon is found), FDRV will return a 0, along with error status.

ENTRY : HL=> Drive specifier.

ENTER : CALL 4479H

EXIT : C= Drive information
Bits 0-2; Drive number

Zero flag status :
SET : Valid drive specifier found
RES : Invalid or no drivespec found

CKDR : 447CH

CKDR is used to determine the availability and write protect status of a disk drive.

ENTRY : C= Drive information :
Bits 0-2; Drive number

ENTER : CALL 447CH

EXIT : C= Drive information :
Bits 0-2; Drive number
Bit 6; Reset=5 1/4", Set=8"

Zero flag status :
SET : Drive available
RES : Drive not available

Carry flag status :
SET : Write protected
RES : Not write protected

Interrupt Chain Routines -

The TRS-80 real-time clock generates interrupts to the CPU several times a second (each 33 ms on the Model-III, or 25 ms on Model-I). On each interrupt, Dosplus will service a "chain" of interrupt tasks. The chain is made up of 12 2-byte task slots. Of these, slots 0-7 are serviced each 33 ms (25 ms for Model-I). Slots 8-11 are serviced every 267 ms (200 ms on Model-I). Each slot contains a pointer to another word in memory, which in turn points to the actual interrupt task routine.

Detailed below are the Dosplus routines used to insert and delete tasks from the interrupt chain.

SET : 4413H (Model-III) or 4410H (Model-I)

SET will insert a task into the interrupt chain at any given slot. For example, if we wish to insert a task into slot #4 of the interrupt chain (this is the 33 ms chain), and the interrupt task driver entry point is F652H, we would do the following :

```
        LD      A,4             ;SLOT #
        LD      DE,START        ;POINTER TO ENTRY ADDRESS
        CALL    SET             ;INSERT TASK INTO CHAIN
START   DEFW    0F652H          ;ENTRY ADDRESS
```

This code will insert the task into the chain, and set the entry address equal to F652H

Upon entry to each task in the interrupt chain, the IX register will point to the address containing the pointer to the task entry point. In the example above, when the task is entered IX will contain the address of START.

ENTRY : A= Slot number, 0-11

DE=> Word in RAM pointing to interrupt task entry point.

ENTER : CALL 4413H (4410H Model-I)

EXIT : BC is destroyed

RESET : 4416H (Model-III) or 4413H (Model-I)

RESET performs the inverse of SET, that is, it removes a task from the interrupt chain.

ENTRY : A= Slot number containing task information to be removed.

ENTER : CALL 4416H (4413H Model-I)

EXIT : BC is destroyed

Useful ROM Routines -

GET : 13H
==========

GET will input a byte from a device or file. GET is often used to allow sequential byte-by-byte access to disk files. Error (NZ) status should be ignored when GETting from a device.

ENTRY : DE=> Device or File DCB.

ENTER : CALL 13H

EXIT : A= Input byte

Zero flag status :
SET : No error
RES : Error has occurred. Error code is in register A.

DE is destroyed

PUT : 1BH
==========

PUT is used to output a single byte to a device or file, and as such, is useful in performing sequential I/O. Error status should be ignored when PUTting to a device.

ENTRY : DE=> Device or File DCB

A= Byte to be PUTted

ENTER : CALL 1BH

EXIT : Z flag status :
SET : No error
RES : Error has occurred. Register A contains error code.

DE is destroyed

KEYIN : 40H
==========

This routine will accept a line of input from the keyboard, starting at the current cursor position. This routine is terminated by either a carriage return or a break character.

ENTRY : B= Length of desired input line

HL=> Input buffer in which to place data

ENTER : CALL 40H

EXIT : B= # of characters recieved (less <ENTER> or <BREAK>

C= # of characters originally in field. (Same as B on entry)

Carry flag status :
SET : <BREAK> key was pressed to exit routine
RES : <BREAK> key was not pressed

DE is destroyed

INKEY : 49H
==========

INKEY will fetch a single character from the keyboard. A call to INKEY will not return to the calling program until a key is pressed.

ENTER : CALL 49H

EXIT : A= Key pressed

DE is altered

KBD : 2BH
==========

A CALL to KBD will scan the keyboard for a single character. Unlike INKEY, if a character is not found during the scan, KBD will return to the calling program.

ENTER : CALL 2BH

EXIT : A= Character (0 if no character)

Zero flag status :
SET : No character found
RES : Character found

DE is destroyed

DSP : 33H
==========

DSP will display a single character at the current cursor position and advance the cursor one position if the character is displayable. DSP is subject to re-routing via the FORCE command.

ENTRY : A= Character to be displayed.

ENTER : CALL 33H

EXIT : DE is destroyed

Dosplus Error Codes -

The following is a list of the Dosplus error codes in both decimal and hexadecimal, followed by the associated error message.

Dec	Hex	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 Device Space Available
34	22	Attempted To Use Non Program File As A Program
35	23	Memory Fault During Program Load
36	24	Attempted To Load Read Only Memory
37	25	Illegal Access Attempted To Protected File
38	26	I/O Attempted To Unopen File

39	27	Device In Use
40	28	Protected System Device
41	29	Device Not Available
42	2A	Unknown Error Code
43	2B	Unknown Error Code
44	2C	Unknown Error Code
45	2D	Unknown Error Code
46	2E	Unknown Error Code
47	2F	Unknown Error Code
48	30	Record Not Found During Read
49	31	Track 000 Not Found During Read
50	32	Aborted Command During Read
51	33	Unknown Error Code
52	34	Header ID Not Found During Read
53	35	Header ID CRC Error During Read
54	36	Data Record CRC Error During Read
55	37	Bad Block Detected During Read
56	38	Record Not Found During Write
57	39	Track 000 Not Found During Write
58	3A	Aborted Command During Write
59	3B	Unknown Error Code
60	3C	Header ID Not Found During Write
61	3B	Header ID CRC Error During Write
62	3C	Data Record CRC Error During Write
63	3D	Bad Block Detected During Write