Received 9-30-83

POWERSOFT
PRODUCTS FROM BREEZE/QSD, INC.

11500 STEMMONS FWY.
SUITE 125
DALLAS, TEXAS 75229
(214) 484-2976

September 9, 1983
Re: DP-II Series A.Ø1

Dear DP-II Registered Owner,

Enclosed, please find your FREE DP-II upgrade containing the latest version of DP-II. The system has been renamed, as Micro Systems Software is no longer involved in this project.

All the SYS modules have been reassembled for smoother operation, CONVERT is now fully functional, and other obscure problems have been taken care of, in the process.

Notes on FILES:
---

1) ALL files (except DISKZAP) have internal changes.
2) DO NOT MIX the OLD system or CONFIG files with NEW ones!
3) BASIC/TXT has been modified, as TRSDOS(tm) BASIC/CMD is now licensed and on the disk. TRSDOS(tm) 2.Ø is no longer needed.
4) PROFILE+/TXT,*/PF+ is no longer supported. (as with SCRIPSIT, RSCOBOL, or RSBASIC).
5) ST8ØIII/PAT must be reapplied to the original.
6) MOD16/CMD has been renamed to THINLINE/CMD for Mod 12/16 use.
7) EDAS/CMD is no longer included, but you may COPY over the one you have.

Notes on COMMANDS:

CONFIG - Mod II drives need HL=Y. Mod 12 is like Mod 16.
RENAME - Do NOT use the "TO" seperator. Example to use:
RENAME OLDNAME/BAS NEWNAME/BAS <enter>
SYSTEM(LOGO) - Logo is no longer available.

Special note on making a DOUBLE-SIDED System disk:
---

1) Type THINLINE <enter>
2) FORMAT :d (a true double-sided disk as such)
3) SYSGEN :d
4) COPY !:Ø :d,SPW='PASSWORD'
5) Place the :d disk in drive Ø
6) Type: I :Ø,M <enter>
7) SYSTEM,SAVE='THINLINE'

We are forced to take a stand concerning support for certain Model II software packages. We will not be able to support the following RADIO SHACK programs for the reasons given afterwards:

SCRIPSIT Series
PROFILE+
RSCOBOL or any product written in RSCOBOL
RSBASIC* or any product written in RSBASIC
*/Microsoft's compiler works fine.

While most Model II/12 software will run under DP-II with little, if any modifications, the above named products from RADIO SHACK will only be supported by TANDY on the system that they released them on, which is TRSDOS(tm). They have their good reasons for this. These programs contain "un-documented" calls to the operating system. This is usually (actually always) considered a NO-NO in programming, because everytime the DOS changes, these un-documented locations change, and the result is that the whole program either needs to be re-assembled or patched and re-released for that new operating system. It is a FACT that most Model II/12 software that is written for TRSDOS(tm) and containing only legal and published calls will run under DP-II without changes or slight modification. BASIC programs will run. Most machine language programs will run. Some may need a slight modification, and some of these patches are on your disk. VISICALC runs with the supplied patch. It uses documented calls.

SNAPPWARE will not be made compatible with DP-II either. These products, excellent as they may be, are too tied into the TRSDOS(tm) system to work properly under DP-II.

Please note that we will not be attempting to correct, modify, patch, or in any way make programs "work" with DP-II. This is an impossible task. We have provided patches for various popular programs on the disk. There are several new applications being developed to run under DP-II. If interested, please write.

DP-II has come as far as it can under its present configuration. This version you have recieved (A.Ø1) will be the final update of this system. It's little quirks have been taken care of, and works fast, slick, and reliably. We have created a very practical useable system that is dependable and fast at the same time, but it will not allow for programs that do not "follow the rules". As great as this new system is, it STILL will NOT support undocumented calls. We are sorry if this inconviences you. We would like to have these packages running, but as explained, it was impossible. If and when Tandy changes versions of their DOS, they will also have to deal with these problems. These products are labeled as working under TRSDOS ONLY, therefore we cannot justify refunds for this purpose. There is no reason that you can't insert a dedicated SCRIPSIT or PROFILE disk, when using those programs.

DP-II is a system that has fantastic capabilities for NEW software applications to be written for. Don't forget that under our version of BASIC you get an average of 4K more programming area, plus label-addressing, etc.

To serve you better we ask that your questions or comments be mailed to us. Include screen dumps or whatever you think might help explain the problem. You may call, but our programmers are not always available for questions on the phone, or perhaps cannot stop what they are doing at that moment. Also, you might have to wait a few moments while your registration is checked. On your envelope, mark: ATTN: CUSTOMER SUPPORT (DP-II)

We will answer you as soon as possible with a personnal reply.

Thank you for your cooperation,

PowerSoft Products
Mnet#: 767Ø3,374

To: All registered DOSPLUS II owners
Re: DP-II Series A.01

Dear New DP-II Owner,

Enclosed please find your MASTER DISK containing the latest version of DP-II. All the SYS modules have been reassembled for smoother operation, CONVERT is now fully functional, and other obscure problems have been taken care of in the process.

Notes on FILES:

1) ALL files (except DISKZAP) have internal changes.
2) BASIC/TXT has been modified, as BASIC/CMD is now licensed and on the disk. TRSDOS(tm) 2.0 is no longer needed, however, you must still type: DO BASIC <enter> to apply the patches. (You only need to do this ONCE.)
3) PROFILE+/TXT,*/PF+ is no longer supported. (as with SCRIPSIT, RSCOBOL, or RSBASIC).
4) ST80III/PAT must be reapplied to the original.
5) MOD16/CMD has been renamed to THINLINE/CMD for Mod 12/16 use.

Notes on COMMANDS:

CONFIG - Mod II drives need HL=Y. Mod 12 is like Mod 16.
RENAME - Do NOT use the "TO" seperator. Example to use:
RENAME OLDNAME/BAS NEWNAME/BAS <enter>
(TO is assumed by system)
SYSTEM(LOGO) - Logo is no longer available.
ATTRIB - ACC and UPD parameters cannot be used with wildmasks.

Special note on making a DOUBLE-SIDED System disk:

1) Type THINLINE <enter>
2) FORMAT :d (a true double-sided disk as such)
3) SYSGEN :d
4) COPY !:0 :d,SPW='PASSWORD',I <enter>
5) Place the :d disk in drive 0
6) Type: I :0,M <enter>
7) SYSTEM,SAVE='THINLINE'

We are forced to take a stand concerning support for certain Model II software packages. We will not be able to support the following RADIO SHACK programs for the reasons given afterwards:

SCRIPSIT Series
PROFILE+
MultiPlan
Enhanced Visicalc (regular VC will work)
RSCOBOL or any product written in RSCOBOL
RSBASIC or any product written in RSBASIC (Microsoft's compiler works fine)

While most Model II/12 software will run under DP-II with little, if any modifications, the above named products from RADIO SHACK will only be supported by TANDY on the system that they released them on, which is TRSDOS(tm). They have their good reasons for this. These programs contain "un-documented" calls to the operating system. This is usually (actually always) considered a NO-NO in programming, because everytime the DOS changes, these un-documented locations change, and the result is that the whole program either needs to be re-assembled or patched and re-released for that new operating system. It is a FACT that most Model II/12 software that is written for TRSDOS(tm) and containing only legal and published calls will run under DP-II without changes or slight modification. BASIC programs will run. Most machine language programs will run. Some may need a slight modification, and some of these patches are on your disk. VISICALC runs with the supplied patch. It uses documented calls.

SNAPPWARE will not be made compatible with DP-II either. These products, excellent as they may be, are too tied into the TRSDOS(tm) system to work properly under DP-II.

Please note that we will not be attempting to correct, modify, patch, or in any way make programs "work" with DP-II. This is an impossible task. We have provided patches for various popular programs

on the disk. There are several new applications being developed to run under DP-II. If you are interested, please write.

We have created a very practical usable system that is dependable and fast at the same time, but it will not allow for programs that do not "follow the rules". As great as this new system is, it STILL will NOT support undocumented calls. We are sorry if this inconveniences you. We would like to have these packages running, but as explained, it was impossible. If and when Tandy changes versions of their DOS, they will also have to deal with these problems. These products are labeled as working under TRSDOS ONLY, therefore we cannot justify refunds for this purpose. There is no reason that you can't insert a dedicated SCRIPSIT, MULTIPLAN or PROFILE disk, when using those programs.

DP-II is a system that has fantastic capabilities for NEW software applications to be written for. Don't forget that under our version of BASIC you get an average of 4K more programming area, plus label-addressing, etc. It is excellent for business applications due to its speed, reliability, and extra memory.

One difference from TRSDOS BASIC is that our LOC and LOF functions return a bigger number than theirs. Since different computers and DOS versions vary in this area, we suggest that you change your LOC(x) and LOF(x) statements to add "+LO" to each. At the beginning of your program, set LO=0 for TRSDOS or LO=-1 for DP-II. The other area affected is GET and PUT without a record number specified. Page 504 of the Tandy manual states that the LOC function returns the number used for a GET or PUT without a record number. This is what ours returns, yet theirs always returns the LAST record number accessed. Something is inconsistent and we recommend always specifying a record number in GET and PUT statements. We are not making any changes at this time in order to keep compatible with our previous release.

Roger Fuller has provided the changes necessary to Lewis Rosenfelder's PEEK/POKE changes for TRSDOS BASIC. Note that applying the patches below will disable the OCT$ and NAME (or GOTO LABEL definition keyword) functions in BASIC. Use the PATCH utility to apply this to BASIC.

Optional PATCH to BASIC/CMD (modified version) for PEEK/POKE
First enter the command (from DP-II Ready): PATCH BASIC/CMD.BASIC.
Then, when the * prompt appears, type:

R=1 B=23H D=D0
R=1 B=28H D=4F 4B
R=2 B=35H D=D0 45 45 4B
R=51 B=CFH D=C3 FF 67
R=65 B=1BH D=CD DD 3C D5 E7 2C CD EA 3C
R=65 B=28H D=D1 12 C9 CD 5D 44 7E C3 FB 3A
R=69 D=74H D=05

Now press BREAK to exit the PATCH utility.

We have several packages available to enhance the use of DP-II.

BACKREST - a hard drive BACKUP/Restore utility that is FAST! Handles files larger than a floppy with ease. $75.00

MICROTERM - A "smart" terminal package that supports the latest state-of-the-art in modems, as well as being able to communicate with just about any host. $75.00

PowerMOD/DP - A machine language disk utility that allows you allows you zap or alter files, memory, the disk, and has MANY uses. $50.00

PowerMAIL+ - A FAST machine language mailing/data system. Will handle over 90,000 names with proper storage! Versatile report generator. Write for complete details! $150.00

To serve you better we ask that any questions or comments be mailed to us. Include screen dumps or whatever you think might help explain the problem. You may call, but our programmers are not always available for questions on the phone, or perhaps cannot stop what they are doing at that moment. Also, you might have to wait a few moments while your registration is checked. On your envelope, mark: ATTN: CUSTOMER SUPPORT (DP-II). We will get back to you with a personal reply.

Thank you for your cooperation,

PowerSoft Products
CIS User ID: 76703,374

POWERSOFT

17060 Dallas Parkway, Suite 114 • Dallas, TX 75248

January 19, 1988

Dear DOSPLUS II Registered Owner,

We are writing you regarding the fact that the date on the DOSPLUS II Operating System ran out on 12-31-87. If you still use the system, then you obviously know this.

In 1984, the company that originally marketed DOSPLUS II, MicroPower, Inc., out of Florida, went out of business. PowerSoft inherited the small amount of inventory and the mailing list, and when it was all gone, the product was dropped.

We started getting a few phone calls in December concerning a date fix. There were several obstacles here. 1) the original programmers were long gone, 2) we did not have a Model II/12/16 to test this with and 3) only about eight people have contacted us about this.

But it was important because the people who did call us explained that they were running their business, etc. on DOSPLUS II, and things were still running fine for them. So somehow, we had to find a patch.

After some tracking down, we found one of the original authors of the system, Kim Watt, in Milwaukee. He still had the source code in his closet, but did not have a Model II, nor access to one.

Kim is a mainframe programmer now, and really was not familiar with the code any longer. Also he is busy, as we all are, and asked to be paid for his time, as we all would expect to be if it was us.

So, after checking around the surrounding areas, he finally found a Model II in pretty decent shape that he could have access to. Then he had to refamiliarize himself with the tools, the source code and the system itself. Finally he took apart the date routine and wrote a patch that could be installed by the user without sending in his disk. Our cost in getting Kim to write this patch for us was almost $1,500.

So, if you still use the DOSPLUS II Operating System, and would like to extend the Date function, please order the DP-II Date Patch from us. Your package will also include instructions on maintaining the date forever, so this is the last update you'll have to buy.

The cost of the Date Patch is $50. This includes step-by-step directions for installation. Every keystroke you have to type will be listed line by line. When you're done, re-boot, make a backup and you're through. We are sorry we have to charge for this patch, but since MicroPower is out of business, we figured the best we could do was find one of the original programmers and pay him to write it.

If we can help you with this patch, please contact us. We accept Visa, MasterCard or check. We do not have a Model II here, so please do not ask us to send you a new Master Disk with the patch already applied. It is not possible.

We do have a few DP-II oriented programs still available at bargain prices.These programs work great and do not contain bugs. There is not any support for them, however. If you still use the system, take advantage of these specials while we still have a few left. They're all great programs and supplies are definitely limited. Please include $3 for shipping/handling. Write for Data sheets on each, if more information is required.

MicroTerm - Smart Terminal package for modem and DP-II	$25 with full documentation.
PMOD - Powerful and easy Disk, File or Memory editor for DP-II	$25 with full documentation.
PowerMAIL Plus - A GREAT 4-Star mailing list system for DP-II	$50 with full documentation.

Thank you,

PowerSoft Products
17060 Dallas Parkway, Suite 114
Dallas, TX 75248

Now "Smart" Telecommunications for Mod II/12!

MicroTerm

We present **MicroTerm** - the high speed SMART Terminal package - for today's AND tomorrow's telecommunications needs. **MicroTerm** is the first truely high speed "smart" terminal package for the Mod II/12 system. It runs at speeds up to 2400 baud without the need for insertion of null characters. With the advent of the new smart modems (like the Hayes 1200™ and Radio Shack™ Modem II), plus all the new information systems available, this becomes more and more important to your needs.

MicroTerm is MORE than just a terminal program, however. It's a complete communications package. It supports file transfer in both the traditional ASCII mode and the new "error-free" direct-file- mode.

Not only does **MicroTerm** surpass all features ever made available for telecommunications on this machine, but it also offers MANY features that offer ease of operation second to none. This program was written with the USER in mind. You don't have to be a programmer to understand **MicroTerm**!

MicroTerm contains powerful translation tables that let you use your computer with any variety of communications service or main-frame. Its unique MACRO-KEY function allows you to have TEN "user-defined" keys that transmit up to 64 characters at a touch of a key. **MicroTerm** also allows you to dial the phone and transmit the buffer at a specified time completely un-attended by the operator. It stores your phone numbers in a data file as well, so going from **Dow Jones**™ to **CompuServe**™ is a snap and fast! **MicroTerm** is supplied with a User's Manual that clearly explains program operation and answers many questions even before they need to be asked.

MicroTerm will open up the world of telecommunications for as never before! By using this program, you can do MORE in less time! The serious computer user cannot afford to be without **MicroTerm** in their library.

Check these features:

1. Spooled parallel printer output and screen print functions.
2. Easy to use translation tables with ready made default files for most applications.
3. Maximum auto dial support with user definable tables to allow dialing from a list of numbers at the touch of a key.
4. MacroKey function allows you to have up to TEN user-defined keys with a maximum of 64 characters per key. Can be used for auto-logons, etc.
5. Return to **MENU** at any time and execute a command while still receiving data! Screen will be updated upon return. Never any lost data with **MicroTerm**!
6. Supports file transfer with **ASCII** or new **Direct-File-Mode**.
7. DOS commands from MENU without exiting.
8. Large CAPTURE Buffer for uploading and downloading of data.
9. Supports most major brands of modems with full auto-dial capabilities where used.
10. Supports many of the newest features that that the newest state-of-the-art modems allow. The HAYES 300 or 1200 just shine with **MicroTerm**! Works equally well with **Radio Shack**'s newest smart modems.

MicroTerm comes supplied on disk with the **DP-II "Kernel" Operating System.** The DP-II Kernel is a smaller, stripped down version of **DOSPLUS II** that is meant for distribution of a single application. MicroTerm has a lot of features that require this DOS, so when you want to communicate, you simply boot your MicroTerm disk, rather than your **TRSDOS**™ disk. The disk is NOT protected, so you may make copies for your own use, PLUS data is "convertible" for transferring files between DP-II and **TRSDOS**™.

Includes simple and easy to use program with clear User's Guide. Only $25.00

PowerSOFT Products - 17060 Dallas Parkway, Suite 114 - Dallas, TX 75248
(214) 733-4475

DOSPLUS II

DISK OPERATING SYSTEM
FOR THE TRS80 MODEL II AND Z80 MODEL 16

Acknowledgements:

DOSPLUS II is the product of three months of VERY hard work on the part of a talented team of programmers and technical writers. You are seeing the results of this. DOSPLUS II is the most powerful Disk Operating System ever written for a microcomputer. We who have had the privilege to work on this project would like to gratefully acknowledge the tireless dedication of a few and the valuable assistance of several in the creation of this system.

The few -

System authors :	Kim Watt and Steve Pagliarulo
Documentation authors :	Mark Lautenschlager and Renato Reyes
Project leaders :	Dennis Brent and Larry Studdard

The several -

Pete Carr, Todd Tolhurst, Tom Price, Bill Stockwell, John Long, Lance Micklus, D.L. Herman, and Vernon Hester

These people have either contributed time as Beta test sites or suggestions and input toward the system design. We would like them to know they are appreciated.

We hope that you will like your new system. It is a great deal different than anything else you have ever used before, so we strongly recommend that you read carefully the "DOS operations" section of the manual before you actually begin using the system.

If you have problems with your DOSPLUS II, there are two companies providing technical support for the system :

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

PowerSOFT Inc.
11500 Stemmons Freeway
Suite 125
Dallas, TX 74229
(214) 484-5783
MicroNet 70130,203

Both companies will answer questions and offer support on the Xtra-80 SIG on CompuServe. It is listed as the "QSD SIG" in the Special Interest Groups menu.

As much as possible, problems of a non-critical nature should be referred via mail to leave the telephone support lines free for serious problems of highly urgent natures.

We have enjoyed bringing you this system. We will do everything in our power to see that you never regret the descision to switch to DOSPLUS II. If there is ever ANYTHING that we can do to help you implement your system, please let us know. We do reserve the right to say "NO!", but it never hurts to ask. Thank you.

The MicroPower "Team"
Boca Raton, FL and Dallas, TX
September 1982

DOSPLUS II User's manual table of contents

The DOSPLUS II techincal manual has its own separate table of contents. Please reference that for locating technical information.

The technical manual begins immediately after the DOSPLUS BASIC section. You may locate it via the insert tabs.

Standard files included with DOSPLUS II

The following files are included with your DOSPLUS II and can be found on the master diskette :

Demo/txt	This is a DO file that gives a brief demonstration of the DOSPLUS II system. It is set on an AUTO when you receive the disk and will execute the first time that you boot up. You may remove this AUTO via the AUTO command or the program will do it for you after it has run.
Init2/txt	This is a DO file that will intialize a hard disk as two volumes. Explained in SYSGEN.
Init4/txt	Same as above except sets it to four volumes.
Config2/txt	This is a DO file that will configure the rest of the disk drives after Init2 is done.
Config4/txt	This is a DO file that will configure the remaining volumes after Init4 is done.
Basic/txt	This is a DO file that copies BASIC from TRSDOS to DOSPLUS and patches it for use under DOSPLUS with BASICP/CMD. Consult the BASIC section for details.
Nb/flt	Filter file that turns off the blinking cursor. Filter the display using this file.
Control/6	Filter file that moves the BREAK key to Ctl-6 and causes the BREAK key to send an FFH code instead. Filter the keyboard using this file.
Profile+/txt	This DO file copies all the Profile II+ modules from TRSDOS to DOSPLUS and calls the patches one at a time.
*/pf+	Any file with a "/pf+" extension is a patch file for use with the DO file Profile+/txt.
Vc/pat	Patch file for VisiCalc. Once Visicalc is patched with this file, it will no longer be necessary to use Svcint first.
St80iii/pat	Patch for ST80III program.
Atb/pat	Patch for ST80's ASCII to Binary program.
Bta/pat	Patch for ST80's Binary to ASCII program.
Cksum/pat	Patch for ST80's Checksum program.
Ftype/pat	Patch for ST80's FTYPE module.
Bigben/pat	Patch for the file Bigben/cmd that is included with TRSDOS.
Dvorak/flt	Filter to allow operation under a DVORAK keyboard. Filter the keyboard using this file.
Mod16/cmd	Configuration file provided for you. Execute this file to configure your machine as a two drive Model 16.

Please note that new patch files and filters will be provided as the need arises. Such files could include patches to major programs or drivers for non-standard hardware. All of these will be provided at nominal charge to registered owners. You will be individually notified by form letter when such things become available. This will only apply if you have filled out and returned the registration card included with this manual.

We therefore suggest strongly that you do so at once.

Introduction

Welcome to DOSPLUS II! DOSPLUS II is an exciting new concept in Model II Disk Operating Systems that started with and is based on the premise that a system does not have to be limited in features and difficult to use in order to be fast. With DOSPLUS II, we hope to be the first to shatter the "If you can understand it, it can't be good" myth. DOSPLUS II is a system that provides an unprecedented level of flexibility for the programmer along with an equally unprecedented level of user-friendly operation for the novice.

This manual is divided into five parts :

The operations manual. This portion of the manual covers what you need to know about the system before you can effectively operate it. It covers first time use and explains general syntax and operating concepts.

The library of commands. This portion of the manual covers the library of commands. A library command is a function that is intrinsic to the system, that is, contained within the actual system itself.

The utilities manual. This portion of the manual covers the DOSPLUS II utilities. A utility is a program that is part of the system, but is not contained within the system. These programs (at least some of them) may be removed by the user in the interests of disk space and efficiency.

The BASIC enhancements. This portion of the manual describes the enhancements that the BASIC modification package (included standard with DOSPLUS II) provides for your MicroSoft (Radio Shack) BASIC.

The technical manual. DOSPLUS II is almost 100% TRSDOS compatible in all of the documented supervisory calls, but has added many more calls you will wish to take advantage of in your programming. This section covers all the supervisory calls and internal documented RAM addresses. It also includes a section on diskette format and directory organization.

We hope that you will be very pleased with your DOSPLUS II, and believe that you shall find it very easy to learn to use. There ARE some differences between DOSPLUS II and other systems, though, so we strongly suggest that you read through the manual before attempting to implement the system.

First time operation

If this is the very first time you are using DOSPLUS II, you should first make a BACKUP of your Master diskette and then file the Master away in a safe location.

Booting up -

First, make certain that all cables are connected correctly and that the machine has power. After switching on the machine, place the DOSPLUS II Master disk in drive 0. Close the drive door. If the system fails to boot at once or reports an error, press the reset button on the face of the computer directly underneath the power switch and try again.

The DOSPLUS II copyright and logo will be displayed and you will be asked for the date. You may enter the date or press ENTER or BREAK to skip. Then you will be asked for the time. Again, you may enter the time or press ENTER or BREAK to skip. The logo and those two questions may be disabled via the SYSTEM command if you wish (see the library command SYSTEM).

Immediately after booting, you will be taken to the "DOS command mode". The prompt "DosPLUS II" and a cursor will be displayed, This mode is that which allows you to enter commands to the system. If you are a Model 16 user, you should at ONCE type "MOD16" and press ENTER. This will load a Model 16 configuration file provided for you that sets the needed parameters for Model 16 operation. If the drive has timed out before you have a chance to type this in, you will receive a system error message. Press ENTER to return to the DOS command mode and try again. Once the configuration is loaded, the drives will never be a problem.

If you are a Model 16 user, you should set some variety of a configuration file on automatic execution (see the library command AUTO) so that you will not have to bothered. The only required configuration difference is that the "motor delay" must be set for the Model 16 drives (see the library command CONFIG).

Hard disk owners should refer to the section on SYSGEN to obtain instructions on how to install DOSPLUS II on their hard disks. However, even hard disk owners should first backup their Master disks as per the instructions below.

Backing up with multiple drives -

At this point, you should be at the DOS command mode with your DOSPLUS II Master disk in Drive 0. Place whatever disk you wish to use for your backup in Drive 1. It does not matter whether the disk is blank or not.

From the DOS command mode, type "BACKUP" and press ENTER. The backup program will load, display its header, and prompt :

Source drivespec ?

Reply with a "0" (a numeric "0", not an alphabetic "O"). This question is asking you which drive the disk we are backing up FROM is located in. Since we are backing up from the Master disk in Drive 0, we reply with that. You will then be asked :

Destination drivespec ?

Reply with a "1". This question is asking you which drive the disk we are backing up TO is located in. Since we placed the disk to contain the backup in Drive 1, we reply with that.

DOSPLUS II will then read first from the source disk and then from the destination. If the destination disk is blank or in an incompatible format, BACKUP will format it and proceed with the backup. If the disk was NOT blank, you will be told :

Diskette contains data, use or not ?

At this point you have three options :

(1) Abort the backup.

(2) Continue using present destination format.

(3) Continue after re-formatting destination disk.

To abort the backup, type "N" and press ENTER. DOSPLUS II will then ask you to insert a system disk and press ENTER. If the DOSPLUS II Master disk is still in Drive 0, then a system disk is already in place, so just press ENTER. You may use BACKUP to backup between two non-system disks if you wish.

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

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

When BACKUP is backing up the disk, it will read just as many granules as it can fit into available memory before writing them to the destination disk. It will write every granule that it read and then come back and verify them. BACKUP only copies granules that currently contain data, so do not be alarmed if you notice that certain cylinders numbers got "missed" or some change rapidly. They simply were empty or partially full and so there was no need to copy them.

BACKUP attempts to make a mirror image backup. That is, the destination disk will be exactly like the source. If it encounters too many flaws on the destination disk and cannot place data in the same location as it exists on the source, it will abort with an error.

BACKUP also will not continue after any sort of disk read error. In the event one should occur and you cannot backup the disk, you may use COPY to remove what files you can and preserve all possible data (see the library command COPY).

When BACKUP is finished, it will flash the message :

Insert SYSTEM disk (ENTER)

Verify that the DOSPLUS II Master disk in still in Drive 0 and then press ENTER. Your backup is complete. File your Master away in a safe location.

Backing up with a single drive -

At this point, you should be at the DOS command mode with your DOSPLUS II Master disk in Drive 0. Obtain whatever disk you wish to use for your backup. It does not matter whether the disk is blank or not.

From the DOS command mode, type "BACKUP" and press ENTER. The backup program will load, display its header, and prompt :

Source drivespec ?

Reply with a "0" (a numeric "0", not an alphabetic "O"). This question is asking you which drive the disk we are backing up FROM is located in. Since we are backing up from the Master disk in Drive 0, we reply with that. You will then be asked :

Destination drivespec ?

Again, reply with a "0". This question is asking you which drive the disk we are backing up TO is located in. Since we are backing up using a single drive, we will backing TO Drive 0 as well as FROM it. Therefore, we reply with that.

DOSPLUS II will then read first from the source disk and then from the destination. It will prompt you as to when to insert each of the disks. After inserting the disk prompted for, you will press ENTER. It is MOST important that you do not confuse the order of the two disks and insert source instead of destination or vice versa. Also bear in mind that from time to time, DOSPLUS II will need a system disk. It will prompt for that. At that point, you should give it the Master disk and press ENTER. Pay attention to the prompts and be careful.

If the destination disk is blank, BACKUP will format it and proceed with the backup. If the disk was NOT blank, you will be told :

Diskette contains data, use or not ?

At this point you have three options :

(1) Abort the backup.

(2) Continue using present destination format.

(3) Continue after re-formatting destination disk.

To abort the backup, type "N" and press ENTER. DOSPLUS II will then ask you to insert a system disk and press ENTER. If the DOSPLUS II Master disk is still in Drive 0, then a system disk is already in place, so just press ENTER. If not, then insert the Master disk before pressing ENTER. You may use BACKUP to backup between two non-system disks if you wish.

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

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

As BACKUP makes the backup for you, it will prompt you for the source disk and then the destination disk and every so often for the system disk. Watch these prompts and be certain to give the machine the correct disk at the correct time. Do NOT press ENTER repeatedly. DOSPLUS II has type-ahead and those ENTER keys will be held until the next prompt at which time DOSPLUS II will proceed before you can switch the disks.

Also, when BACKUP is backing up the disk, it will read just as many cylinders as it can fit into available memory before writing them to the destination disk. It will write every cylinder that it read and then come back and verify them. BACKUP only copies cylinders that currently contain data, so do not be alarmed if you notice that certain cylinders numbers got "missed". They simply were empty and so there was no need to copy them.

BACKUP attempts to make a mirror image backup. That is, the destination disk will be exactly like the source. If it encounters too many flaws on the destination disk and cannot place data in the same location as it exists on the source, it will abort with an error.

BACKUP also will not continue after any sort of disk read error. In the event one should occur and you cannot backup the disk, you may use COPY with the "prompt" parameter to remove what files you can and preserve all possible data (see the library command COPY).

When BACKUP is finished, it will flash the message :

Insert SYSTEM disk (ENTER)

Insert the DOSPLUS II Master disk (or your backup disk, it doesn't matter at this point because both are system disks) in Drive 0 and press ENTER. Your backup is complete. File your Master away in a safe location.

General Syntax

In this portion of the manual we will cover the information you will need to be aware of in order to effectively execute commands under DOSPLUS II.

General notation:

For the sake of simplicity, we will use certain standard formats when displaying command lines and so forth. When displaying a command line example, for instance, a word that appears all in capitals indicates that it must be typed in exactly as it appears. The notations are :

Capital letters	Word must be typed in as shown.
Lowercase letters	Information to be entered by user based on a list of valid values and parameters for that command.
Brackets [ ]	Indicate an optional delimiter. These are used to specify data flow (i.e from which file or device to which other file or device). These may be omitted unless you wish to alter the direction of data flow.
Braces { }	Indicate the parameter field. This field is used to specify additional data that modifies the action of the command.
H, B, O, Q, D	These are base specifiers. They indicate which base the number they follow is in. "H" indicates a hexidecimal number, "B" indicates a binary number, "O" and "Q" both indicate octal base numbers, and "D" indicates a decimal base number. If none of these is present, decimal base is assumed.

Some terms with which you will need to become familiar :

Separator	This is a character used within a command field to indicate the various portions of the field. They are used to "separate" the areas of the command. These are not optional and must be present for the command to interpret correctly.

Terms (cont.) :

Delimiter	A word used to signal which area of the command field follows. These need only be included if you are going to enter the areas of the command line in a non-standard order, otherwise they are assumed.

Memory usage:

DOSPLUS II uses memory from 0000H to 27FFH. This area of RAM is referred to as "system memory". User programs should never reside in this area. The memory location 2800H is the default "Lomem". Lomem is the point at which DOSPLUS II will start user memory. Most system (library) commands and certain specified utilities will use memory below 2800H. If you wished to make use of this, you could locate your program at 2800H and then adjust Lomem with SYSTEM (see the library command SYSTEM) to cover it. DOSPLUS II would then protect that area of memory automatically, preventing your program from being overwritten unless the loading program specifically requests that area of memory.

The default value for the top of useable memory (called "Himem") is FFFFH in a 64K machine, 7FFFH in a 32K unit. This value will change as DOSPLUS II allocates high memory for various drivers and filters that the user may call into play. In its standard form, DOSPLUS II will refrain from using high memory whenever possible, but the more of its special features you call upon, the more space in RAM it needs and it will allocate high RAM for those.

The area between Lomem and Himem is called "User memory". This is the area that the user programs and data will reside in. Whenever we refer to user memory, that is the area we will be talking about. User memory can change, as DOSPLUS II allocates high RAM or the user adjust Lomem, so you should check these from your programs. The addresses in memory where the Lomem and Himem values are stored are documented in the technical section. If you have your programs adjust these values as they load in, then you are assured that DOSPLUS II will not overlay them with drivers of its own at some later time.

All of this, of course, concerns only the machine language programmer. BASIC programmers and the average user will never even know that there IS a Lomem and a Himem, much less that they are changing.

Certain commands and utilities (specifically the library command COPY in its single drive mode, and the utility programs FORMAT, and BACKUP) will use all available memory when called upon. If the amount of memory free for their use is not enough to perform whatever operation you have called upon them for, they will abort with an "Insufficient memory" error. Free system memory is honored and all data outside this area is preserved.

Booting up the system -

To begin using DOSPLUS II, you must first "boot up the system". That term means, simply, to load DOSPLUS II into the computer and begin. To accomplish this, insert the DOSPLUS II diskette into Drive 0 (the internal drive), and press the reset button located on the face of the Model II.

After a few moments, the screen will clear and the DOSPLUS II logo and copyright notice will be displayed. The system will prompt you for the time and date.

Answer those questions with the allowable formats for each. DOSPLUS II is very flexible when it comes to the form of input it will accept. For a detailed list of formats that will work, consult the TIME and DATE commands in the library commands section of this manual. To simplify matters now, enter the date in the standard "MM/DD/YYYY" format and the time in the standard "HH:MM:SS" format.

If you wish, these initialization questions may be disabled via the SYSTEM library command (see the library command SYSTEM). Even the logo may be turned off, if that is your desire.

When booting up the system, after all initialization data is input (if any of them are turned on), DOSPLUS II will automatically execute any commands that have been placed on the AUTO function (see the library command AUTO). You may abort this automatic execution by holding down the ENTER key. There is, however, a method to set a "non-interruptable" AUTO function. If that is the case, then you have no choice but to let the automatic command sequence run its course.

Assuming that there was no AUTO function engaged, after all initialization parameters are complete, you will see the prompt:

DosPLUS II

and followed by a cursor. At this point, you may enter a command for the system to execute. This is called the DOS command mode. That is a very important term to remember, as we will be referring to it constantly throughout this manual.
Entering a command :

Whenever you are at the DOS command mode, you may type a command (up to 80 characters in length), press ENTER, and have DOSPLUS II evaluate and execute it. This command may consist of either one of a couple of things:

> A library command. A command that is part of the actual DOSPLUS II system. The name of this command will not actually correspond to any disk file name in the directory.
>
> A program name. This would be the name of a machine language program stored in executable format on the disk. DOSPLUS II utilities and user programs may all fall into this category.

For example, if you were to type:

"LIB" and then press ENTER

the system library commands would be displayed on the screen. The procedure flows as follows:

(1) DOSPLUS II examines the command you have just entered to see if there is any error contained in the line. If you type "!()=?", for example, DOSPLUS II will respond immediately with the error "Improper file name". If there are no apparent errors in your command then DOSPLUS II will proceed to the next step.

(2) DOSPLUS II checks to see if it is the name of a valid library command. If it is, it is executed at once.

(3) If it is NOT, then DOSPLUS II checks all available drive devices to see if what you have typed is the name of a file located on one of them. DOSPLUS II checks the devices in the order Device 8 - Device 15. If the system is in its stock form, that will be Drives 0-7, but you may reconfigure that if you desire. This is explained further in the next section of the manual - "File and device specifications".

(4) If DOSPLUS II doesn't find a matching filename, it will give you the appropriate error message.

Types of commands [command syntax]:

When we speak of "command syntax", we are referring to the structure of the command. How is it put together? What sort of format do we follow when entering a command? What are the various parts of the command line?

The command syntax tells you how to put commands with parameters and make sense to the system in doing it. In the manual we will show command syntax briefly in a special area directly underneath the brief command description immediately following the command name. This area will be set off from the regular text by two bars of the "=" character.

You have three basic types of commands: (1) Direct (no I/O channel affected) commands, (2) Simple commands (only one I/O channel), and (3) Complex commands (more than one I/O channel used).

There are very few direct commands in DOSPLUS II. Nearly every command can at least have its I/O device specified. For example, if you type "LIB" and press ENTER, you get a listing of the system library commands on the screen. But if you type "LIB TO @PR" and press ENTER, you will get the same list on the line printer. That is an example of what we mean by a simple command.

Your specific command forms are as follows:

Direct commands

COMMAND {options} comment

"COMMAND" is the DOSPLUS II library command.

"{options}" is the list of parameters that may be used to in some way modify the action of the command. They may be switches, in which case the mere presence of the term in the option field will affect the command, or they may be parameters, in which case you must specify a value or text string along with the term.

"comment" is an optional field that may be used to display a comment as to the intended action of the command line. If a comment field is to be used, then the option field MUST terminate in a right brace ("}"). If no comment field is used, the brace is optional, the command will terminate at the end of line character.

Simple commands

COMMAND I/O channel {options} comment

"COMMAND" - see above.

"I/O channel" is the file or device specification that tells DOSPLUS II where the effect of the command will be. For a more detailed explanation of the term I/O channel, consult the section "File and device specifications".

"{options}" - see above.

"comment" - see above.

Complex commands

COMMAND [FROM] I/O channel [TO] I/O channel {options} comment

"COMMAND" - see above.

"[FROM]" indicates that the source I/O channel follows. This is the channel that the data will be coming from, or the channel that is initiating the action.

"I/O channel" - see above.

"[TO]" indicates that the destination I/O channel follows. This is the channel that the data will be flowing to, or the channel that will be acted upon.

"{options}" - see above.

"comment" - see above.

A more detailed explanation of the command line will be afforded in the next section, "File and device specifications", after we have clearly defined these terms and outlined exactly what an I/O channel is. The above information was designed as a quick overview for the more experienced user, or a refresher for the novice, only.

Differences from TRSDOS -

There are two differences in the command structure between DOSPLUS II and TRSDOS :

(1) DOSPLUS II is device independent.

(2) DOSPLUS II ignores case in commands.

TRSDOS required that all commands be issued is upper case letters only. TRSDOS is case dependent throughout its entire system that way. DOSPLUS II, on the other hand, will allow you to enter commands in either upper or lower case and is case independent throughout its entire system that way (except were noted in special cases).

We feel that this is a great deal more flexible and friendly to the user.

Under TRSDOS, your library commands dealt only with files. Under DOSPLUS, you may use files and devices interchangably. DOSPLUS is completely device independant. Because of this we have I/O channels where TRSDOS has filespecs. Again, this is far more flexible and user friendly.

For a more detailed description of devices and device independence, see the portion called "Devices and device specifications" in the next section of this manual - "File and device specifications".

System or data disk -

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

The system disk is so called because it contains the Disk Operating System or DOS. The DOSPLUS II disk 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 similar to a child without any formal education. Capable of some very basic functions (due to the ROM) and certainly capable of learning; but unable to perform any advance functions without aid.

The first drive device must ALWAYS contain an operating system! When you execute a CONFIG display, this drive will be the first one listed. You may have renamed it or re-routed which drive it is referring to, but it must still be a system disk (even if it is the hard disk). This is covered in greater detail under the library command CONFIG.

A data disk is one that holds the programs and files that you have created but does not have an operating system resident upon it. Single drive users will only be able to make limited use of data disks. You create data disks by using FORMAT (see the introduction section GETTING STARTED and the utility program FORMAT).

Disk Master password -

At the time of format, each disk is assigned a Disk Master password. You have the option of setting this password at that time. You may not set the password if you wish (in effect making the password a string of blanks). DOSPLUS II is one of the few systems that will allow you to enter a null password (i.e. pressing ENTER for the password).

The Disk Master password is used for a variety of things :

(1) Altering disk information with PROT.

(2) Password override on KILL and COPY.

(3) In place of file password.

Because DOSPLUS II will accept the Disk Master password in place of a file password any time that it goes to open a data file or load a program, you should set all passwords to something. To have a disk with no password set is to effectively unprotect all files on the disk.

The Disk Master password will be required during most, if not all, global operations affecting files and the disks themselves. It is therefore important that if you set a password, you do NOT forget it.

The password on the DOSPLUS II disk is "PASSWORD".

User file or System file -

DOSPLUS II has two types of files: System files and User files. For a definition of a file, see the "File and device specifications" section of the manual.

System files are those that are essential to the proper operation of DOSPLUS II and its library commands. They are denoted by the "/SYS" extension and by the "S" in the "Attrib" column of the directory display (see File and device specifications and the library command DIR).

A "minimum system disk" for executing programs in single drive systems would therefore only have to contain these files in order to operate. All other files may be removed as space demands.

This brings us to user files. A user file is anything that is not a system file. This includes your DOSPLUS II utilities (such as FORMAT and BACKUP). They can be BASIC programs, data files, or anything else that is not actually a part of the DOSPLUS II system.

None of these programs are vital to system operation and may be removed if needed. Be careful, though. Many functions that are just taken for granted to be part of the system (like FORMAT and BACKUP) are actually utility programs (user files) and will no longer function if removed from the system.

Should you need to copy or kill a protected utility file, the password on those files is "CMD". The user may not access the system files in the standard manner at all. However, an exception has been made for the PATCH utility (see the utility program PATCH). When patching a system file, you may use the password "SYS".

File and device specifications

In this section we will attempt to cover four areas of great importance to the DOSPLUS II user :

(1) File specifications.

(2) Devices and device specifications.

(3) Detailed breakdown of the command line.

(4) Detailed explanation of the terms used.

This section is one of those "must read" portions of the manual.

File specifications -

The only way to store information permanently and retrieve it later is to place 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 nothing more than a large electronic file cabinet.

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

filename/ext.password:ds(diskname)

"filename" is a sequence of 1 to 8 characters used to specify the file. These may be any character with the exception of those listed below as "reserved characters". The DOS will automatically convert all lower case letters to upper.

"/ext" is the optional filename extension consisting of up to three characters. Like a filename, this may be made up of any characters with the exception of those that are designated below as "reserved".

".password" is the optional file password consisting of up to eight characters. This password will be used in conjunction with your file's protection level set via ATTRIB to prevent unauthorized access.

":ds" is the optional drive specification noting which drive this particular file is stored on. This may be any two characters with the exception, of course, of reserved characters. If given, it must correspond to a drive specifier currently defined within the system.

"(diskname)" is not implemented under the DOSPLUS II system. However, TRSDOS used this in their system and many application programs have it in them. To prevent any incompatibility, DOSPLUS II will not return an error if the disk name is specified. Remember, though, it is NOT used.

Note that there can be NO blank spaces within the filespec. DOSPLUS will terminate the filespec at the first blank space encountered. For example, "BAD NAME/DAT" will be seen by the DOS as "BAD".

Also, all areas of the filespec that begin with a specific character (i.e. extension with "/", password with ".", and drivespec with ":") must begin with that character. If the character is omitted, an error will result.

There are certain characters that are reserved by the system and may not be used in a filespec. Most of these are used elsewhere in the current system to indicate special cases, but some of them are being reserved for future revisions and versions of DOSPLUS II. The list is :

!	Exclamation point
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Period
/	Slash mark
;	Semi colon
:	Colon
@	At sign
=	Equals sign
?	Question mark
"	Double quotes
'	Single quote
{	Left brace
}	Right brace
*	Asterisk
	Space

You also may not have an ASCII 03 (end-of-text) or an ASCII 13 (carriage return) in your filename as either one of these is used to signal the end of the command line to the DOS. Therefore they are also reserved.

The filename, extension, and drive specification all serve to contribute to a file's uniqueness. The password does not. It merely controls access to the file (see the library command ATTRIB).

Further detail and examples regarding filespecs -

Throughout this manual 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 five parts. For instance, given the filespec :

PRICE/DAT.DOLLAR:1(DOSPLUS)

This filespec has all five parts. The first part is the name "PRICE". This name can be up to eight characters long, any may contain any character with the exception of the list of reserved characters already given. Some example filenames :

Legal	Illegal	Reason
MONEY	?MONEY	"?" is a reserved character
JUNSALES	JUNESALES	Too many characters

The second part of a filespec is the extension. In our example, PRICE/DAT.DOLLAR:1(DOSPLUS), the extension is "/DAT". The extension is separated from the name by a slash mark (i.e. "/"). An extension may use any non-reserved character and may be up to three characters in length. The extension is useful in indicating what sort of file the filespec is describing. Here are some examples :

Extension	Use
BAS	BASIC language program
TXT	Text file
DAT	Data file
CMD	Executable Z-80 object code. Usually called a "command" file
FLT	A filter file used to manipulate character I/O to the various devices
PAT	A patch file used by the patch utility to make corrections to the system and modify outside software
ASM	Assembly language source file
CIM	A "core image" file. A file that consists of data transferred directly from memory to disk. Not necessarily executable code.
DVR	A driver file used by the SET command to implement a device.
SYS	A system file. Part of the actual DOSPLUS II operating system.

While the extension is not a required part of the filespec, it is used often 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

The one exception to all this is the Z-80 object file (also known as the "command" file). In order for the DOS to be able to load and execute these programs directly from the DOS command mode, they must have some form of extension. We recommend highly that the extension "/CMD" be used, since the DOS will assume that if given no other and that frees you from having to constantly specify the extension when executing machine language files such as DOS utilities and BASIC itself.

In our example, PRICE/DAT.DOLLAR:1(DOSPLUS), the third part of the filespec is ".DOLLAR", and it is called the password. A password can be given to any file to restrict access to it. You may, by using the password along with the protection level (see the library command ATTRIB), set the level of access that a user may have to the file. It may vary widely. You may require them to know the password before they can access the file at all or you may require it only if they wish to modify the file in some way. You may set up a program file so that it may be "run" only without the password. Any attempt to load, list, or otherwise modify the file would require the password.

A password may be up to eight characters in length, and can contain any non-reserved characters. It is separated from the filename or extension by a period (i.e. "."). The password, like the extension, 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 (see the library command PROT), and even then only if you know the Disk Master password.

The fourth element of the filespec is the drive specification (DRIVESPEC for short). In our example, PRICE/DAT.DOLLAR:1(DOSPLUS), the drivespec was ":1". This drivespec simply informs DOSPLUS II that the file "PRICE/DAT" the we are referring to resides on the drive currently named "1". For a further explanation of drives, read into the next portion of this section - "Devices and device specifications".

For our purposes now, let it suffice to say that a drivespec is a one or two character name that indicates which of the drives we are referring to. The standard drivespecs that DOSPLUS II was shipped with were drivespecs 0 through 7, with 0 through 3 defined as floppy drives and 4 through 7 defined as rigid drives.

The drivespec is also an optional portion of the filespec. If you do not give a drivespec, DOSPLUS will search through all the drives in the system starting with device 8 and moving on up (this is called a "global search"), until it finds a filespec that matches the one you have given.

The fifth element of a filespec is the disk name. In our example, PRICE/DAT.DOLLAR:1(DOSPLUS), the disk name was "(DOSPLUS)". This is included in DOSPLUS II solely for the purpose of providing compatibility with TRSDOS. The disk name, when included in the filespec, should be the last item and be encased in parenthesis.

Under TRSDOS, this was a valid portion of the filename and unless the disk name matched what you had specified, an error would result. No such error will result under DOSPLUS II. The system will not reject a filename that contains a disk name, for the sake of compatibility, but it will not be used for anything.

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, DOSPLUS could not tell the difference between them and the existing file would be destroyed and replaced with new data.

Three of the five parts of the filespec determine uniqueness: the filename, the extension, and the drivespec. The password does NOT. What this means is that if two filespecs have the same filename and drivespec, but a different extension, they are two distinct files. If however, two files have the same filename, extension, and drivespec, but only different passwords THEY DENOTE THE SAME FILE.

For example :

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. In this case, an ounce of caution is truly worth a pound of recovering data lost because of carelessly overwriting a previous file.

Differences from TRSDOS -

There are only three differences between DOSPLUS II and TRSDOS file specifications :

(1) DOSPLUS II removes many of the character restrictions that TRSDOS imposed.

(2) DOSPLUS II does not use the disk name.

(3) DOSPLUS II ignores case in filespecs.

The disk name portion of the filespec was used under TRSDOS to make certain that the file you were seeking was not only the proper file, but on the proper disk. We felt this to be cumbersome and not useful in most cases, so it has been removed. As stated earlier, if a disk name is specified, it will NOT produce an error. However, DOSPLUS II will also do nothing with it.

Under TRSDOS, you had to begin each part of the filespec except the drivespec with an alphabetic character (i.e. A-Z). This restriction has been removed under DOSPLUS II. Portions of the filespec may begin with numbers, punctuation symbols, almost ANY character than can be typed from the keyboard.

By including more characters (TRSDOS restricted you to A-Z and 0-9), we also give you more potential filenames.

Also, under TRSDOS, if you specified a filespec in lower case, it was a different file than the same filespec in upper case. If you forgot to lock the caps key down, this could be catastrophic. DOSPLUS II will ignore case in a filespec.

Devices and device specifications -

The DOSPLUS II system has sixteen devices built into it. For the sake of accuracy and ease of reference, you have the devices split up into two distinct groupings. The first eight devices (devices 0 through 7) are called SYSTEM DEVICES. The second eight devices (devices 8 through 15) are called DRIVE DEVICES. Following is a list of the two and whether or not it is an input or an output device :

Device	Default name	Class
Keyboard	KI	Input
Display	DO	Output
Printer	PR	Output
SIO A	CA	Input or output
SIO B	CB	Input or output
User defined	U1	User defined
User defined	U2	User defined
User defined	U3	User defined

Devices (cont.) :

First drive	0	Input or output
Second drive	1	Input or output
Third drive	2	Input or output
Fourth drive	3	Input or output
Fifth drive	4	Input or output
Sixth drive	5	Input or output
Seventh drive	6	Input or output
Eighth drive	7	Input or output

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

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

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

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

Some restrictions -

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

You must also understand clearly that although DOSPLUS II has sixteen devices and all of them are "devices", there is a great difference between the system and drive devices. They are not at all interchangable. Again, remember, you may address a FILESPEC as if it were a system device (this is called a channel). You may not address a drive as if it were a character orientated device.

Addressing devices -

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

(1) The type indicator.

(2) The device name.

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

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

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

In most cases in DOSPLUS II, you may use character orientated devices in place of filespecs. This is part of what is called "device independence". The ability to use devices and files interchangably creates a new term: "I/O channel". Throughout the manual, you will see many references to an I/O channel. Use of this term simply indicates that at that location you may specify either a filespec OR a devicespec. You may not, however, specify a drivespec when asked for an I/O channel.

Summary of device handling in DOSPLUS II -

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

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

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

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

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

(1) A devicespec.

(2) A filespec.

(3) A drivespec.

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

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

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

Detailed explanation of the command line -

The command line is the means by which you communicate with DOSPLUS II. When you are at the DOS command mode (remember that term?), you may enter up to 80 characters of text that commands DOSPLUS II to do something. This line of text is called the command line and has four parts: (1) The command, (2) The I/O field, (3) The parameter field, and (4) The optional comment field. Let's look at each of these in turn.

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

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

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

is the same as :

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

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

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

is NOT the same thing as :

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

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

For example :

T??T/B??

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

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

T??T/B*

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

T/BAS

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

T*/BAS

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

T!

is the same as :

T*/* or T**

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

COPY !:0 :1

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

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

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

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

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

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

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

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

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

DIR :0 TO @PR {ALPHA} - Prints alphabetized directory

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

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

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

Definition of terms -

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

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

Definition of terms (cont.) :

Term	Definition
Devicespec	An at sign "@" followed by a one or two character device name. Used to refer to one of the eight system devices. May only be used when character I/O is specified. It can be specified when an I/O channel is requested.
Channel	A channel is a character orientated I/O path. When a channel is requested, it is indicative of the fact that the data will be moved a byte at a time. File by file I/O is not allowed with channels. A channel may be either a filespec or a devicespec. It may NOT be a drivespec except in cases where a drivespec is only part of a filespec.
Wildmask	A filespec containing wildcard characters. Used to make the effect of a command global to several files. May not be used when a channel is requested. Consists of a filename and extension only. It can be used in conjunction with a channel, but cannot be specifed AS the channel. For further details on the use of wildmasks, see the section above - "Detailed explanation of the command line".
Parameter	An optional control field that can specify additional information on exactly HOW you want the indicated command to function. Can be a switch (On or Off), a string (passwords, etc.), or a value (buffer size, record length, number of lines per page, etc.) If the parameter is a switch, usually the mere mention of the parameter will engage it (i.e. "=Y" will be assumed).

Definition of terms (cont.) :

Term	*Definition*
Separator	Used to separate delimiters and channels, parameters, etc. Within the I/O field, separators MUST be a space. Within the parameter field, they may either be a space or a comma. If you use commas within the I/O field, DOSPLUS II will terminate the I/O field and start looking for parameters. Separators are NOT optional. For the command line to be evaluated properly, you must separate the various portions of the fields.
Delimiter	A field specifier. Will be either FROM, TO, or USING. Indicates direction within the I/O field. These may not be used as filenames (i.e. you can't call a file TO/CMD, because "TO" is a reserved word). Remember, these must be surrounded by separators. You need not actually mention these terms in the command line unless you wish to specify the various portions of the I/O field in something other that the default order (e.g. specify the destination channel before the source, etc.). If the delimiter is present, it will override any default positioning and re-route I/O any way you wish.

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

There were several references made to the term "DCB". A DCB is simply a <D>evice <C>ontrol <B>lock. An explanation of exactly what a DCB is and does is afforded in the technical manual. There are also "FCB"s. These are <F>ile <C>ontrol <B>locks and, again, the technical section will address their functions.

Built in features of DOSPLUS II

DOSPLUS II has several built in features that are activated by a series of keystrokes. They are :

Command & key sequence	Description
Screen printer (CONTROL & dash "-")	Prints contents of screen. Useful in providing instant hardcopy of current display.
Repeat last command (Slash "/" & ENTER)	Repeats last DOS command. Useful in performing multiple executions of the same function.
Multiple commands (command;command)	Enter more than one command on a line. Placing the semi colon between the commands forces a carriage return.
Pausing commands (HOLD)	This allows you to suspend a command's output. Most DOSPLUS II commands that cause a great deal of output can be paused by pressing the HOLD key. To start them again, press the HOLD key again. Useful to pause file listings, etc.
Aborting commands (BREAK)	This allows you to abort a command. Most DOSPLUS II commands will also allow you to abort before the command is complete. Pressing the BREAK key should return you to the DOS command mode. Useful in aborting file listings, etc.

These built in features are resident at almost all times and can be accessed simply be pressing the proper sequence of keys. Under certain special circumstances, though, these may be disabled. That will of course be documented individually with the command that is disabling it.

DOSPLUS II Library of commands

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

APPEND	(Append two I/O channels together)
ATTRIB	(Alter file's attributes)
AUTO	(Set auto execute command)
BOOT	(Execute system "cold-start")
BUILD	(Create command chain file)
CAT	(Display diskette's file catalog)
CLEAR	(Clear user memory and files)
CLOCK	(Turn on/off system clock display)
CLS	(Clear screen)
CONFIG	(Alter system configuration)
COPY	(Copy device/file to device/file)
CREATE	(Create and pre-allocate disk file)
DATE	(Display or change system date)
DEBUG	(Activate system memory monitor)
DIR	(Display detailed file listing)
DO	(Execute command chain file)
DUMP	(Save memory to disk file)
ERROR	(Display detailed error message)
FILTER	(Filter I/O to/from specified device)
FORMS	(Alter printer driver parameters)
FREE	(Display free space data on drive)
I	(Initialize disk swap)
KILL	(Kill specified device or file)
LIB	(Display list of library commands)
LINK	(Join two logical devices)
LIST	(List file to device)
LOAD	(Load Z80 object file into RAM)
PAUSE	(Pause execution of a DO file)
PROT	(Alter disk's protection status)
RENAME	(Rename a device or file)
RESET	(Restore device to default driver)
ROUTE	(Re-direct I/O to device/file)
SCREEN	(Send contents of screen to device)
SET	(Set device driver)
SETCOM	(Configure communications interface)
SYSTEM	(Customize your operating system)
TIME	(Display time or set system clock)
VERIFY	(Toggle automatic read after write mode)

APPEND

This command allows you to append a device or a file to the end of another device or file. To append a device to a device is the same as copying that device to the other. You may, if you wish, append a device to a file and specify load module format (this frees you from restricting APPEND to data files and ASCII programs).

==========

The command syntax is :

APPEND [FROM] channel1 [TO] channel2 {param}

"channel1" is the source file or device specification. This may be any valid file or device specifier. This is the device or file that will be appended and must be an input channel.

"channel2" is the destination file or device specification. This also may be any valid file or device specifier. This is the device or file being appended to and must be an output channel.

"{param}" is the optional action switch. You have two.

The parameters are :

CMD=switch	Appends to destination device or file in load module format (i.e. a /CMD file). When you append with this option, the last four bytes of the destination file will be overwritten in order to strip the old transfer address. The transfer address of the appended file will be used.
STRIP=switch	Backspaces one byte from the end of file on the file being appended to. This is useful in stripping off "end-of-file" markers on data files before appending the new data to it.

Note that both of these parameters require that the file's End-of-file byte be correctly set. This is covered in greater detail in the "Finally" section of this command.

Abbreviations :

As with all DOSPLUS II commands, the [FROM] and [TO] delimiters are wholly optional on this command. You may delete either one of them or both. For example :

APPEND channel1 channel2 {param}

accomplishes the same exact results as the earlier example.

The parameters may be abbreviated as "C" for "(C)MD" or "S" for "(S)TRIP". Of course, as with all DOSPLUS II commands, these are NOT case dependant. Lower case works just as well as upper. Therefore :

APPEND channel1 channel2 {CMD}

is the same as :

APPEND channel1 channel2 {C}

which is the same as :

append channel1 channel2 {c}

==========

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

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

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

APPEND also has an optional switch to append to the destination file in load module format. Load module format is simply the format used to write a machine language program to the disk so that it can be loaded in later. Every instruction in the program is accompanied by indicators showing where in memory it will load. Because you may now append in load module format, you may in fact append one machine language program on to the end of another. The instructions in the appendage, if they conflict with instructions in the initial file as to memory location, will overlay those instructions from the initial file.

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

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

PLEASE use extreme caution when appending devices. As with any system this flexible, it can be mis-used and "hang-up" the system. Think through your logic carefully when appending devices.

Examples:

```
APPEND FROM DATAFIL1 TO DATAFIL2
```

This command will take all the data in "DATAFIL1" and append it to the end of "DATAFIL2". You could have abbreviated that command as : APPEND DATAFIL1 DATAFIL2.

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

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

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

And the file "NEWMOD/BAS" contained the line :

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

After you had saved both of these in ASCII (c-f: TRSDOS manual Disk BASIC section) and executed the above APPEND command, the next time that you loaded in the file "OLDPROG/BAS", you would get the following :

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

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

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

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

APPEND PATCH/CMD PROGRAM/CMD {C}

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

APPEND @KI DOCUFILE/TXT:1

This command will append any further data that is input from the keyboard (i.e. any further keystrokes) on to the end of the file "DOCUFILE/TXT" that is located on drive one. This would allow you to append further instructions onto the end of a build file, for example.

APPEND TO SERIAL/DAT:0 FROM @CA {S}

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

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

Finally:

Remember, in all of times that you will use APPEND, the source file will remain completely unaffected.

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

Please remember that you MUST append from an input channel (source) to an output channel (destination). A list of default devices and their names and classes is available in the operations section of this manual. A disk file may function as either an input or an output channel.

Another important note on using APPEND is that the parameters (STRIP and CMD) will not function properly if the file's End-of-file byte is not set correctly. Files that have been transferred from TRSDOS using CONV (see the utility program CONV) may exhibit this syndrome. To correct the problem, load the file with the OFFSET program (see the utility program OFFSET) and save it back into itself. OFFSET will them set the End-of-file byte properly for DOSPLUS II.

Files that are created under DOSPLUS II will always have the end-of-file byte set correctly.

ATTRIB

This command allows you to set a file's "attributes". A file's attributes include its passwords, protection level, whether it is a system or user file, visible or invisible, non-shrinkable, or modified. By using this command, you may change these attributes for any file to which you have access.

==========

The command syntax is :

ATTRIB wildmask {param=exp,param=exp...}

"wildmask" is the standard DOSPLUS II wildmask file specification that designates the file or group of files that we are referring to. If the file is currently password protected, you must specify the password in this.

"param" is the optional parameter to be set or altered. These are listed below.

"exp" is the expression which gives the new value. Depending on the parameter, it may be a switch (Y or N), a string, or a numeric value. The default values are listed below.

The parameters for ATTRIB are :

PW=string	Disk Master password. If you are using ATTRIB on a number of files with a wildmask, then some of them may be protected. In order to have access to these files, you must specify the Disk Master password.
ACC=string	Access password. Can be up to any eight non-reserved characters. This is the password that will be required before the file may even be accessed.
UPD=string	Update password. Again, can be up to any eight non-reserved characters. This is the password that will be required before the file is modified.
PROT=value	Protection level. A value 0-7 that indicates what level of access is permitted a particular file. Must be used in conjunction with a password.

INV=switch	Invisible file. This option allows you to make a file invisible. That means it will not display in a standard directory or file catalog. The "I" option must be used to view these files (see the library commands DIR and CAT).
KEEP=switch	Non-shrinkable file. If a file posesses this attribute, it will never decrease in size on the disk. Useful in preserving pre-allocated sequential files' disk space.
MOD=switch	Modification flag. This flag indicates the file has been modified since it was last copied or backed up. You may use this parameter to manually set or remove this flag.

The default values described below are those values that the parameter defaults to if specified in the command line without an accompanying expression. They are :

ACC	No default. If included in a command line, an expression MUST be included specifying the password.
UPD	Same as ACC.
PROT	No default. If PROT is included in the parameter field, a protection level MUST be specified.
INV	On. If this option is included in the parameter field, the file will be made invisible. If this is already the case, the file will remain unchanged.
KEEP	On. If this option is included in the parameter field, the file will be set as non-shrinkable. If this is already in effect, file status will remain unchanged.
MOD	On. If this option is included in the parameter field, the file's mod flag will be set. If it is already set, it will be unchanged.

Each of these parameters may be abbreviated by specifying only the first letter of the parameter. ACC=A, UPD=U, PROT=P, INV=I, KEEP=K, MOD=M. For example :

ATTRIB filename {INV}

and

ATTRIB filename {I}

are the same command.

==========

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

A protection level is useless unless a password has been set for that file. You see, if no password has been set, then in effect no password IS the password. The default password for any file is a series of blanks. This is also the default when no password is specified with the filespec. Therefore, when the user omits the password, they have in actuality SPECIFIED the correct password and they are allowed full access to the file.

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

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

You have several protection levels to choose from. You refer to them and set them by their numbers. This list will illustrate those that you have a choice of :

Number	Name	Protection level
0	Full	No protection set. Total access.
1	Kill	Able to delete file.
2	Rename	Rename, Write, Read, Execute.
3	----	Not used at this time.
4	Write	Write, Read, Execute.
5	Read	Read, Execute.
6	Execute	Execute only.
7	None	No access. Not a user option.

Protection level 1, kill, allows you complete access to a file. You may kill it, rename it, write to it, read it without executing, or execute it. The advantage to using level 1 instead of just omitting a protection level (thus setting level 0) is that it lets the user know that the file is question was protected but they are specifically allowed into the file.

Protection level 2, rename, allows you to do everything to a file EXCEPT kill it from the disk. Protection level 3 is not implemented in this release of DOSPLUS II, but ATTRIB will allow you to set this level. Protection level 4, write, will allow you to write to a file or load it without executing, but you may NOT rename the file or kill it.

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

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

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

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

When a file is invisible, it does not get displayed via a normal directory display. In order to see these files, you must specify the "{INV}" option from the DIR command (see the library command DIR). This is very useful when a file is a permanent part of your working DOS system and you do not wish to see that filename constantly displayed when you list the disk's directory. This option affects only that area. Simply because a file is invisible doesn't mean that it is protected. You must set all those items independantly.

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

The final parameter that you can alter with ATTRIB is the modification flag (MOD FLAG for short). This is one of the most useful items in DOSPLUS II's directory. This flag tells you when a file has been updated since you last copied it or backed up the disk that it resides on. Updating a file refers to writing to the file. If you simply read from a file, you have done nothing to alter that file, therefore the mod flag is not set.

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

This parameter may, from time to time, need to be set or reset manually. ATTRIB allows you to do that.

Examples:

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

These two commands will have the same effect. In this example, we are addressing the file "UTILITY/PRG" located on disk drive ":1". We are setting the update password to "PASSWORD", the protection level to "6" (execute only), and making the file invisible. Note that the access password was NOT set. This will allow you to run the program without knowing a password, but you may not modify or in any way examine the code without using the update password.

```
ATTRIB FILE {MOD=N}
ATTRIB FILE {M=N}
ATTRIB FILE,M=N
```

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

```
ATTRIB PAYDATA:AA {KEEP=Y}
ATTRIB PAYDATA:AA {K}
ATTRIB PAYDATA:AA,K
```

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

Finally:

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

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

For example :

```
ATTRIB FILE {I}
```

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

```
ATTRIB FILE/* {I}
```

will search out all files that have the filename "FILE" and any extension and make them invisible, but it will only do this for Drive 0.

AUTO

This command allows you to set an "automatic command" to be executed upon boot-up of the system. This may be a library command, the name of a configuration file, or a program. The maximum number of characters allowed in this statement is 32. If you require more that that, you will have to use a "DO" file (see the library commands BUILD and DO) and have AUTO call this file.

==========

The command syntax is :

AUTO drivespec command

"drivespec" is the optional drive specifier that tells DOSPLUS II which disk you wish to store this AUTO command on. If omitted, the current system drive is used.

"command" is the AUTO command that you wish executed upon power-up. The first 1 or 2 characters of this command can indicate the type of AUTO to be used.

Command switches :

! If the exclamation mark appears as the first character of an AUTO command, the non-breakable AUTO will be used. Under normal circumstances, you may prevent the AUTO from being executed by holding down the ENTER key as the system is booted. For a non-breakable AUTO, you do not have this option.

If the pound sign appears, either by itself or with the exclamation mark, as the first character(s) of the AUTO command, then the invisible AUTO is used. Normally, the AUTO command is displayed as it is executed. If this option is in effect, this will not be the case. The command will execute unseen.

Please note that these command switches may appear in either order (!# or #!) if you wish to specify them both. To reset an AUTO, enter the AUTO command (with the drivespec, if you like) without specifying a command. For example :

AUTO

would reset the AUTO command currently set on the system drive. To boot the system avoiding the AUTO, hold down the ENTER key as the system is reset.

==========

The AUTO command allows you to define special configuration files with SYSTEM (see the library command SYSTEM) and load them in automatically upon power-up. By using the invisible option, you don't even have to SEE this procedure.

AUTO may be used with any valid library command (or series of library commands) under 32 characters in length. If you use the multiple command feature (i.e. command;command), AUTO will write these commands to the disk exactly as you enter them, length permitting. For example, "AUTO LIB;FORMS" would write that to disk so that when you booted the system the commands "LIB" and "FORMS" would be executed. It will NOT write "AUTO LIB" to the disk and then execute a "FORMS" command. Remember, the total length of the AUTO command must not exceed 32 characters.

As was explained earlier in the operations section, you may enter multiple commands on the same line as long as you separate these commands with a semi colon ";". This forces a carriage return and enters the command to that point. This means that you may actually have two or more commands imbedded in your AUTO statement as long as the TOTAL length of the command does not exceed 32 characters.

AUTO may best be summed up as saying that you may use for three main purposes :

(1) To load a configuration file (see the library command SYSTEM).

(2) To execute a program from boot-up.

(3) To begin an automatic command chaining file (see the library commands BUILD and DO).

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

Examples:

AUTO SYSCON

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

AUTO !SYSCON

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

```
AUTO #!SYSCON
```

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

```
AUTO :1 DO START
```

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

```
AUTO :B
```

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

Finally:

When using the AUTO command on a system that is going to require the loading of a special configuration file (see the library command SYSTEM), you must execute the configuration FIRST. For example, let's assume that we have a configuration file named "S2201/CMD" on our system disk. We want to, upon boot-up, execute this file and run a BASIC program called "MENU". This would have to be handled at the time the configuration file was created.

When you use the SYSTEM command to create your configuration files, you may specify any additional statements that you wish executed with this file. You enter these as multiple commands with the ";" separating them.

For example, to save the configuration file "S2201/CMD", you would have executed a statement similar to this :

```
SYSTEM {SAVE='S2201'}
```

To have the system execute this file and then DO (see the library command DO) a file called STARTUP/TXT, you would type this instead :

```
SYSTEM {SAVE='S2201'};DO STARTUP
```

The AUTO statement would look like this :

```
AUTO S2201
```

When you re-booted, AUTO would execute the file "S2201/CMD", which would configure the system and the begin the DO function with the file "STARTUP/TXT".

Remember, to boot-up the system avoiding the AUTO, hold down the ENTER key. Also remember that this will not work if the non-breakable AUTO (I.E. "!") Is selected.

BOOT

This command will allow you to perform a cold system reset from software. Calling this command from your program or typing it in from keyboard produces the same results as pressing the reset key on the face of the Model II.

==========

The command syntax is :

BOOT

There are no parameters for this command.

==========

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

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

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

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

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

BUILD

This command offers you the ability to create an ASCII text file on the disk. These can be used for "DO" files, patch program files, or filter files (or anything that requires an ASCII text file).

The command syntax is :

BUILD filespec {param=switch}

"filespec" is the standard DOSPLUS II file specification that you have selected to contain your text. This may optionally contain a drivespec, if you wish to build this file on something other than the system drive. If you don't specify an extension, DOSPLUS II will automatically add the extension "/TXT" to it.

"param" is the optional command modifier. It will affect the action of the command when included.

Your parameter is :

APPEND=switch	Optional switch to indicate that you wish to append the instructions you are about to enter on to the end of an already existing file. May be either "Y" or "N". If you do not specify "Y", then BUILD will re-use any space allocated to that file.

Default value :

APPEND	No. BUILD will re-use the file space. However, if the parameter appears in the command line with no switch, BUILD will assume that you wish to append.

This parameter may be abbreviated to the letter "A". For example :

BUILD filespec {APPEND=Y}

and

BUILD filespec {A=Y}

are equivalent.

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

Any ASCII text file may be used for these applications. You must only remember never to exceed more than 79 characters without a carriage return.

This means that you may also create these files from BASIC or machine language applications programs (such as a word processor). Therefore, your programs could create the needed files based on information gleaned from the user and the user would never actually interface with the DOS.

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

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

```
Enter text (79 chars/line)
```

At that point you are free to type up to 79 characters of text. When you have finished typing a line, press ENTER to store that line. When you are finished, press BREAK at the next blank line and BUILD will return to DOSPLUS II. Should you press BREAK without pressing ENTER, the line will be stored in the file without a terminating carriage return. This means that the partial line will be placed on the screen but never actually entered.

Examples:

```
BUILD TEST:0
```

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

```
BUILD TEST:0 {APPEND=Y}
BUILD TEST:0 {A=Y}
BUILD TEST:0,A
```

These three commands will all have the same effect. They also will open a file "TEST/TXT" on drive ":0", but if this file already exists; they will append the new commands to the end of the file without over-writing the current contents.

```
BUILD STARTUP/BLD:0 <Enter>
Enter text (79 chars/line)
FORMS {W=80} <Enter>
BASIC MENU/BAS-F:1-M:65000 <Enter>
<Break>
```

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

```
DO STARTUP/BLD
```

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

Finally:

When using BUILD and DO, if you wish to print a line if instructions or comments on the screen, you may do so. Any line that begins with a period "." will not be executed by DOSPLUS II. Therefore, to place non-command lines into your DO file, simply start them off with a period. For example, ".Insert the #1 disk" is a comment line and "Dir :4" is not.

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

The patch utility uses a default extension of "/PAT" and the FILTER command uses a default extension of "/FLT". When creating text files for these two, you may want to specify their default extensions when you call BUILD to begin creating the file. For the most part, you will be using BUILD to create files for the DO command, therefore BUILD itself uses the same default extension as DO (i.e. "/TXT").

Also, when entering lines into a file, you may press BACK SPACE to delete a character and ESCAPE to delete a line. No other editing functions are supported.

CAT

This command will display a disk's "file catalog". A file catalog differs from a directory in that it contains ONLY the filename and extension while the directory contains a great deal of information about each file. You also have the option of specifying a wildmask so that only filespecs that match certain criteria are displayed.

The command syntax is :

CAT [FROM] drivespec [TO] channel [USING] wildmask {param=exp...}

"drivespec" is the name of the drive for which you desire the file catalog. If it is not specified, CAT will globally display the file catalogs of all drives.

"channel" is the optional output channel. That is, where you want the file catalog to be sent. If it is not specified, the screen (i.e. TO @DO) is assumed.

"wildmask" is the optional wildmask to restrict CAT to a certain group or class of files. If it is not specified, all files will be displayed (i.e. "!" is assumed).

"{param=exp}" is the optional action parameter that indicates what type of file catalog you want to see. If no parameters are given, the default values listed below will be in effect.

The parameters are :

SYS=switch	File catalog will contain system files as well as standard entries.
INV=switch	File catalog will contain both visible and invisible user files.
KILL=switch	File catalog will contain names of any deleted files not yet wiped from the directory or over-written by an active file.
ALPHA=switch	File catalog will be displayed in alphabetical order.

Default values. If any of these switches are specified in a command line without an expression, DOSPLUS II will assume "Y" and act accordingly. The default values listed here are those that are in effect when the parameter is NOT present in the command line.

SYS	No. User files only will be displayed.
INV	No. Visible files only will be displayed.
KILL	No. Active files only will be displayed.
ALPHA	No. File listing will be in the order that the filenames actually appear in the directory itself.

The FROM, TO, and USING delimiters may be omitted unless you wish to specify the various portions of the I/O field in a non-standard order. The parameters may be abbreviated :

SYS=S, INV=I, KILL=K, and ALPHA=A

==========

This command will display only the filename and extension for the files stored on the specified disk. Under many circumstances, that is all you are interested in anyway. This command may actually be used more than DIR by the average user.

The ability to specify an output channel when using CAT means that you may direct the file catalog to the printer, the serial port, a disk file, or whatever is a legal I/O channel. If you remember from the operations section, an I/O channel can be almost anything except for a drivespec (devicespec, filespec, etc.).

The ability to use the wildmask with CAT means that if you wish to view all the files with a "/CMD" extension on drive ":0" or all the files with a "/CMD" extension on all the drives, it is a simple procedure.

CAT is very useful in finding which drives a specific program is on. Under DOSPLUS II, you have the option of (on certain types of drives) up to 256 user files. That can be an awful lot of names to read through looking for the one you want. This can make it cumbersome to locate a single file. By setting a wildmask specific enough to weed out any extraneous files, you may speed up the search tremendously.

The simplest form of CAT is :

```
CAT
```

Which will display a file catalog of all visible user files on all drives. Next simplest would be :

```
CAT :1
```

Which has the same effect, but restricts itself to those visible user file located in the disk drive named ":1".

Your output should look something like this :

Files drive: "ds" "disk name" -- "disk date"

filename/ext filename/ext filename/ext filename/ext

"ds" is the drivespec either you selected or it assumed, whichever the case may be. At any rate, it is the drivespec for which the file catalog being displayed is from.

"disk name" is the name of that disk and "disk date" is the creation date that is stored on the directory.

Your filespecs will be displayed four across in the format "filename/ext", as shown above. If the file has no extension, just the name will be displayed.

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

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

```
CAT @PR
```

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

```
CAT TO @PR
```

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

```
CAT :1 @PR
```

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

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

is the same thing as :

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

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

```
CAT TEST/DAT
```

will produce an error, while :

```
CAT USING TEST/DAT
```

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

Examples:

```
CAT :0 {SYS=Y,INV=Y,KILL=Y}
CAT :0 {SYS,INV,KILL}
CAT :0 {S,I,K}
CAT :0,S,I,K
```

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

```
CAT USING PER/DAT
```

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

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

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

```
CAT :1 {INV=Y,ALPHA}
CAT :1 {I,A}
CAT :1,I,A
```

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

Finally:

Remember that the default output channel is the display. If you omit the output channel, then the file catalog will be sent to the screen. This can cause a problem with your wildmask field if you are not careful.

You see, the wildmask field follows the output channel field in the command line. If the following four conditions are true :

(1) You have specified a source drivespec so that the field is full in that position.

(2) You have omitted the output channel in order to send the output to the default channel: the video.

(3) You have omitted the delimiter USING because you feel that the wildmask is in its proper place.

(4) You do not have any wildcard characters in your wildmask that might show DOSPLUS II that it IS a wildmask.

then DOSPLUS II will overwrite the file you have specified in the wildmask with the output of the catalog. Without the USING delimiter and with the output channel not specified, the wildmask moves into the output channel area. Because the source drivespec WAS specified, it won't try to use the wildmask for that and generate an error there. Finally, because there are no wildcard characters to indicate a wildmask, DOSPLUS II takes it as a standard filespec. Since a filespec is a valid output channel, it gets overwritten. Therefore, a little bit of carelesness can destroy the very file you were looking to find.

What all this means is, if you are going to use a wildmask with the source drivespec specified and you are NOT going to specify an output channel and your wildmask does not have any wildcard characters in it, then you must use the USING delimiter.

For a more detailed explanation of wildcards and wildmasks, consult the operations portion of the manual under "File and device specifications".

CLEAR

This command allows you to fill either a file or user memory with user defined data. It is used to get a "clean slate", so to speak. Please note that this command operates in two separate and distinct manners. The two modes are mutually exclusive (if you are clearing a file, you can't clear memory and vice versa). There are two completely different command syntaxes given. Each should be treated as an individual. You may not combine the parameters from the memory CLEAR with the file CLEAR and vice versa.

==========

The command syntax is :

CLEAR filespec {param=exp}

or

CLEAR {param=exp...}

"filespec" is the standard DOSPLUS II file specification that will be used to indicate to CLEAR that you wish to operate on a file and direct it to the specific file that you wish to clear. If a filespec is NOT present, then CLEAR will assume that you are using the second form (clearing memory).

Your parameters are :

START=value	This is the memory address that you wish to begin filling memory at. If omitted, the value set for LOMEM will be used. This value may not be lower than LOMEM. This parameter applies only to the second form of CLEAR (clearing memory).
END=value	This is the memory address that you wish to fill memory up to. If omitted, the value set for HIMEM will be used. This value may not exceed HIMEM. Again, this parameter only applies to the second form of CLEAR (clearing memory).

DATA=value	This is the optional value that you wish to fill memory with. It may be expressed in any base. Remember, if you use a base other than decimal, you will need to affix the value type suffix (i.e. 7FFFH). If no value is specified, zero will be assumed. This parameter applies to either form of CLEAR (clearing a file or memory).

Default values :

START	LOMEM.
END	HIMEM.
DATA	00.

Abbreviations:

START=S, END=E, DATA=D

==========

This command is used whenever you wish to clear out memory without resetting the system or wish to clean up a file's disk space. The two modes of this command, as mentioned earlier, are mutually exclusive. You cannot mix memory and file clearing within this command.

Software developers will find in most convenient to be able to clean out memory or a file when they are writing programs. The ability to clear out a file will allow you to "start over" with fresh data space so that you can see what the program has written to the disk this time. When debugging machine language programs, it is always most convenient to be able to be assured that the program you are working on is the only thing in memory.

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

Examples:

```
CLEAR
```

This example will fill all of user memory (the area between the LOMEM and HIMEM values) with zeros.

```
CLEAR {START=5000H,END=7000H,DATA=6CH}
CLEAR {S=5000H,E=7000H,D=6CH}
CLEAR,S=5000H,E=7000H,D=6CH
```

These three commands are equivalent. All three of them will fill memory between addresses 5000 hex and 7000 hex (inclusive) with the value 6C hex.

```
CLEAR TESTFILE/TXT
```

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

```
CLEAR DATA:TD {DATA=229}
CLEAR DATA:TD {D=229}
CLEAR DATA:TD,D=229
```

These three commands will all accomplish the same thing. They will search the drive named ":TD" for the file named "DATA". If the file is located, CLEAR will fill it with the value 229 dec (E5 hex).

Finally:

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

Another thing to bear in mind is that if you wish to erase a file's data before you kill it from the disk, it will be necessary to go through the extra step of clearing the file first.

Also, please note that when specifying values for the "Data" parameter, either one or two byte values may be used. If you specify a one byte value, then the byte will be duplicated. That is, whatever one byte value you specify will be used for BOTH prats of the two byte fill value.

CLOCK

This command allows you to turn on and turn off the system clock. When turned "ON", the system clock will be displayed in the upper right hand corner of the screen. It will be in the "HH:MM:SS" format.

==========

The command syntax is :

CLOCK switch

"switch" is the optional action parameter to inform DOSPLUS II whether to turn the clock display on or off. ON will be assumed.

Your switches are :

ON=switch	Display on.
OFF=switch	Display off.

==========

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

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

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

Examples :

```
CLOCK ON
CLOCK
```

This command turns on the clock display.

```
CLOCK OFF
```

This command turns off the display.

CLS

This command clears the display. It can be used to erase information from the screen that you don't want others to see or simply to remove cluttered data. This is also used to clear the screen before execution of the library commands, since they don't clear it automatically.

The command syntax is :

CLS

There are no parameters for this command.

This command, when executed, will cause the display to be cleared immediately. It will NOT, however, reset the current video mode. For example, if you are in the double wide (40 characters per line) mode and you execute a CLS, the screen will clear but you will still be in the double wide mode.

Under DOSPLUS II, the library commands do not automatically clear the screen before execution. The reason is simple. Because DOSPLUS II is device independent, any device may be specified for the output channel, not just the video. That makes sending a CLS first impractical. Suppose you execute a "CAT TO @PR". You wouldn't want to send a CLS instruction to the printer. Because the devices could be completely renamed and re-routed, it is impractical to attempt to sense when the display is being output to (as a special case).

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

DIR

would become :

CLS;DIR

You may also use this command during a DO file to erase information from the screen if you do not want the operator to see it. If, for example, you were creating a password protected file and you did not want the operator to see the filename, you could place a CLS at the beggining of the command line. Such that :

```
CREATE DATAFILE/DAT.PASSWORD
```

would become :

```
CLS;CREATE DATAFILE/DAT.PASSWORD
```

The statement would be on the screen for only a split second, far too briefly for anybody to read it. This measure of protection allows you to use a "confidential" command chaining file.

Examples:

```
CLS
```

This command will clear the screen.

```
CLS;FREE :0
```

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

CONFIG

This command allows you to supply DOSPLUS II with certain information about your system's disk drives. DOSPLUS II maintains a set of "DCT"s (<D>rive <C>ontrol <T>able) that supply it with information about the drive such as density, number of sides, and cylinder count. However, there are certain parameters that the DOS has no way of ascertaining unless you specify them. CONFIG allows you to specify these. CONFIG also will display the current parameters for each drive's DCT.

The command syntax is :

CONFIG [drivespec] {param=exp,param=exp...}

"drivespec" is the drive specification that indicates to DOSPLUS II which drive you are CONFIGuring. Consult the section "File and Device specifications" earlier in this manual for a more detailed definition.

"param" is the optional action parameter that indicates what aspect of the drive you are CONFIGuring.

"exp" is the modifier for the parameter. Its value will modify the action of the parameter. Depending on which parameter we are modifying, it may be a switch ("Y" or "N") or a value.

You have two distinct sets of parameters. One for floppy drives and one for rigid drives. They are :

Floppy drives :

FLOPPY=switch	Indicates that the drive being configured is a floppy disk drive.
WP=switch	Sets the software write protect for that drive.
MD=switch	Sets the motor on delay for that drive. Used with Model 16 computers.
HL=switch	Sets the head load delay for that drive. Used with Model II internal drives.

STEP=value	Sets drive track to track step rate in milliseconds. Values are : 0 - 3ms. 1 - 6ms. 2 - 10ms. 3 - 15ms.
SKIP=switch	Sets "skip" parameter for that drive. This instructs the system to read every other track on the disk.
SIZE=value	Sets the size for that drive. Allowable sizes are "5" or "8" for five and eight inch.
PD=value	Physical drive. This points to which actual drive in the chain that the drivespec you are configuring will address. You change this parameter to alter the order in which the drives are scanned.

Rigid drives :

RIGID=switch	This indicates to the system that the drive being configured is a rigid drive.
SIZE=value	Informs DOSPLUS II as to the actual size of the disk drive (5 or 8 inch platters).
FIXED=switch	Indicates whether the rigid drive being configured is a fixed or removable platter drive.
WP=switch	Sets the software write protect for that drive.
STEP=value	This sets a relative value for the step rate of the hard disk. This value will not be necessarily the millisecond step rate for that drive, but is usually some value output to the controller to indicate the manner of stepping desired for this unit.
HO=value	Head offset. This value tells the system at which physical read/write head this logical drive begins.

CO=value	Cylinder offset. This value indicates at what cylinder the logical drive starts at.
HC=value	Head count. This value indicates how many actual read/write heads are in a logical drive.
TS=value	Track size. This value indicates how many sectors there are per track on the rigid drive.
PD=value	Physical drive. This parameter indicates which actual drive in the chain this logical drive is addressing. You alter this parameter to change the order in which the drives are scanned or to partition a single rigid drive into several logical drives.

Default values. In the case of a command like CONFIG, there aren't really any default parameters. Each system has different parameters that are considered normal. The hard disk parameters, for the most part, will be set by the drivers included with DOSPLUS II for your particular hard disk. We will explain not default, but optimum settings as we go through each parameter in detail.

Abbreviations :

Floppy drives :

FLOPPY	F
WP	W
MD	M
HL	H
STEP	S
SIZE	No abbreviation.
SKIP	No abbreviation.
PD	P

Rigid drives :

RIGID	R
WP	W
STEP	S
HO	H
CO	C
HC	No abbreviation.
TS	T
PD	No abbreviation.
FIXED	No abbreviation.
SIZE	No abbreviation.

Displaying current settings -

To get a display of your current system configuration settings, simply type "CONFIG" without any parameters and press ENTER. The display will show, in ascending order, the eight drive devices and those settings currently in effect for them.

Any drive devices that are not currently enabled will be displayed as "Nil". If you attempt to access one of these, you will be rewarded with an error message. To initialize that drive into the system, you use the SET command (see the library command SET). You would use the syntax "SET :dn :dn" (where ":dn" is the drivespec you are engaging.

Your system comes with only the floppy disks initialized. To engage the hard disks, use SET in the manner described above to "turn on" drives 4 through 7. If you wish, you may later alter the order of the drives as explained below (see the parameters FLOPPY, RIGID, and PD).

For a more detailed and technical discussion of this procedure, consult the SET command.

The CONFIG display line -

The drive device numbers are displayed in the farthest left hand column and are preceded by a "$". There will be eight of them, numbered 8 through 15. These devices and their order will never be changed. When doing a global search, the DOS will always scan beginning with device 8 and proceeding through device 15. Whatever order you have set the disk drives in, in relation to this device order, is the order in which the drives will be scanned.

By far, the most important thing to remember, though, is that device 8 must ALWAYS contain be the system drive. To point that device to another, non-system, disk is to insure a great deal of problems.

The second column in the CONFIG display line is the drivespec. These are the two character names assigned to the various disk drives. By default, they are named ":0" through ":7", numbered in ascending order. This drivespec is only the name by which you reference the drive from program. It has nothing to do with the actual operation of the drive.

The user may rename the drives to anything he desires (see the library command RENAME) within the limits of the reserved character table printed in the "File and Device specifications" section of the manual. The only restriction is, you cannot have two disk drives with the same name. If you wanted to swap two drive names, you would have to assign one a temporary name.

After the drivespec is the write protect indicator. If you have set the WP parameter to software write protect that drive, the letters "WP" will appear after the drivespec.

The next piece of data given in the display line is drive size. DOSPLUS II supports two sizes of disk drives, 8 inch and 5 1/4 inch. In the floppy disks, at this writing, there was no manner of attaching 5 1/4 inch drives to the Model II/16. Therefore, the option is present in the system, but reserved for future releases should the appropriate adapters become available. However, many of the hard disk units supported by DOSPLUS II use the 5 1/4 inch rigid drives. For these units, the parameter is needed.

You set drive size via the "SIZE" parameter (explained later). The display will change to reflect 8" for 8 inch drives or 5" for 5 1/4 inch drives. Remember, this is currently implemented only for the rigid drives. Setting the SIZE parameter for a floppy drive at this time does nothing.

The next statement in the display line is the media type. This will serve to indicate what you have configured that drive as (i.e. Floppy or hard drive). It will be displayed by either the word "Floppy" or the word "Rigid". You alter this via the "Floppy" and "Rigid" parameters, which are detailed later.

You have eight drive devices in all. These eight devices can contain any combination (floppy/rigid) of drives. A rigid drive may have more than one drive device assigned to it. This is called "partitioning" the drive. Floppy disk drives may not be partitioned, although more than one device may point to the same physical drive. In those cases, you are merely addressing the same disk drive via two different drivespecs. This is NOT advisable if you wish to insure the integrity of the data contained on that disk.

The next information given is the physical drive number. It is an important parameter, because along with the Floppy or Rigid parameter, it controls which actual drive the drive device will address. It will be shown in the display line as "PD=nn", where "nn" is the physical drive number.

For example, if you set device eight as a floppy drive via the "floppy" parameter and as physical drive 0 via the "PD" parameter, that device will then address your first floppy disk drive. Since device eight is the system drive, you will have to have a system disk in drive 0 for the DOS to operate.

If you set device nine as a rigid drive and assigned it physical drive number 0, it would address the first hard disk drive. Then, when DOSPLUS II would perform a global search, it would first go to device eight. This would send it to the first floppy disk drive. Then DOSPLUS II would scan device nine, which would address the first hard disk.

You can see how easy it is to "customize" your system. If you have two floppy disk drives and one hard disk, and you want the hard disk to be in the third drive position, just configure it that way. Set the third drive device, device ten, as rigid and physical drive 0. If you want the hard disk to be the system drive, just configure device eight as the first hard disk. Please note, however, that you MUST have already formatted and installed the system files on the hard disk prior to this configuring to avoid a system crash.

Throughout all of this re-routing, the drivespecs didn't change. They always remained what they are unless you rename them. It doesn't matter which drive device addresses which physical drive or which drivespec addresses that drive device.

Any drive positions that are currently unused may be deleted from the table of drives by using the KILL command to kill that drive device (see the library command KILL). For example, "KILL :2", will delete the drive that the device named ":2" addresses. These will be displayed as "Nil" in the CONFIG display line. You may want to remove any unused positions in order to speed up DOS operation slightly. If the system knows that a drive is disabled, it will not attempt to access it during a global search. This will result in slight speed increases in some operations.

The next piece of information contained in the CONFIG display line is the drive cylinder count. This information is stored in the drive's DCT (<D>rive <C>ontrol <T>able) and is actually written on the disk itself. If you have not accessed that drive yet, then the information displayed there will be the default values (77 cylinders for a floppy drive or 200 cylinders for a rigid drive). Once the system has accessed that drive and read its DCT, though, the information will be accurate for the way that that disk was formatted. This will be displayed in the CONFIG line as "Cyls=nnn", where "nnn" is the number of cylinders to which the drive was formatted.

If you wish to force DOSPLUS II to read the DCT information for all drives in the system such that the display will be accurate for the way the system is currently set up, you may use the "mount" parameter of the "I" command (see the library command I). Otherwise the system will adjust its configuration in memory as it accesses the various drives.

Also remember that any configuration that was in effect when you did a "SYSTEM {SAVE=}" will be, in effect, the new default values. When you load that configuration file, the DCTs in memory for all the drives will be set to whatever their last known values were (those at the time of the system save).

It is at this point that the CONFIG display line will take two completely different appearances. One for the floppy disks and one for the rigid. We will cover each in turn.

Floppy disks -

The next piece of information given in a floppy disk configuration display is the number of sides. Once again, this information is stored in the disk's DCT and recorded on each disk. DOSPLUS II will pick up this information automatically the first time that it accesses those disks. This is merely a display. It will be shown as "Sides=nn", where "nn" is the number of sides specified at the time the disk was formatted.

When DOSPLUS II is accessing disk drives, it will continue to use the last known DCT information. You must tell the system manually when it is time to re-read the DCT for a particular disk drive. The only time that this will be needed is when you are switching between double and single headed disks. If a drive in your system previously contained a single headed disk and you insert a double headed disk, DOSPLUS II must be made aware of the fact that the DCT should be re-read the next time that drive is accessed.

All other parameters (Density, Track count, etc.) that are stored in the DCT will become evident to the system when they have changed. On the other hand, changing from single to double headed will not necessarily produce an immediately evident change. That is why you must inform the system manually.

The next piece of information, disk density, is also contained in the DCT and will be picked up automatically by the system. This will be represented in the display line by either an "Den=S" or a "Den=D" (for <S>ingle or <D>ouble density). The capacity to format diskettes single density in the Model II is unique feature of DOSPLUS II.

Those parameters (Cylinders, Sides, and Density) are all set for the disk at the time of format (see the utility program FORMAT). You do not, and can not, adjust those parameters with CONFIG. Their purpose in the display line is to inform you of the current status of your system.

The next parameter in the floppy disk display line is step rate. This indicates how fast, in milliseconds, you wish the disk drive to step between cylinders. Model II Disk drives may be set for a maximum step rate of "1". Model 16 Disk drives may be set for a maximum step rate of "0". The step rate value is a relative figure. Your actual rates are :

Value	Step rate
0	3 Milliseconds
1	5 "
2	10 "
3	20 "

Please be careful that you do not set the Disk drives in the Model II for a step rate faster than that which they are able to handle. This can cause extremely unreliable operation. This parameter will be displayed as "Step=n", where "n" is the currently defined drive step rate.

The next information given in the display line is motor delay. This parameter instructs DOSPLUS II to delay before accessing a disk drive in order to allow the drive's motor to come to speed. This will be shown in the display line as either "MDelay=Y" or "MDelay=N", indicating whether the delay is on or off. On the Model II, this is not needed and this parameter should be set to "N". On the Model 16, this is very definitely needed and this parameter should be set to "Y" for the system to function correctly.

Following that parameter is the head load delay. This parameter will cause DOSPLUS II to delay after accessing a disk drive in order that the read/write head may "load" against the media. This parameter will be displayed as "HLoad=Y" or "HLoad=N", depending as to whether or not you have turned this option on or off. This parameter will be required for all Model II drives, but is NOT required for any Model 16 drives. Certain aftermarket disk drives may also require this parameter set to "Y", consult your drive dealer for exact information.

The last parameter displayed on the line is the skip parameter. Displayed as either "Skip=Y" or "Skip=N", if this parameter is set equal to "Y", DOSPLUS II will "double step" the disk drives (i.e. read every OTHER track). This parameter is intended primarily for use after the five inch drive adapters have been implemented, so that you may read 35/40 track disks in 80 track drives. Currently there is no real practical application of this parameter. If you were to set it by mistake, you could cause a system read error. Leave this option disengaged (i.e. set equal to "N").

Rigid disks -

The first unique parameter (i.e. after the "Cyls" display) on the rigid disk's CONFIG line is the "Fix" or "Rem" parameter. This parameter is used to indicate whether the rigid disk you are using is a fixed platter or a removable platter assembly. You configure this setting via the use of the "FIXED" parameter (i.e. either "Fixed=Y" or "Fixed=N").

At this writing, there were no removable drive systems supported by DOSPLUS II, so DOSPLUS II does not currently use this parameter. At this time, you may alter it, but the only thing that changes is the display line. It does NOT affect system operation.

Next in the rigid drive display line is the head count. It will be shown in the display line as "HCnt=n", where "n" is the number of read/write heads that logical drive is set for. Remember, this parameter will be equal to the number of platters in the drive times two. If you have not done so already, we suggest that you read the section of the operations manual on hard disk theory before attempting to configure your hard disk. Standard Radio Shack 8.4 megabyte hard disks (Cat. # 26-4150/51) have 4 heads. If you are using a different drive, this may change. Consult your drive owner's manual or the dealer from whom you purchased the drive to be sure.

Following the head count will be the drive step setting. This will be shown in the display line as "Step=nnn", where "nnn" is the currently defined step setting for that drive. This is NOT the same thing as a floppy drive's step rate (discussed earlier). This is a one byte value between 0 and 255 (00 and FF hex) that indicates what manner of stepping you have chosen. DOSPLUS II will output that value to the proper port to configure the drive. Standard Radio Shack 8.4 megabyte hard disks (Cat. # 26-4150/51) should have the step option set to "0". To discover the proper setting is for your drive, if it is different, consult (once again) either the drive owner's manual or the dealer from whom you purchased the unit.

The next piece of information shown in the display line is the head offset. This will be displayed as "HOff=n", where "n" is the number of the head you wish this logical drive to start with. Once again, if you have not read the section in the operations manual covering the manner in which DOSPLUS II deals with hard disks, you should read that before attempting this section. Each logical drive may begin with any physical read/write head that you desire. The head offset parameter allows you to set that.

This value will be exact. Since we start numbering the heads with head "0", if you wish to begin this drive with the second head, you skip one head (i.e. a head offset of "1" or start with head number "1").

Following the head offset value will be the cylinder offset value. This will appear in the CONFIG display line as "COff=nnn", where "nnn" is the number of the cylinder you wish this logical drive to start at. This parameter is usually used with the style of hard disk paritioning that uses larger and fewer cylinders; therefore in most cases, it will not be used in conjunction with the head offset parameter.

Your cylinder offset parameter will be used to reflect the point on the hard disk that you wish the next partition to begin at. For example, if you were using a 256 cylinder hard drive with two platters (meaning that you have four surfaces and therefore four heads), and you wanted to partition it into four equal segments, you could do it in the following manner.

(1) Set the head count to four for each rigid logical drive (i.e. HC=4).

(2) Since 256 divided by 4 is 64, you would set each cylinder offset to its proper mulitple of 64. For the first drive, the cylinder offset would, of course, be 0. For the second drive, 64. The third drive, 128 and the fourth drive, 192.

By setting your hard disk up in that manner, you in effect have four 64 cylinder hard disks with four heads per drive instead of one large 256 cylinder unit. This allows you the same amount of total space, but it allows you to maintain four separate directories (one for each drive), with up to 256 files in each. You could have set that up differently if you wished. It is at the user's discretion. You could have made two 128 cylinder drives or one 200 cylinder and one 56 cylinder or any other combination that would meet your needs.

After the cylinder offset parameter is the sectors per track (or track size). This will be expressed as "TS=nn", where "nn" is the number of sectors on each track. This is NOT an abitrary figure. Every hard disk is capable of a certain number of sectors per track and if there is no user's manual that contains that information, then the dealer that sold you the drive should certainly know. Standard Radio Shack Model II/16 8.4 megabyte hard disk units (Cat. # 26-4150/51) are capable of 34 sectors per track.

Please do not confuse this with the number of sectors per cylinder. This figure is for one track only! One track on one side of one surface. DOSPLUS II is intelligent enough to calculate the number of sectors per cylinder from this figure and the head count value.

Altering your CONFIG settings -

Now that we have covered in detail the CONFIG display line and talked a little about each parameter contained within it, we will address the task of altering these settings. We will cover altering the floppy disk drive parameters first and then discuss altering the rigid disk parameters.

The general form for altering the CONFIG parameters is :

```
CONFIG drivespec {param=exp...}
```

Floppy disk parameters

Floppy:

This parameter indicates to DOSPLUS II that the disk being CONFIGured is a floppy disk. Although this parameter will take a switch (i.e. "Y" or "N"), there is no need to use one. "Y" will be assumed when FLOPPY is in the parameter list and "FLOPPY=N" is the same thing as saying "RIGID". Therefore, to set this parameter, simply specify the drive number and include the word "FLOPPY" in the parameter list.

Example :
```
CONFIG :0 {FLOPPY}
CONFIG :0,F
```

WP:

This parameter sets the software write protect for the drive being configured. This allows you to, without having to actually remove and replace the write enable tab on the disk, protect one of your disks from being accidentally written to. It has the same exact effect as removing the write enable tab within the DOSPLUS II system. However, other programs that use their own disk I/O drivers may not recognize this. Set the parameter equal to either "Y" or "N". "Y" will be assumed.

Example :
```
CONFIG :0 {WP=Y}
CONFIG :0 {WP=N}
CONFIG :0,W
```

MD:

This parameter controls the motor on delay for the disk drives. On the Model II, this is not needed and should be left set equal to "N". On the Model 16, this IS needed and should be set to "Y". When included in the parameter list, "Y" will be assumed.

Example :
```
CONFIG :0 {MD=Y}
CONFIG :0 {M}
CONFIG :0,MD=N
```

HL:

This parameter controls the disk drive head load delay. If you set this parameter to "Y", DOSPLUS II will include an extra delay the first time that it accesses a disk drive in order for the read/write head to "load" against the media. This is NOT needed for any standard Radio Shack Model II or 16 drives and should be left set equal to "N". Certain other drive manufacturers may require this delay. Consult your drive dealer to be sure. If mentioned in the parameter line, "Y" will be assumed.

Examples :
```
CONFIG :0 {HL=Y}
CONFIG :0 {HL=N}
CONFIG :0,H
```

Step:

This parameter controls the speed at which DOSPLUS II will attempt to step your disk drives between tracks. This is a relative value as the table below indicates :

Value	Actual step rate (track to track in milliseconds)
0	3 milliseconds
1	6 "
2	10 "
3	15 "

You must set the step parameter equal to one of those values when you include it in a parameter list. There is no default value for such a setting. Standard Model II drives will accept a step rate of "1". Model 16 drives can step at "0".

Example : CONFIG :0 {STEP=1}
CONFIG :0 {S=1}
CONFIG :0,S=1

Skip:

This parameter, when set to "Y", tells DOSPLUS II to "double step" or read every other track. This function is really designed for the time when we will be able to attach five inch drives to the Model II, so that we can read 40 track disks in 80 track drives. It has no practical use with the eight inch drives, almost all of them being 77 track. It should, for standard operation, be left set to "N". If included in the parameter list, "Y" will be assumed.

Example : CONFIG :0 {SKIP=Y}
CONFIG :0 {SKIP=N}
CONFIG :0,SKIP

Size:

This parameter informs DOSPLUS II whether the drive being CONFIGured is a five or eight inch disk drive. At this writing, the hardware interface to enable five inch drives on the Model II/16 was still under development. We have implemented this parameter in the system in preparation for the day when it will be available, but the system currently does nothing when you set a drive as five inch (other than change the display line). This parameter should be left set to "8". If you include this in the parameter list, you must specify either "5" or "8". There is no default value for this setting.

Example : CONFIG :0 {SIZE=8}
CONFIG :0 {SIZE=5}
CONFIG :0,SIZE=8

PD:

This parameter controls which physical drive unit that the drive device being configure will address. If you have four floppy drives in your system, then you have four actual (physical) drives. These are numbered 0, 1, 2, and 3. You may have the drive devices point to these in any order you like.

For instance, let's assume that the first four drive devices (named ":0" through ":3") are all floppy drives. You have the four physical floppy drives mentioned earlier. You decide that you wish to change the order of these drives (i.e. have the external bay drives scanned in the opposite order of normal). To do that, you would need to have drive device ":1" address physical drive 3 and drive device ":3" address physical drive 1. Drive device ":2" will still address physical drive 2 because that is still the middle drive. All you would do is set the "PD" parameter for drive device ":1" to "3" and the "PD" parameter for drive device ":3" to "1".

Remember, you only have four physical floppy drives (0 - 3). Also keep in mind that this parameter has no default value. If you are going to mention "PD" in a parameter list, you must specify a drive number (0 - 3).

Example :

```
CONFIG :0 {PD=0}
CONFIG :0 {PD=3}
CONFIG :0,PD=0
```

Rigid disk parameters

Rigid:

This parameter indicates to DOSPLUS II that the disk being CONFIGured is a rigid disk. Although this parameter will take a switch (i.e. "Y" or "N"), there is no need to use one. "Y" will be assumed when RIGID is in the parameter list and "RIGID=N" is the same thing as saying "FLOPPY". Therefore, to set this parameter, simply specify the drive number and include the word "RIGID" in the parameter list.

Example :

```
CONFIG :4 {RIGID}
CONFIG :4,R
```

Fixed:

This parameter indicates to DOSPLUS II whether or not the rigid disk being CONFIGured is a fixed or removable platter drive. Currently, there are no removable platter rigid drives availible for the Model II/16 which are supported by DOSPLUS II. This parameter is included for the day when these units ARE supported. You will be advised via the addendum sheets that cover any special hard disk drivers that you may receive whether or not this particular driver enables this parameter. To set this parameter to "Y" now only changes the display line. It will NOT affect system operation and should be left as "FIXED=Y" or "FIX" (which is the default setting).

Example :

```
CONFIG :4 {FIXED=Y}
CONFIG :4 {FIXED=N}
CONFIG :4,FIXED
```

WP:

This parameter sets the software write protect for the drive being configured. This allows you to protect against an accidental write to the drive. It has the same exact effect as removing the write enable tab on a floppy disk. With DOSPLUS II, this is the only way to "write-protect" a rigid drive. However, other programs that use there own disk I/O drivers may not recognize this. Set the parameter equal to either "Y" or "N". "Y" will be assumed.

Example :
```
CONFIG :4 {WP=Y}
CONFIG :4 {WP=N}
CONFIG :4,W
```

Step:

This parameter allows you to set a relative step rate value for your rigid drives. This value is used to configure your pariticular hard disk for the manner of stepping desired. This is NOT the same type of step rate value that you configure for floppy drives. On the floppy drives, you set a value and DOSPLUS II interpreted it and stepped the drives accordingly. For this parameter, you enter a value and whatever value you enter DOSPLUS II will output to the hard disk when telling it how to step. How the drive reacts to this is a function of the disk drive controller.

There is no set value for this parameter. Each drive must have its correct settings. This information should be in the drive owner's manual and if not there, then the dealer who sold you the drive should know. For standard Radio Shack 8.4 megabyte Model II/16 hard drives (Cat. # 26-4150/51), the proper step setting is "0". There is no default rate for this. If you include it in the parameter list, you must specify a value.

Example :
```
CONFIG :4 {STEP=0}
CONFIG :4 {STEP=3}
CONFIG :4,S=0
```

HO:

This parameter sets the drive head offset. This informs DOSPLUS II how many heads to skip when accessing a particular logical drive. If you choose to configure your hard disk with each surface as a separate drive, this parameter is used. Each logical drive would have a head count (HC) of "1" and the head offset (HO) would tell the system which head.

For example, a two platter drive has four heads. If each surface (or head) was set up as a separate drive, then the head count for each would be "1" and the head offset would start at "0" and go up one for each logical drive. Therefore, the drive that you desire to be first would have a head offset of "0" (meaning not to skip over any heads) to cause it to begin with the first head. The second drive would have a head offset of "1" (meaning skip one head) to cause it to begin with the second head. This would continue until all logical drives had been assigned their own head.

Remember that you may never have a head offset greater than the head count of the drive minus one. In other words, you cannot specify a head offset value that would cause the system to begin a logical drive at a non-existent head. Also remember that there is no default setting for this parameter either. If you specify the head offset parameter in the list, you must assign it a value.

Example :

```
CONFIG :4 {HO=0}
CONFIG :5 {HO=1}
CONFIG :4,HO=0
```

CO:

This parameter allows you to set a rigid drive's cylinder offset. It, like the head offset, is used when you are partitioning the drive. However, this parameter is usually used with the style of hard disk partitioning that uses larger and fewer cylinders; therefore in most cases, it will not be used in conjunction with the head offset parameter.

Your cylinder offset parameter will be used to reflect the point on the hard disk where you wish to begin the next partition. For example, if you were using a 256 cylinder hard drive with two platters (meaning that you have four surfaces and therefore four heads), and you wanted to partition it into four equal segments, you could do it something like this. First, set the head count to four for each rigid logical drive (i.e. HC=4). Then, since 256 divided by 4 is 64, you would set each cylinder offset to its proper mulitple of 64. For the first drive, the cylinder offset would, of course, be 0. For the second drive, 64. The third drive, 128 and the fourth drive, 192.

By setting your hard disk up in that manner, you in effect have four 64 cylinder hard disks with four heads per drive instead of one large 256 cylinder unit. This allows you the same amount of total space, but it allows you to maintain four separate directories (one for each drive), with up to 256 files in each. You could have set that up differently if you wished. It is at the user's discretion. You could have made two 128 cylinder drives or one 200 cylinder and one 56 cylinder or any other combination that would meet your needs.

Please note that you may not configure a drive with a greater cylinder offset than it has cylinders (minus one). In other words, you may not assing a cylinder offset that would begin a logical drive at a non-existant cylinder. Also, use extreme care when dividing up a hard disk that your partitions do not overlap cylinders. This will cause extreme system errors and potentially great losses of data.

Example :

```
CONFIG :4 {CO=0}
CONFIG :5 {CO=64}
CONFIG :4,CO=0
```

HC:

This parameter is used to set the head count for a rigid drive. Every rigid drive is made up of platters. Each platter has two surfaces and each surface has its own read/write head. Therefore, to determine your drive's head count, multiply the number of platters times two. Standard Radio Shack 8.4 megabyte Model II/16 hard disk units (Cat. # 26-4150/51) have two platters or four heads.

If you are configuring with method one (smaller and more numerous cylinders), you will use each surface as a separate drive and each partition will have a head count of "1". If you are configuring with method two (larger and fewer cylinders), you will use all surfaces in each partition and split the drive using the cylinder offset parameter. For a standard Radio Shack hard disk, this would leave each partition with a head count of "4".

Example :

```
CONFIG :4 {HC=4}
CONFIG :5 {HC=1}
CONFIG :4,HC=4
```

TS:

This parameter allows you to instruct DOSPLUS II regarding the track size of the rigid drive. This value will be set to the number of sectors that are on each track. This is NOT an arbitrary figure. You must find out what the setting should be for the drive that you have and set this parameter accordingly. If the drive owner's manual does not have this information, then consult the dealer who sold you the unit. Standard Radio Shack 8.4 megabyte Model II/16 hard disk units (Cat. # 26-4150/51) have 34 sectors per track, so "TS" should be set to "34".

Remember, this is NOT the number of sectors per cylinder. This is CONFIGuring for one track on one side of one surface. DOSPLUS II will automatically calculate the number of sectors per cylinder from this parameter and the head count value.

Example :

```
CONFIG :4 {TS=34}
CONFIG :5 {TS=33}
CONFIG :4,TS=34
```

PD:

This parameter controls which physical drive unit that the drive device being configured will address. If you have two rigid drives in your system, then you have two actual (physical) drives. These are numbered 0 and 1. You may have the drive devices point to these in any order you like.

For instance, let's assume that you have a standard Radio Shack 8 megabyte hard disk unit. When connected to the computer, that would be physical rigid drive "0". If you were to purchase an additional hard disk unit and attach it also, that would be functioning as physical rigid drive "1". The PD parameter simply defines to which actual hard disk unit the settings in that CONFIGuration line are referring.

If you only had the one Radio Shack drive, you could point one logical drive to it or eight, it makes no difference. For each logical drive that you want addressing that first physical drive, you would set the PD parameter for that drive to "0". You would use the other hard disk parameters to make certain that two logical drives do not occupy the same area of the disk.

Example :

```
CONFIG :4 {PD=0}
CONFIG :5 {PD=0}
CONFIG :4,PD=0
```

Some recommended settings -

Radio Shack Model II floppy disk drives:

For these drives, the only setting that would be changed from the way that DOSPLUS II is shipped is "Step=1". For all drives in your system, set this parameter.

Radio Shack Model 16 floppy disk drives:

For these drives, there are TWO parameters that need to be altered from the default (Master disk) settings. The first is "MD=Y" and the second "Step=0". For all floppy drives in your system, those two parameters should be set in the prescribed manner.

When you first power-up your system, you may type "MOD16" and press ENTER if you like. There is a pre-made configuration file stored on the disk for you that will configure your system as a two drive Model 16, setting the "MD" and "Step" parameters. This will give you an example of what all floppy drive configurations should look like.

Radio Shack Model II or 16 8.4 megabyte hard disk drive:

There are two 2 ways that we recommend you set up your hard disk : (1) as two logical drives and (2) as four logical drives.

First, the two logical drives. Set the following parameters to the recommended values :

Drive 4	Drive 5
Rigid	Rigid
Fixed=Y	Fixed=Y
HC=4	HC=4
Step=0	Step=0
HO=0	HO=0
CO=0	CO=128
TS=34	TS=34

Example :

```
CONFIG :4 {R,FIXED=Y,HC=4,S=0,HO=0,CO=0,TS=34}
CONFIG :5 {R,FIXED=Y,HC=4,S=0,HO=0,CO=128,TS=34}
```

Next, the four logical drives. Set the following parameters to the recommended values :

Drive 4	Drive 5	Drive 6	Drive 7
Rigid	Rigid	Rigid	Rigid
Fixed=Y	Fixed=Y	Fixed=Y	Fixed=Y
HC=4	HC=4	HC=4	HC=4
Step=0	Step=0	Step=0	Step=0
HO=0	HO=0	HO=0	HO=0
CO=0	CO=64	CO=128	CO=192
TS=34	TS=34	TS=34	TS=34

Example :

```
CONFIG :4 {R,FIXED=Y,HC=4,S=0,HO=0,CO=0,TS=34}
CONFIG :5 {R,FIXED=Y,HC=4,S=0,HO=0,CO=64,TS=34}
CONFIG :6 {R,FIXED=Y,HC=4,S=0,HO=0,CO=128,TS=34}
CONFIG :7 {R,FIXED=Y,HC=4,S=0,HO=0,CO=192,TS=34}
```

If you have divided the hard disk into two logical drives, then when you RFORMAT each of the logical drives, you will specify "128" when it asks you for cylinder count. If you have divided it up as four logical drives, however, the proper answer to the cylinder count question is "64". In both instances, the answer to the query regarding the number of surfaces will be 4.

The reason for this is simple, Radio Shack's 8.4 megabyte hard disk has 256 tracks per surface. If you are dividing that into two even parts, you have 128 cylinders per division. If you are dividing it into four even parts, you have 64 cylinders per division.

As you become more experienced and used to the methods of "partitioning" hard disks into smaller logical drives, you will be able to do more and inventive things with your hard disk. If, as you are configuring the rigid parameters, you notice that one of the settings in our recommended table happens to match what is already there, don't worry. For the rigid recommendations, we gave you all information as opposed to the floppies where we told you only what needed to be changed.

Some potential problems using CONFIG -

(I) My hard disk seems to have unexplained "crashes" after operating fine for some time. These seem to be random (occurring almost at will) and completely destroy one of my partitions.

(A) This sounds like "drive overlap". On the hard disk, you have sectioned off various parts of the hard disk as being separate drives (the "logical" drives discussed earlier). If you, by some oversight, have set the DOS such that two logical drives "overlap" or contain the same area of the hard disk, the data can be corrupted.

The most common form of this is setting CONFIG for one size of logical drive and RFORMATing another. For example, if you set up you CONFIG statements for two 128 cylinder drives and then formatted for four 64 cylinder drives, this would occur.

The partition on the disk would be 64 cylinders and formatted as such, but the CONFIG line indicates that this drive has 128 cylinders. This WILL cause problems. First and foremost, when it seeks the second partition, the head offset ("HO") parameter will be wrong and the DOS will not be able to find the drive. Instead, when it looks for the second partition at cylinder 128, it will find the third.

(2) My floppy disk drive has all sorts of disk read errors, re-tries all the time, and I can't seem to load files from it.

(A) This problem sounds like the step rate is set too fast. If you exceed the step rate that the disk drive is capable of, then the unit will fail to respond in the manner described above. If this is happening to you, slow down the step rate.

(3) My floppy disk refuses to read altogether and is making odd noises when the head moves.

(A) This would seem to indicate that the "Skip" parameter has been set for that drive. Remove it and all should be well.

(4) I can't seem to get my system up and running on the hard disk.

(A) Read the recommended settings for the hard disk that are discussed above. Decide which to use and set your drive accordingly. Then consult RFORMAT to learn how to intialize the drive to receive data. Once you have formatted the drive, consult SYSGEN to learn how to install the system on it.

If you are still having problems past all these, please get in touch with MicroPower Technical Support team at the telephone numbers published in the "acknowledgements" portion of this manual (immediately following the table of contents).

(5) I've changed my configuration several times, but every time I reset the machine, what I have done is lost and I must start over.

(A) This is a misunderstanding of the way CONFIG works. CONFIG only changes the parameters IN MEMORY! It alters how the system is running at the moment that it is invoked. To make permanent changes, you must save the current system configuration with the SYSTEM command (see the library command SYSTEM). You will create a disk file that contains your configuration. Then whenever you re-load this file (by simply executing it), the configuration will be set to EXACTLY how it was when you saved the file. If you wish, you may put this filename on an AUTO statement.

COPY

This command allows you to copy data from one point in the system to another. It operates in several modes and will copy either a byte at a time or an entire file at a time, depending on which mode it is in. By using the wildmask feature, you may copy all or selected files from one drive to another with a single command line. Whenever you copy to or from a file, that file's MOD flag will be reset. Also, all file attributes will be cloned. That is, whatever protection or attributes the source file had will be transferred to the destination file.

==========

The command syntax is:

1. COPY [FROM] channel [TO] channel
2. COPY [FROM] filespec [TO] filespec {param=exp...}
3. COPY [FROM] filespec [TO] drivespec {param=exp...}
4. COPY [FROM] drivespec [TO] drivespec [USING] wildmask {param=exp..}

Mode 1 -

This mode is used to copy from I/O channel to I/O channel. It is here that you may copy one device to another. This mode functions one byte at a time. An I/O channel may be either a devicespec or a filespec, but it may not be a wildmask or drivespec. This mode of COPY has no valid parameters. Any included from the list below will be ignored.

Mode 2 -

This mode is used to copy one file from one drive to another when both the source and destination filespecs are specified. You would use this mode when you are copying a file and you wish to rename that file as you copy it. For example, "COPY TEST:0 TEST1:1" would change the name of the file as it copied it. This is a file orientated copy and devices may not be used.

Mode 3 -

This is really little more than a shorthand version of Mode 2. Only in this mode, you may not rename the file as you are copying it. The destination filespec is assumed to be the same as the source filespec. Therefore, you do not need to re-specify the destination filespec. Only the drivespec will be required. Because you cannot rename the file, this mode must be copying between two separate disks, either in two different drives or single drive between two disks.

Mode 4 -

This mode is used to move one OR several files between two drives. It is a file by file copy using a wildmask to affect only a selected class of files. If you do NOT specify the wildmask (i.e. COPY :0 :1), DOSPLUS II will assume that you wish to copy all files from one drive to another and will default to the "!" wildmask (see the section File and Device specifications).

Your parameters are :

DPW=string	Destination password. When using a wildmask copy, you may occasionally attempt to over-write a protected file on the destination disk. To be allowed access to these files, you must specify the destination disk's Disk Master Password. If this is NOT given, COPY will not allow you to copy over protected files. This only applies on wildmask copies. If you are specifying the filespecs, you may include the destination password with it.
ECHO=switch	Display filenames as they are copied. When using a wildmask copy to move several files from one drive to another, this parameter will allow you to see the filenames as they are copied. If you do NOT specify ECHO, or you specify "ECHO=N", COPY will not display the filenames as it copies the files. If you specify both the QUERY and ECHO parameters, QUERY will override ECHO (you will not see the filename twice). This applies only to a wildmask copy.
INV=switch	Copy invisible files. When doing a wildmask copy, normally only visible files will be copied. If you wish COPY to move invisible files also, you must specify this parameter. This applies only to a wildmask copy.
KILL=switch	Delete source file. If you specify this parameter, after COPY has copied the file to the destination drive, it will kill it on the source drive. Use caution when combining this parameter with a wildmask copy, as you may delete more files than you mean to. This parameter applies to all forms of file orientated copies.

MOD=switch

Copy by MOD flag. This parameter allows you to control COPY based on whether or not the MOD flag is set. If you specify "MOD" or "MOD=Y", then only those files that have the MOD flag set will be copied. If you specify "MOD=N", then only those files that do NOT have the MOD flag set will be copied. This applies only to wildmask copies.

OVER=switch

Prompt for overwrite. When doing a wildmask copy, if COPY encounters the same filespec on the destination drive, it will attempt to overwrite the file. If you specify this parameter, though, COPY will prompt you first before attempting to overwrite. If you answer "N", then COPY will skip that file and proceed to the next. This is valuable in wildmask copies to assure that you do not overwrite files by accident. This parameter applies only to file copies.

PROMPT=switch

Prompt for disks. This parameter allows you to do a "single drive copy". If specified, COPY will prompt you for the source, destination, and system disks as needed. If the file exists on the destination disk, COPY will inform you and ask if you wish to overwrite it. This parameter only applies to NON-wildmask file orientated copies.

QUERY=switch

Prompt for copy. If you specify this parameter when doing a wildmask copy, COPY will prompt you for each file before copying it. If you respond "N", then COPY will pass over that file and proceed to the next. This is useful when you have many similar filespecs on a drive and you wish to copy some, but not all, of them. This applies only to file copies.

SPW=string	Source password. When doing a wildmask copy, you may occasionally attempt to copy a protected file from the source drive. To be allowed access to these files, you must specify the source disk's Disk Master Password. If this is NOT given, COPY will not allow you to copy those protected files. This applies only to wildmask copies. If you are specifying the filespecs, you may include the source password with it.
TINY=switch	Copy with tiny buffer. Normally COPY will use all available memory when copying a file. It will read as much as it can of a file before writing it out. This greatly increases the speed and efficiency of the copy, especially with rigid disks. There may be certain times that you do not want this method to be used, however. By specifying the "TINY" parameter, you will cause COPY to use a much smaller area of memory and copy a file sector by sector. This will slow down the copy, but it will prevent COPY from using memory outside of the system overlay area and potentially corrupting important data. This parameter applies to all forms of file orientated COPY.

Default values. The value for any parameter that carries a switch, if included in the parameter field without a switch is assumed to be "Y". The default values given here are those that are assumed for each parameter when no mention of the parameter is made.

DPW	No password set.
ECHO	No. Do not display filenames.
INV	No. Copy only visible files.
KILL	No. Do not delete source file.
MOD	No default.
OVER	No. Overwrite without asking.
PROMPT	No. Copy assuming system installed.
QUERY	No. Copy without asking.
SPW	No password set.
TINY	No. Use all available memory.

Abbreviations :

DPW	D
ECHO	E
INV	I
KILL	K
MOD	M
OVER	O
PROMPT	P
QUERY	Q
SPW	None. Must be spelled out.
TINY	T

==========

The average user will find themselves using COPY more than almost any other command in DOSPLUS II. This command operates in three ways with two different styles of data.

(1) One channel to another copying a byte at a time.

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

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

In the command syntax box, we described the four "modes" of COPY. The first mode corresponds with method (1) above. The second and third modes employ method (2). The fourth mode uses method (3). Let's look at each of these.

Mode 1 -

In this mode we are copying a :

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

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

An example of copying a device to a file would be copying the keyboard ("@KI") to a diskfile ("FILENAME/EXT"). The format would be "COPY @KI TEXT:1". Any output from the keyboard (i.e. characters that you type...) would be sent to the disk file "TEXT" on Drive 1.

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

An example of copying a file to a device would be copying a text file ("FILENAME/EXT") to the comm line ("@CA" or "@CB"). This would allow you to send data directly from a disk file out the commline. The format would be "COPY TEXT:1 @CB". This would instruct the DOS to copy the file "TEXT" located on disk drive ":1" out the first commline.

When you are using a channel to channel copy, you may abort the copy by pressing the BREAK key. The copy will terminate if a Control-C is received from the sending device.

Technical note : This does not apply if no characters have been received yet from the sending device. DOSPLUS II will, in effect, "wait" at the sending device before checking the keyboard again. In order to abort, at least one character needs to have been copied.

Modes 2 and 3 -

In these modes, we are copying a :

* File to a file.
* File to a drive.

Therefore, if a copy involves a device or more than one file, it is not functioning in these modes.

An example of copying a file to a file would be if you copied the file "TEST1" from Drive 0 to Drive 1. The format would be "COPY TEST1:0 TEST1:1" or "COPY TEST1:0 :1". Notice the two different manners of addressing that. Those are the two forms that make the difference between these modes.

In the first form, where the second filespec IS specified, you have the option of changing the filespec as you copy it. For example, "COPY TEST1:0 TEST2:1" would be perfectly legal. When the file "TEST2" on Drive 1 was examined, you would find that is it the same as the file "TEST1" on Drive 0.

Using the second form, where the second filespec is NOT specified, you may not rename the file while you are copying it. For example, "COPY TEST1:0 :1" is going to create a file "TEST1" on Drive 1. Since copying without changing the filespec is a far more common occurrence than copying with the change, you will be using this form a great deal.

Remember that when you are using either of these forms you are engaging a file by file copy. This uses the "big buffer" (all available RAM) for the copy. This greatly increases the speed and efficiency of the copy, especially between two volumes of the hard disk, but can corrupt programs in memory.

To preserve these programs, you must specify the tiny buffer option and force COPY to copy only one sector at a time. This will slow down the copy, but it will force COPY to keep its buffer within the system overlay areas and out of user memory altogether.

If you are going to copy a file into a different area of the same disk, you MUST change the filename. Therefore, the second form is only legal when moving files between two disks. This can be within a single drive is you are using the mount parameter.

Mode 4 -

In this mode we are copying between a :

* Drive and a drive.

Therefore, if your copy involves a device or only a single file, you should be using one of the other forms.

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

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

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

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

Please bear in mind that if a file is invisible it will NOT be copied unless the "Inv" parameter has been specified. This will become very important when setting up non-standard system disks with SYSGEN and COPY.

The filename will NOT be displayed during a copy unless you ask for an "Echo". Under many circumstances, you will want to see what files COPY is moving, so you will want to use this parameter. The "Query" and "Over" parameters become important, also.

If you use the "Query" parameter, DOSPLUS II will ask you if you wish to copy each file BEFORE it actually copies it. If you use the "Over" parameter, DOSPLUS II will ask if you wish to overwrite a file (when it finds the same filespec on the destination drive) BEFORE it actually overwrites it.

Examples:

```
COPY FROM :0 TO :1 USING ! {INV,ECHO,OVER,SPW='PASS'}
COPY :0 :1 ! {INV,ECHO,OVER,SPW='PASS'}
COPY :0 :1 ! {I,E,O,SPW='PASS'}
COPY !:0 :1,I,E,O,SPW='PASS'
```

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

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

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

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

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

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

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

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

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

Finally:

It was stated earlier, but reinforce the fact in your mind that you MUST copy FROM an input device TO an output device. To do otherwise is a fatal error.

Refer to the Operations Section under "File and Device specifications" for a complete list of devices and their classes. Remember that devices such as files and the commline can be either input OR output. Therefore, they will appear in either position.

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

CREATE

This command allows you to create disk files and pre-allocate their space. You also have the options of specifying how large to make the file, the logical record length, etc. You may have DOSPLUS II clear out the file space and specify the data to be used. If you wish, CREATE will verify the file space also. You may also set the KEEP flag when creating a file. This command is most often used in creating and pre-allocating data files for applications programs, but has many other implementations as well.

==========

The command syntax is :

CREATE filespec {param=exp...}

"filespec" is the standard DOSPLUS II file specification that informs CREATE what the name of the file you wish to create is. You may also specify the drive on which to place the file. If the file already exists, CREATE will abort with an error.

"param" is the optional parameter indicating what further action you might wish CREATE to take past simply creating a directory entry. This would include pre-allocating the file, filling it with data, etc.

"exp" is the optional expression that modifies the action of the parameters. This may be a switch or a value, depending on which parameter we are using.

Your parameters are :

DATA=value — Fill data. If you specify this parameter, CREATE will fill the file space with the specified data after creating it. This parameter is only valid if you have indicated to CREATE to pre-allocate space for the file. This may be expressed in decimal or hex form (append an "H" to hex input). Values may be one or two bytes in length.

GRANS=value

Number of grans. This parameter allows you to pre-allocate a file for a specified number of granules. Granules are defined as being the "minimum unit of disk allocation". A table of disk formats and granule sizes is located in the technical section of this manual. The number of free granules on a disk may be discovered by using the "DIRCHECK" utility.

KEEP=switch

Set keep flag. This parameter allows you to set the "KEEP" flag for a file after you create it. When a file has this flag set, the system will never de-allocate that file's disk space. This flag may also be set or removed via the ATTRIB command (see the library command ATTRIB).

KILO=value

Number of kilobytes. This parameter allows you to pre-allocate a file for a specified number of kilobytes. Specify the number of total kilobytes that you wish the file to take up on the disk and this CREATE will allocate that much space to it. Be certain to specify the number realizing that this value is assumed to be kilobytes. In other words, to specify 100 kilobytes use "KILO=100" and not "KILO=100000". Also, it must be an integer value.

LRL=value

Logical record length. This will allow you to specify the logical record length of the file you are creating. This will be of great importance in creating data files to use from BASIC, because BASIC will not let you open a file with a different logical record length than it was created with. You may use any logical record length between 1 and 256, inclusive.

SIZE=value	Number of records. This parameter allows you to pre-allocate a file for a specified number of logical records. The amount of actual space used for the file will be dependant on the file's logical record length.
VERIFY=switch	Verify disk space. This parameter instructs CREATE to read each sector of a file's disk space after creating it. It is only valid if you have both created and pre-allocated a file and is usually used in conjunction with the "DATA" parameter (although it doesn't have to be). It will report any encountered errors. This parameter is useful in detecting flawed areas of the disk BEFORE important information is stored there.

Default values. The value for any parameter that carries a switch, if included in the parameter field without a switch is assumed to be "Y". The default values given here are those that are assumed for each parameter when no mention of the parameter is made.

DATA	No default. If included, data MUST be specified.
GRANS	0. No space allocated.
KEEP	No. Allocate/de-allocate dynamically.
KILO	0. No space allocated.
LRL	256.
SIZE	0. No space allocated.
VERFIY	No. Do not verify file's sectors.

Abbreviations :

DATA	D
GRANS	G
KEEP	K
KILO	None. Must be spelled out.
LRL	L
SIZE	S
VERIFY	V

==========

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

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

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

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

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

Using CREATE to pre-allocate data files can greatly increase the speed of data handling. Because the file already exists and has its space allocated, time will not have to be taken to do it dynamically. Also, depending on the disk, the file will tend to be less segmented. The fewer segments the file is in, the less that the drive has to move the head around when reading in the data.

Pre-allocation -

To determine the size of the file when you wish to pre-allocate, you have three options. You may : (1) Allocate by the number of records, (2) allocate by the number of granules, or (3) allocate by the number of total kilobytes.

When allocating by number of records, then the logical record length has a great bearing on file size. The logical record length of a file in DOSPLUS II does little more than affect how the directory entry will look. This is at system level. However, from BASIC, this information becomes VERY important.

You can choose any logical record length between 1 and 256, inclusive. This, as just stated, will have no effect on machine language programs under DOSPLUS II. The only place the logical record length is vital to accessing the file is when you are using BASIC. One of the restrictions placed on us by the Disk BASIC in the Model II is that you may NOT open a data file for random access unless the logical record length of the file is specified correctly when the file is opened. You may not open a sequential data file for access at all with anything other than a logical record length of 1.

Therefore, when creating a file for use by the DOSPLUS II system itself, you may simply use the logical record length of 256 (unless for some reason the accurate display in the directory is desired). However, when creating data files for BASIC, be certain to determine the logical record length in advance of creating the file so that you may set it correctly.

You will adjust the logical record length of the files you create with the "LRL" parameter (see the command box above). When pre-allocating a file by records, you specify the number of records you desire in that file. This is accomplished by using the "SIZE" parameter. Simply set Size equal to however many records you anticipate. The actual physical size of the disk file will be equal to the number of records specified times the logical record length used. Of course, if the logical record length is 256, then SIZE will equal the number of sectors.

Allocating a file by granule assumes that you have at least a passing familiarity with what a "granule" is. A granule is defined as the smallest unit of disk allocation. A disk is made up of sectors. Each sector is 256 bytes long (512 bytes on a Radio Shack 8.4 megabyte hard disk). These sectors are grouped into tracks. The tracks are concentric circles of data on the disk. Each track has a pre-defined number of sectors on it. As data is written to the disk, space must be provided for this.

If DOSPLUS II were to allocate space to a file one sector at a time, the result would be very slow. The drive would constantly be stepping out to the directory track to ascertain where the next free sector was and assign it to the file you are writing to. Therefore, DOSPLUS II will allocate space several sectors at a time. This unit of allocation is called a "granule". On a standard 8 inch single sided floppy disk, a granule is made up of 5 sectors. There are a total of six granules (or 30 sectors) on each track.

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

Allocating a file by kilobytes is simple. Simply figure out how big the file should be and intruct the system. This figure is expressed in kilobytes, so be careful not to ask for more than you desire. For example, "Kilo=100" is asking for 100,000 bytes, not 100. Again, it will not matter what the logical record length is. CREATE will allocate as many records as it needs to to come up to the specifed amount of total disk space. As shown in the example, this value will be adjusted via the "Kilo" parameter.

All of the values that we have just talked about (Size, Grans, and Kilo) MUST be entered in the command line as positive integers (NO fractions will be accepted).

Examples:

```
CREATE NEWDAT:0 {LRL=128,SIZE=100}
CREATE NEWDAT:0 {L=128,S=100}
CREATE NEWDAT:0,L=128,S=100
```

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

```
CREATE PAYROLL:B {DATA=229,GRANS=12,VERIFY}
CREATE PAYROLL:B {D=229,G=12,V}
CREATE PAYROLL:B,D=229,G=12,V
```

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

```
CREATE DATAFILE/DAT {DATA=108,KILO=100,KEEP}
CREATE DATAFILE/DAT {D=108,KILO=100,K}
CREATE DATAFILE/DAT,D=108,KILO=100,K
```

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

```
CREATE BADFILE {DATA=108}
```

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

```
CREATE WORSEFIL {VERIFY}
```

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

Finally:

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

If you do a CREATE with the "Data" and "Verify" parameters, you may be assured that your file's disk space is safe for use.

Bear in mind that BASIC can NOT open a data file with a different logical record length than that with was specified when the file was created and is now stored in the directory entry for that file. Always be careful to specify the correct logical record length the first time.

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

DATE

This parameter allows you to display the currently set system date and, if you wish, to change it. The "Date :" prompt on boot-up will set the date for you, but if you have disabled that question with SYSTEM or wish to change the date without re-booting, this is the method to use.

==========

The command syntax is :

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

The command "DATE" by itself will cause DOSPLUS II to display the currently set system date.

==========

When using the DATE command to set the system date, the date can be specified in a variety of ways. Allowable separators are colons (":"), commas (","), dashes ("-"), slashes ("/"), periods ("."), and spaces. This flexibility allows you to specify the date in whatever format is most comfortable to you.

Also, DATE will accept either a one, two, or four digit year value when accepting a date. This allows you to enter the date as you are used to with TRSDOS and later as you become more familiar with DOSPLUS II, move to the more convenient two digit format.

Examples:

```
DATE 9:13:82
DATE 09:13:82
DATE 09:13:1982
DATE 9-13-82
DATE 9 13 82
DATE 9.13.82
DATE 09/13/82
DATE 9,13-82
```

All of these commands are equivalent and will have the same exact effect. They will set the system date to September 13th, 1982.

```
DATE 9
```

This would set the system date to September 1st, 1980. If the day and year are not specified, DATE will set them to the lowest possible value.

```
DATE
```

This will display the current system date.

Finally:

The date in DOSPLUS II uses an offset of 80 in the years column. That is, the year cannot go below 1980. It also may not exceed 1987. Any of you who are still using DOSPLUS II in 1987 can receive a free patch to allow the offset to handle the next 7 years.

This means that you can take the shortand for the year one step further and express it as 0-7 (for 1980-1987). Hence what was :

DATE 9 13 82

can become :

DATE 9 13 2

The DOSPLUS II date display format is :

day of the week - month day, year - day of year

such that October 8th, 1982 would be displayed as :

Fri - Oct 08, 1982 - 281

DEBUG

DEBUG is DOSPLUS II's built-in memory monitor to facilitate the debugging of your machine language programs. DEBUG, when turned on, is a "ghost". That is, is does not engage until one of two items happens : (1) you engage it manually by pressing the BREAK key, or (2) you attempt to execute a non-protected program file, in which case you are sent into DEBUG immediately after the program is loaded.

==========

The command syntax is :

DEBUG [switch]

"switch" is the optional "on" or "off" condition. If you do not specify this (i.e. type DEBUG by itself), "on" will be assumed.

==========

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

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

1. Pressing <Break> at any time.

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

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

Command (cont.)	Operation performed
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  = B6A0 S-1-----
BC  = 04B1:  C9 7D E6 C0 6F C9 DD 7E  07 B7 79 20 CD D6 C0 28
DE  = 0604:  FE 19 28 39 FE 0A C0 D1  77 78 B7 28 CF 7E 23 CD
HL  = 403C:  04 C3 FA 35 C3 FA 35 C3  FA 35 C3 30 45 C3 27 47
AF' = 1414 ---H-P--
BC' = 00E4:  70 20 CE 2B 11 14 45 DF  DA 7A 19 11 CE FF 22 B1
DE' = 0F0D:  F5 E4 3E 09 F1 EC 4D 0E  F1 3D C9 D5 E5 F5 E7 F5
HL' = 5151:  FE C0 38 02 3E 2E CD 33  00 23 AF C4 CF 51 10 DE
IX  = 4015:  01 64 4C 00 01 07 00 FF  07 73 04 2F 3E 20 8C 00
IY  = 4C46:  06 03 12 12 12 06 05 1E  1E 1E 05 04 14 0A 0A 08
SP  = 41CD:  00 34 40 38 94 06 15 40  AB 42 00 43 3F 3F 4C 00
PC  = 3000:  C3 5E 32 C3 9B 32 C3 74  32 C3 DA 32 C3 C0 31 C3
      4000:  C3 96 1C C3 78 1D C3 90  1C C3 D9 25 C3 57 4B C3
      4010:  0D 44 C3 D7 44 01 64 4C  00 01 07 00 FF 07 73 04
      4020:  8D 3F 20 8C 00 06 8E 4C  42 00 42 00 00 C3 00 44
      4030:  C3 0D 44 C3 82 44 00 00  00 00 00 00 04 C3 FA 35
```

The general format is :

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

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

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

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

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

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

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

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

```
4000:  C3 96 1C C3 78 1D C3 90  1C C3 D9 25 C3 57 4B C3
4010:  0D 44 C3 D7 44 01 64 4C  00 01 07 00 FF 07 73 04
4020:  8D 3C 20 8C 00 06 8E 4C  42 30 42 00 00 C3 00 44
4030:  C3 0D 44 C3 82 44 00 01  00 00 00 00 00 C3 FA 35
4040:  C3 FA 35 C3 FA 35 C3 30  45 C3 27 47 FF FF FF FF
4050:  FF FF FF FF FF FF FF FF  FF FF FF 91 35 91 35 F1
4060:  35 B4 DF 1D 40 00 3C 00  20 1F 02 91 35 F1 35 F3
4070:  00 C6 43 00 51 0C 0F 9A  43 79 43 59 43 FF FF FF
4080:  D6 00 6F 7C DE 00 67 78  DE 00 47 3E 00 C9 4A 1E
4090:  40 E6 4D DB 00 C9 D3 00  C9 00 00 00 00 40 30 02
40A0:  16 FC FF FF 70 71 00 E4  61 FF 00 00 00 00 00 03
40B0:  00 FE FF B5 40 06 29 19  01 75 73 00 00 00 00 00
40C0:  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
40D0:  FF FF FF 06 29 19 F5 FF  00 72 60 EA 00 00 FF 27
40E0:  75 00 FF FF FF FF E1 61  03 FC 46 00 9C EA 80 72
40F0:  00 00 00 E6 72 E6 00 D1  72 1E 75 3A 75 63 75 49
```

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) :

```
5700:  T h i s   i s   a n   e x a m p
5710:  l e   o f   a n   A S C I I   d
5720:  i s p l a y   m o d e ! . . . .
5730:  . . . . . . . . . . . . . . . .
5740:  . . . . . . . . . . . . . . . .
5750:  . . . . . . . . . . . . . . . .
5760:  . . . . . . . . . . . . . . . .
5770:  . . . . . . . . . . . . . . . .
5780:  . . . . . . . . . . . . . . . .
5790:  . . . . . . . . . . . . . . . .
57A0:  . . . . . . . . . . . . . . . .
57B0:  . . . . . . . . . . . . . . . .
57C0:  . . . . . . . . . . . . . . . .
57D0:  . . . . . . . . . . . . . . . .
57E0:  . . . . . . . . . . . . . . . .
57F0:  . . . . . . . . . . . . . . . .
```

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

DIR

This command will display a disk's "directory". A directory differs from a file catalog in that it contains a great deal of information about each file while the file catalog contains ONLY the filename and extension. You also have the option of specifying a wildmask so that only filespecs that match certain criteria are displayed.

==========

The command syntax is :

DIR [FROM] drivespec [TO] channel [USING] wildmask {param=exp...}

"drivespec" is the name of the drive that you wish the directory of. If it is not specified, DIR will globally display the directories of all drives.

"channel" is the optional output channel. That is, where you want the directory to be sent. If it is not specified, the screen (i.e. TO @DO) is assumed.

"wildmask" is the optional wildmask to restrict DIR to a certain group or class of files. If it is not specified, all files will be displayed (i.e. "!" is assumed).

"{param=exp...}" is the optional action parameter that indicates what type of directory you want to see. If no parameters are given, the default values listed below will be in effect.

The parameters are :

SYS=switch	Directory will contain system files as well as standard entries.
INV=switch	Directory will contain both visible and invisible user files.
KILL=switch	Directory will contain names of any deleted files not yet wiped from the directory or over-written by an active file.
ALPHA=switch	Directory will be displayed in alphabetical order.

Default values. If any of these switches are specified in a command line without an expression, DOSPLUS II will assume "Y" and act accordingly. The default values listed here are those that are in effect when the parameter is NOT present in the command line.

SYS	No. User files only will be displayed.
INV	No. Visible files only will be displayed.
KILL	No. Active files only will be displayed.
ALPHA	No. File listing will be in the order that the filenames actually appear in the directory itself.

The FROM, TO, and USING delimiters may be omitted unless you wish to specify the various portions of the I/O field in a non-standard order. The parameters may be abbreviated :

SYS=S, INV=I, KILL=K, and ALPHA=A

==========

This command will display all available information regarding a disk file. It will detail the filename and extension, it will indicate how large the file is, whether the file is segmented or not, whether or not the file has password protection, and what level this protection is set at. It will give you a file's logical record length, it will tell you if the file has been modified since last copied or backed up, and it will tell you the date set the last time the file was updated. In short, the DIR command reveals all the information about a disk file.

The ability to specify the output channel when using DIR means that you may direct the directory output to the printer, the serial port, a disk file, or whatever is a legal I/O channel. If you remember from the operations section, and I/O channel can be any character orientated I/O path (devicespec or filespec). This allows output to almost anything save a drivespec or a wildmask.

The ability to use a wildmask with DIR means that if you desire the information for one specific file, or the information for a specific class of files, whether or just one specific drive, or on all drives, it is a simple procedure.

You simply set the wildmask to be as specific as you need. If you make a wildmask specific enough to weed out any extraneous files, then you only view the files that you are interested in. If you wish this display to check all drives, simply omit the drivespec from the wildmask. To restrict it to one specific drive, include the drivespec in the wildmask.

The simplest form of DIR is :

```
DIR
```

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

```
DIR :1
```

This accomplishes the same effect, but restricts it actions to the disk in the drive named ":1".

The directory display -

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

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

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

The drive name. This is the current drive specification for the drive being directoried. The drivespec is, of course, the two character designation (preceded by a colon ":") that you address the disk drive through.

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

The disk date. Again, at time of format, both the FORMAT and RFORMAT utility will prompt you for the date (unless the system date is set, in which case they will simply use that). When they format the disk, they will store this date on the directory. This date will then be displayed in the same operations as the disk name (see above). This allows you to get a fair idea of how old the disk is.

The directory entry status. DOSPLUS II only allows a certain number of entries per directory (256 maximum). Once you have filled this up, you may not create any new files on that disk regardless of its free space because there is no room to put it in the directory. This does not necessarily mean that you cannot extend existing files. As long as they don't need to create an extended entry and there IS some free space on the disk, this should work. The directory entry status display tells you how many file entry positions you have free (i.e. how many files remain) and how many possible entries there were for that drive. The second number will never change, but when the first number reaches 0, the directory is full.

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

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

* Filename.
* File attributes.
* Logical record length.
* Number of records.
* Number of physical sectors.
* Number of segments.
* Total size in kilobytes.
* Date last updated.

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

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

The file attributes. These six characters represent the special "attributes" that a file may have, such as the "Keep" parameter, the "Mod flag", and many others. Each character means something different. If a particular attribute is NOT present (i.e. The "Kill" attribute will not show up on an active file), the attribute character for that position becomes a period ("."). The characters are :

Number one. The protection level. This indicates the level of protection assigned to that file. The protection levels are detailed in the ATTRIB command, so if you wish a complete explanation of the parameter, look there. Here we will just list the parameters for reference sake.

Number	Name	Protection level
0	Full	No protection set. Total access.
1	Kill	Able to delete file.
2	Rename	Rename, Write, Read, Execute.
3	----	Not used at this time. Reserved.
4	Write	Write, Read, Execute.
5	Read	Read, Execute.
6	Execute	Execute only.
7	None	No access. Not a user option.

Since there will always be at least a protection level of "Full", there will always be a character in this first position. If no protection has been set, it will be a "0".

Number two. The password status. You may set a password for a file independantly of the protection level (although it is really not effective to not use both). This position will indicate to you whether or not a file has a non-blank password set for it or not.

It will NOT differentiate between an Access and an Update password. If either or both are set, then a "P" appears in this position. If neither of them are currently set, *there will be a period there.*

Number three. The system file flag. If a file has the attributes of a system file, this position will carry an "S". The user does NOT have the option of configuring their own files as system files. This is reserved for the actual system files that make up DOSPLUS II. This parameter is in no way related to the "/Sys" extension.

Number four. The keep flag. If a file has the "Keep" parameter set for it (see CREATE and ATTRIB for a further explanation of this parameter), then an exclamation mark ("!") will appear in this position. If this flag is not set or gets removed, a period will be shown in that position.

Number five. The kill flag. If a file is currently deleted (i.e. inactive), there will be a "K" in that position. If the file is active, then a period will be shown. Normally, killed files do not appear in the directory. They only show up if you have requested them with the "Kill" parameter.

Number six. The invisible flag. If a file is invisible to the system, then an "I" will appear in this position. If a file is visible, a period will be there. As with the killed files, invisible files should not appear in a directory unless you have asked for them.

Number seven. The mod flag. This flag is the most often set and reset attribute flag. This flag indicates that the file has been modified (i.e. written to) since the disk was last backed up or that particular file was last copied. If this IS the case, a plus sign ("+") will appear in that position. If the mod flag is NOT set, then a period will appear at that location. COPY will allow you to copy keying on this flag. For instance, you may copy all the files that have been modified since the disk was last backed up, or you may copy all the files that have NOT been modified since the disk was last backed up. A very useful indicator to have in the directory entry.

An example of an attribute column with all the flags lit up would be "6PS!KI+".

The logical record length. For machine language programs and data files that are to be accessed by machine language programs, this figure is essentially for display only. For BASIC programs, though, it becomes VERY important. If the logical record length of a file is not properly specified when you issue an OPEN command, BASIC will not allow you to open that file. This will be expressed as a numeric value between 1 and 256. If the logical record length is less than 256, the "Number of records" display will be a larger number than the "Number of sectors".

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

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

The number of segments. When DOSPLUS II creates a file, it STARTS it at a random place on the disk. From there, it will seek to extend the file sequentially for maximum efficiency. When it is unable to do that, for whatever reason, it stops that extension of the file and creates a new one at another random location on the disk. These extensions are called segments. When a disk is almost full, with only a few gaps of free space here and there, files will tend to get segmented a great deal. The more segmented a file is, the less efficient the disk access.

If the number is this column is very high and the file itself is NOT extremely large, you could improve the efficiency of your disk access times by reducing the number of segments in the file. The method of doing this is simple. Prepare a disk with a great deal of free space on it. Then copy the files that need to have the number of segments reduced to it. Copy the larger files first, as they will need the most space. When you finish transferring the files, you should have reduced the number of segments in them.

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

The date last updated. As you know, DOSPLUS II maintains a "system date". On power-up, the DOS will prompt you for the date (unless you have disabled this question by using the SYSTEM command). It will preserve this date in memory and any time that it would normally ask you for the date (FORMAT, BACKUP, etc.), it will skip that prompt. One additional feature of having the date set is this display in the directory. Each time that you access a file, the current system date is updated to that file's directory and displayed via the DIR command. If this system date is not set, the date "01/01/80" will be used.

Using DIR -

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

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

```
DIR @PR
```

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

```
DIR TO @PR
```

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

```
DIR :1 @PR
```

is fine. "@PR" is in the proper position for an output channel and all will work well.

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

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

is the same thing as :

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

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

```
DIR TEST/DAT
```

will produce an error, while :

```
DIR USING TEST/DAT
```

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

Examples:

```
DIR :0 {SYS=Y,INV=Y,KILL=Y}
DIR :0 {SYS,INV,KILL}
DIR :0 {S,I,K}
DIR :0,S,I,K
```

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

```
DIR USING PER/DAT
```

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

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

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

```
DIR :1 {INV=Y,ALPHA}
DIR :1 {I,A}
DIR :1,I,A
```

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

Finally:

Remember that the default output channel is the display. If you omit the output channel, then the directory will be sent to the screen. This can cause a problem with your wildmask field if you are not careful.

You see, the wildmask field follows the output channel field in the command line. If the following conditions are true :

(1) You have specified a source drivespec so that the field is full in that position.

(2) You have omitted the output channel in order to send the output to the default channel: the video.

(3) You have omitted the delimiter USING because you feel that the wildmask is in its proper place.

(4) You do not have any wildcard characters in your wildmask that might show DOSPLUS II that it IS a wildmask.

Then DOSPLUS II will overwrite the file you have specified in the wildmask with the output of the catalog. Without the USING delimiter and with the output channel not specified, the wildmask moves into the output channel area. Because the source drivespec WAS specified, it won't try to use the wildmask for that and generate an error there. Finally, because there are no wildcard characters to indicate a wildmask, DOSPLUS II takes it as a standard filespec. Since a filespec is a valid output channel, it gets overwritten. Therefore, a little bit of carelesness can destroy the very file you were looking to find.

What all this means is, if you are going to use a wildmask with the source drivespec specified and you are NOT going to specify an output channel and your wildmask does not have any wildcard characters in it, then you must use the USING delimiter.

For a more detailed explanation of wildcards and wildmasks, consult the operations portion of the manual under "File and device specifications".

DO

This command allows you to begin execution of a command chaining file. These are more commonly called "DO files". They are usually created with the BUILD command (see the library command BUILD). They can be created with any program that creates ASCII files, though. They will contain command lines up to 79 characters in length and terminated by a carriage return. You may abort execution of a DO file by pressing the BREAK key at a pause prompt, unless that function has been disabled.

==========

The command syntax is :

DO filespec {param=exp}

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

"param" is the optional action parameter that modifies the operation of the command.

"exp" is the optional expression that indicates what type of action the parameter it modifies will take.

Your parameter is :

BREAK=switch	Break enable/disable. This parameter allows you to disable the BREAK key abort feature of DO. Normally, you may terminate the execution of a DO file by pressing the BREAK key at a pause prompt. If you specify this parameter, that will be disabled. If you set BREAK="N", then the BREAK key abort will not function. On the other hand, BREAK="Y" will re-enable the BREAK key if it has been disabled previously either here or with the SYSTEM command.

Abbreviation :

BREAK B

==========

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

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

When DO reaches the end of a list of commands, it will return control to DOSPLUS II. The "DosPLUS II" prompt will NOT be immediately re-displayed, though. Only the cursor will be on the screen. If you wish, you may press ENTER to regain the prompt, but it is not needed. Commands may be entered as soon as the control is returned to the keyboard.

Applications of DO -

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

(1) Startup for applications programs.

(2) Routine sets of often used instructions.

(3) Installing patches to the system.

(4) Automatic operation of programs.

In the first, startup for applications programs, DO is perhaps most useful. Many of your applications programs, especially those written in BASIC, will require that you set the FORMS command to certain values or set up a buffer for the comm line with SETCOM, or any operation that needs to be performed before your programs will work correctly.

DO allows you to do this automatically without forcing the novice user or non-technical operator to remember DOS syntax. Simply set up a file with BUILD (see the library command BUILD) that contains the needed sequence of commands. Remember, enter these EXACTLY as you would if you were entering them in the DOS command mode. Then you would have the last statement in your DO file call your program (i.e. BASIC MENU/BAS-F:7).

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

```
FORMS {BS=10000,XLATE=N}
BASIC PAYROLL-F:5
```

Then we would set the statement :

```
AUTO DO STARTUP
```

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

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

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

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

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

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

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

There IS one important exception to this. BASIC programs will not allow this feature. BASIC checks for the BREAK key by scanning the keyboard. Every time that in executes an instruction, BASIC looks to see if BREAK is being pressed. Every time it does that, you lose a character from the DO file. Eventually, the entire file has been used up. This is a function of BASIC, and there is no way around it at this time.

Examples:

```
DO STARTUP {BREAK=Y}
DO STARTUP {B=Y}
DO STARTUP,B=Y
```

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

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

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

Finally:

If you wish to create a "non-breakable" DO file, use the non-breakable AUTO command (see the library command AUTO) and then call your DO file and turn the BREAK key off as you do. The user will have to execute all the way through the file and into whatever program it calls.

DO will use some memory when it executes. It needs to set up a buffer for I/O from the file and a DCB for the file. If any of the programs you happen to load are going to load into the area that DO is using, it will crash at once. DO will adjust the high memory pointer to protect itself. To be safe, please have your programs honor that value.

Remember once again, DO may NOT be used from BASIC.

DUMP

This command allows you to take a specified area memory and transfer it to disk as a file. This can either be data or a program file. There is an optional parameter to control the installation of load file markers in the file in the case of data being stored. Entering "DUMP filespec" will cause all memory between LOMEM and HIMEM (see the library command SYSTEM), an area usually referred to as "user memory", to be transferred to the specified disk file.

==========

The command syntax is :

DUMP filespec {param=exp...}

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

"param" is the optional action parameter that modifies the action of the command.

"exp" is the optional expression that indicates what action the parameter it modifies will take.

Your parameters are :

DATA=switch

Data file format. A normal machine language program file contains something called "load file format markers". These are special block headers in the text of the file that indicate to DOSPLUS II where in memory to load the instructions that follow. A data file will not have these markers. If the memory you are dumping to disk is NOT a machine language program, then it should be treated as a data file and this parameter should be engaged. If you do not specify an extension, either the extension "/CMD" or "/CIM" will be used. "/CMD" if you have NOT selected the DATA parameter and "/CIM" if you have.

END=value

Ending address. This allows you to specify at what memory address you wish DUMP to stop at. If this parameter is not specified, HIMEM is assumed, and DUMP will continue until it reaches that point. This may be given in any form (decimal, hex, binary, or octal). Simply remember to append the correct type specifier to the value if you are specifying something other than decimal.

RELO=value

Relocation address. This parameter allows to to specify a different load address for a file than the one it had when you dumped it to disk. In other words, if you are dumping a program from one area of memory to disk, but you wish that program to load in at another location in memory later, then you would use this parameter. This may also be entered in any standard value input notation.

START=value

Start address. This parameter allows you to specify the address at which you wish DUMP to begin transferring to disk from. If this is not specified, LOMEM will be assumed and DUMP will begin from that point. This may again be entered in any standard value input notation.

TRA=value

Transfer address. This parameter allows you to set a transfer address for the programs you DUMP to disk. A program's transfer address is the location in memory that control passes to after the program file has been loaded. It does not have to be the same as the load address (and often is not). If this is not specified, then the value 0000H (DOS entry point) will be used, and you will be returned to DOS after the file is loaded.

Default values :

DATA	No. Load markers will be included and the extension "/CMD" used for a default.
END	HIMEM.
RELO	START.
START	LOMEM.
TRA	0000H.

Abbreviations :

DATA	D
END	E
RELO	R
START	S
TRA	T

==========

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

Once the file is on the disk, in the case of machine language programs, you may either execute the file directly by typing in the filename from the DOS command mode or you may load the file via the LOAD command and execute it via DEBUG or by specifying the "Run" parameter on LOAD (see the library commands LOAD and DEBUG).

By using the DUMP command, you may enter machine language programs into memory via the modification mode of the DEBUG command and then dump them into a disk file to be executed later.

The DUMP command in DOSPLUS II is unique in the manner in which it allows you to completely control all items of information about the file as you are saving it to disk. You may alter the load address of the program or change the transfer address, whichever you choose.

When transferring memory to a disk file, if you wish the system to store it as data and not a machine language file, then you must remember to specify the "Data" parameter. Machine language programs and data files are stored on the disk in two completely different manners.

When a machine language program is stored on the disk, there are two pieces of information that the CPU needs to know in order to load and execute it properly. First, where the program loads or its "load address". Second, where in memory to pass the program control to after the program has been loaded or its "transfer address".

Under certain circumstances, you may want to alter either or both of these. You may wish, for example, to dump memory between 7000 hex and D000 hex to disk, but to have the system load it back later from 6000 hex to C000 hex. When you dumped that file, you would use the "Relo" parameter. You would set that parameter to "6000H" and from then on, when the system loaded that file, it would start at 6000 hex.

Some programs also use a transfer address that is different from their load address'. In other words, the program does not actually begin executing at the exact same location in memory as it loads. This is commonly handled by the assembler creating the object file, but in the case of a file being dumped to disk, that obviously would not apply. Therefore, DOSPLUS II's DUMP command allows you to set a transfer address for the file.

If you specify neither the "Relo" or the "Tra" parameter, DUMP will assume that : (1) the program loads back into memory at the same area that it came from and (2) the program is NOT to be executed, but instead, you wish to return to DOS after loading the file.

Examples:

```
DUMP TESTFILE {START=7000H,END=D000H}
DUMP TESTFILE {START=28672,END=53248}
DUMP TESTFILE {S=7000H,E=D000H}
DUMP TESTFILE,S=7000H,E=D000H
```

All four of these commands will have the same effect. They will attempt to write the area of memory between 7000 hex and D000 hex to the first available disk drive under the filename "TESTFILE/CMD". The load address and the transfer address for the file will both be left set to 7000 hex. Note that the decimal input was used interchangeably with the hexidecimal.

```
DUMP DATAFILE/DAT:1 {START=3000H,DATA}
DUMP DATAFILE/DAT:1 {S=3000H,D}
DUMP DATAFILE/DAT:1,S=3000H,D
```

These three commands will all accomplish the same effect. They will move the area of memory from 3000 hex to whatever HIMEM is currently set to a disk file named "DATAFILE/DAT" located on drive ":1". It will store the information on the disk in data file format (as opposed to load file format).

Finally:

Because DUMP allows you to dump memory to disk as a data file, it IS possible for you to rescue a BASIC program in memory by using this command. Possible, but very difficult. In order for it to work, you MUST know exactly where the program text started in RAM. BASIC uses a series of "00" bytes to keep track of the end of its programs. So, if you know where the program started in RAM, DUMP from there to the top of memory. BASIC will find the end of file itself when you load what you have dumped, and then you should be able to re-save a corrected file. Not much of a chance, true, but it is possible for the experienced programmer or user to avert potential disaster with it.

The most important thing of all is, because of DUMP's flexibility, DOSPLUS II's DUMP command literally makes it possible for the user who knows what they are doing to move ANYTHING to disk!

ERROR

This command allows you to get a get a detailed message printout of any error number or a display of the last error displayed, depending on the syntax used.

==========

The command syntax is :

ERROR [value]

"value" is the optional error number that you wish to obtain a message for. If omitted, the last error displayed will be re-displayed.

==========

DOSPLUS II does provide you with detailed error messages instead of numbers, but for TRSDOS compatibility and reference's sake, this command will still translate error numbers into error messages. A complete list of the error messages will be published in the technical section of the manual.

ERROR also has the unique feature of recalling the last error displayed. Simply enter the word "ERROR" without a number and the resulting message will be the last error the system displayed.

As stated before, DOSPLUS II itself always prints out a detailed error message. However, some applications software, in keeping with TRSDOS tradition, may give you simply an error number. This command allows you to quickly see what the error message for that number is.

Other programs may use the ERROR command in TRSDOS within the actual program. For that reason, we have left this command in DOSPLUS II.

A very useful application of the ERROR command is the error "replay". If an error occurs and the message is scrolled off the screen, or for some other reason you are unable to read it, this will allow you to pick up the error message later.

Examples:

```
ERROR
```

This command will print to the screen the message corresponding to the last error displayed.

```
ERROR 32
ERROR 20H
```

These two commands are equivalent. They will both print the error message pointed to by Error 32. Note the use of hexidecimal input. This is perfectly legal as long as you add the trailing "H".

FILTER

This command allows you to set up a "filter" on any of the character orientated system devices excluding the user defined devices (i.e. devices 0-4). This filter can translate any given value into any other value.

==========

The command syntax is :

FILTER [FROM] devicespec [TO] filespec {switch}

"devicespec" is the name of the device you wish to install the filter on.

"filespec" is the name of the file that contains the filter. "/FLT" is the default extension if no other is given.

"switch" is the optional action switch. By specifying a "N" in that position, you may load a filter into high memory but not turn it on. Later, you can use the form :

FILTER devicepec {switch}

where "{switch}" is "Y" to turn on the filter. You may also use this to turn installed filters on and off under program control. Because of this, you do not have to de-install and re-install filters to turn them on and off. This format of the command will also display the current filter settings.

==========

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

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

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

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

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

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

```
FILTER @KI DVORAK
```

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

Writing a filter file -

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

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

```
65 decimal
41 hex
101 octal
01000001 binary
"A" quoted literal
```

This capacity makes it easy for even the novice user to create these files. The next step is to include the equals sign ("=") to indicate that the translated value follows. Then you may express the new value for this character. This may also be expressed in almost any manner. Let's assume you wish to translate that capital "A" into a lower case "a". The statements could look something like these :

```
65=97
41H=61H
101O=141O
01000001B=01100001B
"A"="a"
```

you may also mix and match the types :

```
65=61H
41H="a"
"A"=141O
```

Notice that we had to append an "H" to any hexidecimal values, an "O" to any octal values, and a "B" to any binary values. Decimal values are assumed and quoted literals are obvious to the system. Once you have completed the first translation, separate it from the next with either a comma or a space and proceed for as many of these as you need.

An actual example of a filter file might be to filter out certain codes that would cause your printer to go into special print modes. Let's assume those codes are 14 decimal and 15 decimal. Our filter file would be short (only two translations), and would look something like this :

```
14=00 15=00
```

or it could be :

```
0EH=0,FH=0
```

Note that the leading "0" on the hex values is optional. Note that the "00" and "0" expressions are the same. Note that in the first example we separated the translations with a space and in the second we used a comma. Let's also assume that we store this on the disk with a filename of "PRT/FLT". We would then say :

```
FILTER @PR PRT
```

to install the filter. You can create these filter files using the BUILD command (see the library command BUILD).

Examples:

```
FILTER FROM @DO TO DISPLAY/FLT
FILTER @DO DISPLAY
```

These two commands are equivalent. Notice that the FROM and TO words are optional and that the extension "/FLT" is assumed. These will install the filter "DISPLAY/FLT" on the video device.

```
FILTER @DO
```

This command will display any filter that is currently in effect for the video device. If one is NOT present, it will inform you that "No function exists". If one IS present, it will list that filter to the display. It will print the character, unless it is unprintable in which case it will print a period, followed by the value for that character in parenthesis. The equals sign and the new value (in the same format) will follow.

```
FILTER @DO {NO}
FILTER @DO {N}
FILTER @DO,N
```

These three commands will all dis-engage the filter currently on the video device. They will NOT deinstall the filter. Memory will still be reserved and the filter is still there if they wish to re-engage it.

```
FILTER @DO {YES}
FILTER @DO {Y}
FILTER @DO,Y
```

These three commands will re-engage any filter installed on the video device.

Finally:

Any time that you dis-engage or re-engage a filter, the filter will be displayed just the same as if you had requested a display for that device.

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

```
FILTER @DO DISPLAY {NO}
```

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

Once a filter is installed on a device, there is no way to actually remove it from memory short or re-booting the system. You may turn the filter "Off", but you can never remove it and reclaim the memory.

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

FORMS

The FORMS command allows you to set the defaults for your lineprinter. Parameters such as page length, page width, and number of printed lines per page can be set with this command. This command will also allow you to direct lineprinter output to either the parallel printer port or to serial port B. FORMS without any parameters will display the current settings.

==========

The command syntax is:

FORMS {PARAM=exp,PARAM=exp....}

"PARAM" is the parameter to be set or altered. These are listed below. "exp" is the expression which gives the new value. Depending on the parameter, it may be a switch (Y or N), or a numeric value.

The parameters for the FORMS command are:

PAGE=value	Maximum number of lines per page. Standard pages are normally 66 lines.
LINES=value	The number of lines per page that will be used before a top-of-form is executed (normally 60).
WIDTH=value	Number of characters, including blank spaces, that will be printed on a line before a carriage return is forced (normally 132).
TOP	Sends an immediate top-of-form to the lineprinter
XLATE=switch	Determines whether form feed codes are changed to a series of line feeds for printers that do not accept the formfeed code.
	LF=switchDetermines whether a linefeed is to be sent after each form feed, for printers that require it.
SERIAL=switch	Determines whether printer output is to go to Serial Port B rather than the normal printer port.
BS=value	Sets the size of the printer spool buffer.
CODE=value	Sends the specified code out to the printer immediately. Codes may be one or two bytes.
RESET=switch	Resets the system's line counter to 0. Does not affect the actual lineprinter.

UCASE=switch	Forces all alphabetic output to the lineprinter to be in UPPER CASE, for printers that do not have lowercase.
EMPTY=switch	Discards contents of the spool buffer.

All of the above parameters may be abbreviated to their first character. Thus, PAGE may be specified as P, LINES as L, etc.

The initial values for each of the parameters are as follows:

PAGE	66 lines
LINES	60 lines
WIDTH	132 characters
XLATE	Yes. Translate form feed codes to line feeds.
LF	No. No linefeeds after carriage returns
SERIAL	No. Use the standard parallel printer port
BS	Previous value of buffer size.
CODE	None.
RESET	NO.
UCASE	YES.
EMPTY	NO.

When the system is booted, these initial values will be set.

==========

When FORMS is typed without any parameters, the current settings will be displayed on the screen.

The {PAGE=value} parameter sets the page size in terms of the number of lines that can physically be printed on it. Standard 11 inch paper can hold a maximum of 66 lines (6 lines to the inch). This parameter will inform the system of the page size (distance from the top of one page to the top of the next), but the number of lines that will actually be printed is determined by the LINES parameter.

The {LINES=value} parameter sets the number of lines which will actually be printed on each page. The value of this parameter may not exceed the value given for PAGE. If the value of the LINES parameter is less than the value of the PAGE parameter, then the system will execute a top of form to the next page as soon as the number of printed lines equals that of the LINES parameter.

If the value of LINES equals that of PAGE, then the system will not execute an automatic top of form. However, form feed codes (0CH) and vertical tab codes (0BH) will be translated to the correct number of carriage returns for proper pagination.

If the values of both the LINES parameter and the PAGE parameter are 0, then the system will assume an infinite page size, and no top of forms will be executed. However, form feed codes (0CH) and vertical tab codes (0BH) will be passed on to the printer without translation.

The {WIDTH=value} parameter determines the number of characters that will be printed on each line. On a printer which uses normal characters, an 8-inch wide sheet of paper will hold 80 characters per line, while 14.5 inch wide paper will hold 132 characters per line. When the number of characters printed on a line reaches the value of the WIDTH parameter, the system will break the line at that point by inserting a carriage return, and any remaining characters will be printed on a new line. The line counter will also be incremented by one.

Normally, the system will translate tabs to the necessary number of spaces need to reach the next tab field, which are 8 characters apart, and the system's character counter will be incremented by the correct amount. However, if the WIDTH parameter is set to 0, indicating an infinite line length, tab codes (09H) will not be translated but will be sent out to the printer as is, and will be considered a single character when incrementing the system's character counter.

An example of FORMS to set the printer for 80 characters per line, 60 lines per page, on standard 8-1/2 x 11 inch paper would be:

FORMS {WIDTH=80,PAGE=66,LINES=60}

An equivalent form would be:

FORMS {P=66,L=60,W=80}

You will note that the parameters can be given in any order.

The TOP parameter is a special parameter which immediately outputs a top-of-form to the printer and resets the system's line counter to zero. It does not require any switch or value. If the TOP parameter is included along with other parameters, then it will be executed before any other parameter is evaluated.

The {XLATE=switch} parameter determines whether or not certain control codes are translated instead of being output directly to the lineprinter. The codes affected are the form feed code (0CH), the vertical tab code (0BH), and under special circumstances, carriage returns (0DH).

If XLATE is on (XLATE=Y), then:

(a) Form feed and vertical tab codes are translated into the necessary number of line feeds to correctly execute a top-of-form based on the current value of the PAGE parameter;

(b) Carriage returns which are sent out by themselves (ie, the line consists of only a single carriage return) are translated into line feeds in order to correctly produce a blank line on the printer.

If XLATE is off (XLATE=N), then no translation takes place, and form feed and vertical tab codes are treated just like any other character. They are passed through to the printer, and the character counter is incremented by one for each form feed or vertical tab code encountered.

The {LF=switch} parameter will permit linefeeds to be sent out after carriage returns for those printers that need them. Normally, printers which receive a carriage return automatically perform a line feed, but for printers which do not do so, this parameter is available. Setting LF=Y will not affect the line counter any differently.

The {SERIAL=switch} parameter, if turned on (SERIAL=Y) will send all lineprinter output to the Communications Line B channel of the Model II's RS232 system. This would normally be used if a serial printer was being used instead of a parallel printer. Please note that the DOSPLUS II serial printer driver does not support the XON/XOFF handshaking protocol used by other serial printer drivers.

The {BS=value} parameter allows you to set the buffer size to be used by the printer spooler. Normally, the buffer is 2 characters long, but you can change it to any value you want with this parameter. When you specify a value larger than 2, DOSPLUS II will establish a buffer in high memory and lower its HIMEM pointer to protect it.

Changing the spool buffer size may be useful if you have a particularly slow printer. The buffer size cannot be less than 2.

The {CODE=value} parameter is another special parameter. It sends the value of CODE directly out to the printer before any other parameters are evaluated, and is useful if you have a printer that requires special codes to set it up. CODE will take either a one byte or a two byte value. However, if you want to send a two byte value to the printer using this parameter, you must specify them in the reverse order. For example, if you wish to send a CTRL-D (04H) to your printer, you would enter CODE=04H; but if you wish to send the sequence ESC "E" (1BH, 45H) to the printer, you must specify CODE=451BH.

The codes may be specified in any of the numeric bases accepted by DOSPLUS II (decimal, octal, binary, or hexadecimal) but it is recommended that hexadecimal be used when sending a two-byte code, since it is easier to specify in the necessary reverse order.

The {RESET=switch} parameter will reset DOSPLUS II's internal line and character counters back to 0 without sending any codes out to the lineprinter. This is useful if the lineprinter paper is manually adjusted back to the top of form. RESET=Y will then start the line counter from the first line once again.

A FORMS {T} command will also perform a RESET of the line counter, however it will also cause the lineprinter to advance to the top of the next page.

The {UCASE=switch} parameter will force all alphabetic output to the lineprinter to be in uppercase if Y is specified for a switch value. This is useful if the lineprinter being used does not support lower case letters.

The {EMPTY=switch} parameter allows you to remove any characters remaining in the printer device's spool buffer. This is useful when you assign a large buffer size with the BS parameter and later wish to abort a long printout. Normally, printing will continue until the spool buffer is empty. This parameter, however, provides you with a means of discarding the buffer's contents quickly without waiting for the printer.

Examples:

FORMS

Displays the current forms settings on the video display.

FORMS {PAGE=66,L=57,TOP}

Generates a top-of-form to the printer, and then sets the page length to 66 lines with 57 printed lines per page. Note that even though TOP was the third parameter in the list, it is acted upon first by the system.

FORMS {S=Y,B=256,WIDTH=120,XLATE=OFF}

This command would direct the system to send all printer output to Serial Port B using the serial driver. In addition, it specifies a line length of 120 characters, and establishes a spool buffer 256 characters long. Finally, it indicates to the system that form feed codes and vertical tabs should not be translated to linefeeds.

FREE

This command will display the remaining free storage space and remaining available directory space on all mounted disks. The free disk space will be given in kilobytes. If FREE is given a drivespec, then a map of the disk in that drive will be displayed, showing the locations of all available as well as allocated sectors.

==========

The command syntax is:

FREE
FREE [FROM] drivespec [TO] channel

"drivespec" is the drive specification of the drive for which a free space map is to be displayed. If omitted, then free space on all mounted disks will be displayed in summary, not map, format.
"channel" is the output channel to which the information is to be sent. It defaults to @DO, the video display, but may be set to any valid output channel.

The FREE command has no parameters.

==========

The FREE command, when given without any drivespecs, will read the directory of each mounted disk and determine the amount of space available on that disk. "Mounted disks" includes logical drives (such as those on a hard drive which has been split up into one or more logical drives). The free space remaining on each disk will then be displayed, along with the number of available directory slots for new files. Free storage space will be given in kilobytes.

It is quite possible that a disk may have free disk space remaining, but has a full directory. In this case, even though there is storage space remaining on the disk, no new files may be placed on it because there is no more room in the directory to hold information about that new file. Conversely, there may be available directory slots, but no free storage space remaining on the disk. Dosplus II will create the file but will not allocate any space to it in this case.

If FREE is given with a drivespec, then DOSPLUS II will read only that drive, and then display a map of the disk. The map will show all the formatted sectors on the disk and indicate which ones are in use, which ones are free, and which ones are unavailable (locked out). Granules allocated to a file will be displayed with an "X", free granules with a "." (period). The directory track will have its granules displayed with "D."

The information generated by the FREE command is normally sent to the video display, but may be sent to any valid output channel simply by specifying the channel. Output channels may be the lineprinter, the RS-232 communications lines, a disk file, or any valid user-defined output device.

Examples:

FREE

This command will display free space information for all mounted disks on the video display.

FREE TO @PR

This command will send the free space information for all mounted drives to the lineprinter.

FREE :L1 TO @PR

This command will send a map of the disk in drive :L1 to the lineprinter. The "TO" is optional; an equivalent form of the command would be FREE :L1 @PR

FREE TO FREEINF/TXT:A

This command will send the free space information for all mounted disks to a file called FREEINF/TXT on drive :A.

I

This command will set the system up to read the DCT information from a disk into memory. This is mandatory when switching from a double-sided floppy disk to a single sided disk or vice versa.

==========

The command syntax is:

I [:ds] {MOUNT}

where :ds is the drivespec of any valid drive in the system (optional).

The parameter for the I command is:

MOUNT=switch	Reads the DCT information from the disk into memory immediately.

MOUNT may be abbreviated M, and its initial value defaults to NO.

==========

When the I command is issued, the system will be flagged to read the Drive Control information from a mounted disk into memory at the next disk access. If no drivespec is given, then the system will flag the DCTs of all mounted disks as necessary and read the information from each disk at the first access of that disk following the I command. If a drivespec is specified, then the system will set the DCT just for the disk in that drive. This is necessary only when you are using double-sided floppy disk drives and wish to swap a single-sided disk for a double-sided one, or vice versa. DOSPLUS II needs to know whether a disk is double-sided or not. The I command will allow it to determine this information.

If all your disks are formatted identically, then it is not necessary to use the I command whenever you insert a new disk into a drive.

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

Examples:

I

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

I :D0 {MOUNT{

This form of the I command will cause the system to flag drive :D0 for its drive control information. If the disk in drive :D0 was a double sided disk and you wish to read a single sided disk in the same drive, mount the single-sided disk and then issue this command.

If the new disk is formatted identically to the one already in drive :D0, then this command is unnecessary; however, issuing it will do no harm.

KILL

This command will delete a file or group of files from a disk. It will also disable any active devices or drives (the system drive -- device 8 -- may NOT be killed).

==========

The command syntax is:

KILL filespec[:ds] [{PARAM=exp,PARAM=exp...}]
KILL wildmask:ds [{PARAM=exp,PARAM=exp...}]
KILL drivespec
KILL devicespec

"PARAM" is an optional parameter affecting the action of the KILL command. "Exp" is an expression which declares to the system a particular value for that parameter (a switch, a numeric value, or a string value enclosed in quotes).

The valid parameters for this command are:

INV=switch	Specifies whether or not invisible files are to be included when a wildcard search is done.
ECHO=switch	When doing a wildmask search, this will display the name of each file as it is killed.
SYS=switch	Specifies whether or not system files are to be included in a wildcard search.
QUERY=switch	This will cause the system to display the filespec and prompt you for a yes or no reply before proceeding. If you reply "yes" or "y" then the file will be killed. If you reply "no" or "n" then that file will not be deleted.
PW="string"	This parameter declares the DISK master password to the system, which will be used in place of file passwords when wildmasks are specified.

Each parameter may be abbreviated to its first character.

The default values for each parameter are:

INV	NO
SYS	NO
QUERY	NO (do not display the file name or prompt the user)
PW	Defaults to no password. However, if the PW parameter is used, then the disk master password must be explicitly stated. Otherwise, it would be included as part of a single filespec, if needed.
ECHO	NO

==========

When KILL is typed with a filespec, but without a drivespec, the system will perform a global search of all mounted disks until it finds the first occurrence of the file, which it will then delete. If a drivespec is supplied, then only that drive will be searched.

When KILL is typed with a wildmask, and no drivespec is supplied, the current system drive will be searched. The system will search the directory of the specified drive for the first filename that fits the wildmask, and kill it. It will then continue to search for other files which will fit the mask, killing each one that it finds.

When using a wildmask, the KILL command requires that the disk's master password be given with the PW parameter. In this case, the disk master password will be used in place of the file passwords when a password protected file is encountered.

The {INV=switch} and {SYS=switch} parameters are used when doing a multiple-file kill using a wildcard mask. Normally, this type of multiple-file kill includes only visible user files. However these two parameters allow you to include invisible and system files. The INV parameter will include invisible files, and the SYS parameter will include system files (except for BOOT/SYS and DIR/SYS) in the wildmask search.

The {ECHO=switch} parameter may be used when doing a wildmask search, to display the names of the files as they are KILLed.

The {QUERY=switch} parameter will force the system to display the filename before killing it, and prompt the user for a yes or no reply. The file will be killed only if you specifically reply "yes" or "y" when prompted.

This parameter is useful when deleting a group of files using a wildmask search. Whenever a wildmask search is specified, QUERY will default to YES unless you specifically enter QUERY=NO. If there are files which fit the wildmask but which you do not wish killed, using this parameter will allow you to preserve them selectively.

The QUERY parameter will override the ECHO parameter.

The {PW="password"} parameter declares the disk's master password to the system. This password will be used instead of the file access and update passwords when a wildmask search encounters a protected file. The disk's master password must be enclosed in either single quotes or double quotes and may be in upper or lower case, or both.

When doing a wildmask KILL, this parameter is REQUIRED. The disk master password MUST be specified even though the files are not password protected if wildmasks are used.

KILL may also be used to disable devices. If a devicespec is given, then a bit will be set in that device's DCB indicating that it is set to NIL. It will become unavailable until it is re-entered in the table by means of the SET command (see below). For example, KILL @PR will remove the lineprinter device from the system.

Similarly, disk drives may be disabled with KILL, with the exception of the currently-defined SYSTEM drive. The system drive may not be killed. When disk drives are KILLed, they are also set to NIL.

When I/O is performed to any KILLed output device, no error message will be returned; however the data will go nowhere.

When a disk drive is KILLed, any attempt to access that drive will return the error message, "Device not available."

Care should be taken when using KILL to disable devices. The only way out of injudicious use of this command may be to reboot (for example, KILL @KI will disable the keyboard; the only possible recovery from this case would be to reset the entire system). However, if a device is linked to another, the other device will continue to be active even if the first device is KILLed.

KILLed devices and disk drives may be restored to an active state by SETting the device back to itself, for example, SET @PR @PR (See the SET command, below).

Examples:

```
KILL FOOBAR/CMD
```

This command will cause the system to search the directories of all mounted disks for the first occurrence of FOOBAR/CMD, which will then be killed.

```
KILL FOOBAR/CMD:1
```

This command will cause the system to search the directory on drive :1 for FOOBAR/CMD. If it finds the file, the file will be killed. If the file does not exist on that directory, the system will return a "File not found" error.

```
KILL */OLD:A4 {QUERY,PW='Mydisk'}
```

The system will search the directory of the disk on drive :A4 for every file with the extension /OLD. It will then display the filename that it finds which fits the wildmask and ask the user whether that file is to be killed or not. If the user replies "Y" or "YES," then the file will be killed. Otherwise the file will be left alone, and the search will continue for other files with the /OLD extension.

KILL PROG?/BAS:01 {PW="PASSWORD"}

This command will search the directory on drive :01 for any filename that fits the wildmask PROG?/BAS and kill them. Files with names such as PROG1/BAS, PROGA/BAS, PROG$/BAS, etc. would be killed. The disk's master password must be supplied.

KILL MYDATA/DAT.SECRET:0A

This command will remove all traces of the file called MYDATA/DAT.SECRET from the disk in drive :0A.

KILL :02

The disk drive designated as :02 will be disabled, UNLESS it is the system drive. If drive :02 is the system drive, then this command will not be executed. If it is not the system drive, it will be disabled and any attempts to read or write drive :02 will produce an error.

KILL */*:X1 {P="CIA"}

This form of the KILL command is a global KILL. Any filename will fit the */* wildmask form, so the use of this command will result in every file on drive :X1 being killed. In this case, QUERY will automatically default to ON and the user will be prompted as each file is killed.

KILL !:00 {P="MYFILE",Q=N,ECHO}

The ! is a special wildcard character which is the same as the wildcard combination */*. This command would result in every file on drive :00 being killed. The disk password "MYFILE" will be used to access any file encountered which is password protected. The name of each file will be displayed as it is KILLed, and the user will not be prompted before a file is KILLed.

LIB

The LIB command will send a list of the DOSPLUS II library commands to the currently defined output device. This would normally be the video display; however any output device may be specified.

==========

The command syntax is:

LIB [TO] channel

"channel" is any valid output device in the system. If specified, it may not contain any wildmasks.

==========

The LIB command will display a list of the DOSPLUS II library commands, or send the list to a user-specified output channel, which may be any output device (for example, @PR) or filespec (for example, LIBRARY/COM:Q3). If "channel" is not specified, it will default to @DO, the video display.

DOSPLUS II distinguishes between library commands and programs. Library commands are routines which are within the operating system itself. These are the commands displayed with LIB. Library commands are given priority over programs; that is, if a program's filespec is the same as one of the library commands, the system will execute the library command rather than the program. The system is extended by adding programs which perform functions not covered by the library commands. These programs are not part of the operating system itself, and the system is not affected when they are killed. Conversely, library command routines cannot be easily removed from the operating system.

When specifying an output channel, wildmasks should not be used, and will be rejected. For example, the command LIB TO */LST would not be valid. When sending the list of commands to a file, drivespecs may or may not be specified. If omitted, the system will use the first available drive.

Examples:

LIB

This command will display a list of the library commands on the video screen. It is identical to LIB @DO.

LIB TO LIBLIST:I4

This command will send the listing of library commands into a file called LIBLIST on drive I4.

LIB @PR

This command will output the library command listing to the lineprinter.

LINK

This command will link together two devices within the DOSPLUS system. Once the link is established, any output going to either device will also be sent to the other. In the case of linked input devices, any input requested from one device may be supplied by either device. The LINK command by itself will display the current LINKed devices and their settings.

==========

The command syntax is:

LINK [FROM] devicespec [TO] channel

"devicespec" is the primary device (0-7 only) which is to be linked with another. "channel" is a device or file with which the primary device is to be linked.

Neither "devicespec" nor "channel" default to anything. If a channel is specified, then a devicespec must also be specified. If a devicespec is specified, then a channel must also be specified. If neither are specified, then the link settings for all devices, if any such settings exist, will be displayed on the video screen. The I/O direction of the linked devices must be the same, that is, input devices can only be linked to other input devices, and output devices can only be linked to other output devices. Linking an input device to an output device, and vice versa, is illegal.

==========

The LINK command allows simultaneous I/O from two devices in the system. If, for example, you wanted a hard copy of everything that appeared on your video display, you could link the @DO device to the @PR device. After the link is established, then everything going to the display will also be sent to the printer.

It is also possible to link an output device to a file, so that everything sent to that device will simultaneously be sent into a disk file. For example, linking @PR to a file will send all printer output into a disk file simultaneously.

Linking a device to itself (e.g., LINK @PR TO @PR) will RESET that device; that is, any previous linking established will be removed.

Restrictions: (1) Only devices 0-7 can be the primary device. These are the system devices @KI, @DO, @PR, @CA, and @CB, plus the three user-definable devices @U1, @U2 and @U3 (remember that these devices may be renamed; if they are, then the current name of the device is the one which should be used). Drives (devices 8-15) may NOT be specified, either as the primary device or the linked channel. Drivespecs are valid only with filenames.

(2) Input devices should only be linked to other input devices (or devices capable of simultaneous input and output) or channels and output devices may only be linked to other devices or channels capable of output. Linking an input device to an output channel, or vice versa, is possible but the results are not always predictable.

(3) The order in which devices are linked together is important, since the link is essentially in one direction only. For example, if @DO was linked to @PR any output sent to @DO would also appear on @PR, but any output sent to @PR would not appear on @DO.

(4) When linking an output device to a file, remember that the file will remain open until the device is reset and the link removed. If the computer is rebooted without resetting the device, the file may not be readable.

Examples:

LINK @PR TO @DO

This command will send all printer output to the video display simultaneously. However, display output will not be sent to the printer.

LINK @CB FILE1/TXT

This command will send all data coming from Communications channel B to a file called FILE1/TXT. If FILE1/TXT previously exists, then incoming data will overwrite the previous contents of the file. If FILE1/TXT does not previously exist, it will be created on the first available disk drive.

LINK @CA @KI
LINK @DO @CA

These two commands will link Communications channel A to the keyboard device as well as the video display. Since @CA is capable of both input and output, these two links are valid. After these two commands are given, any input that comes over the @CA device will be treated as keyboard input, and any output going to the display will be sent out the communications line. If the communications line A was set up correctly previously, these two commands will allow your computer to be controlled from a remote terminal. However, the keyboard remains active, so that any commands typed in at the keyboard will also be handled normally.

LINK @PR TO @PR

This command will RESET the @PR device. Any linking or routing (see below) which may have been active will be removed.

LINK

This command, with nothing given in the I/O field, will simply display a list of devices with their current LINK settings if any.

EXAMPLES OF ILLEGAL FORMS OF THE LINK COMMAND:

LINK FOO/BAR:00 TO @KI	(Only devices may be linked, not files)
LINK @DO :01	(Only devices 0-7 may be linked; disk drives are devices 8-15)

LIST

This command will list data from a device or disk file to a specified output device or channel. Non-printable ASCII codes will display as periods. Control codes (ASCII Codes 00H to 1FH) may either be displayed as periods, or sent unchanged to the output device/channel.

==========

The command syntax is:

LIST [FROM] channel [TO] channel {PARAM=switch}

"PARAM" is an optional parameter. "switch" is the switch value (Y or N) which may be supplied to PARAM.

The parameter for this command is:

CTL=switch	Determines whether or not control codes (ASCII 00H - 1FH) will be output unchanged or whether they will be displayed as periods (".")

CTL may be abbreviated to its first letter.

If CTL is not specified, NO is assumed, that is, ASCII codes less than 20H will be output as periods. If CTL is specified without a switch value, CTL=YES will be assumed.

The default output device is @DO, the video display. The input channel does not default and must be specified. Wildmask specifications may NOT be used in either the input or the output channel.

==========

The LIST command is normally used for listing a disk file to an output device such as the video display or the lineprinter. However, it may also be used to list data coming from other input devices such as the keyboard or the communications lines. Non-printable characters are listed as periods. However, control codes may be passed to the output channel without being translated to periods by specifying CTL=YES. Note that control codes may affect the action of the output channel. For example, a CTRL-Z (1AH) may cause the @DO device to switch to reverse video; similarly, a CTRL-L (0CH) may cause your lineprinter to execute an unexpected form feed to the next page.

Only the ASCII characters (or translated periods) are listed; their hexadecimal equivalents are not listed, unlike TRSDOS.

Examples:

LIST FOO/BAR

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

LIST FOO/BAR {C}

This command is identical to the first one except that now control codes (00H to 1FH) are output unchanged. All other characters will be listed as is. Depending on the control codes present in the file called FOO/BAR the display may or may not act strangely.

LIST FOO/BAR:P9 TO @PR

The file called FOO/BAR on drive :P9 will be listed on the lineprinter device.

LIST FROM @KI TO @PR {CTL=Y}

This command will echo keyboard input to the lineprinter device, in a fashion similar to the COPY command. Keyboard input will not be passed to the DOSPLUS II system for interpretation as commands. Control codes will be output unchanged (however, the lineprinter may act on certain codes, for example a form feed).

All characters typed in at the keyboard will continue to be sent to the lineprinter until BREAK is pressed.

LOAD

The LOAD command will take a disk file and load it into memory. The disk file is assumed to be either in program (/CMD) or core image (/CIM) format. The user may specify where in memory the file is to load. In either type of file, the user may also specify whether the program is to be run upon completion of loading.

The command syntax is:

LOAD [FROM] channel {PARAM=exp,PARAM=exp...}

"PARAM" is an optional parameter which may be specified with the command. "Exp" is an expression defining the value to be taken by the parameter (switch value or numeric value).

The parameters for the LOAD command are:

PROMPT=switch	Determines whether the system will prompt the user for disk mounts or not. This will permit loading of a file from a non-system disk in the system drive.
RUN=switch	Determines whether the file is to be executed upon completion of loading.
START=address	Determines the starting point in memory for loading a core image file.
TRA=address	If a core image file is loaded into memory, this parameter will determine what address control is to be transferred to if the file is to be executed. This will also override the transfer address in a /CMD file.

Each parameter may be abbreviated to its first letter.

The default values for the parameters are:

PROMPT	NO. Do not prompt for disk mounts.
RUN	NO. Return to DOSPLUS II after loading. Do not execute file.
START	NO.
TRA	NO.

The LOAD command will take a file from disk and load it into memory. If the file is a program file, that is, it is in executable format and has the /CMD extension, then the address at which it is to be loaded will be taken from the file itself (this address is saved when the program is written to disk). If the file is not in executable format, that is, it is a "core-image" file, then the START parameter must be specified.

A "core-image" file is any file that does not contain loader codes. Executable program files contain special codes which tell the system where in memory it is to load, and what its starting address is. A file which does not contain these codes is considered to be a "core-image" file. Such a file may consist of binary program instructions, ASCII text, or binary data. It is generally given the extension /CIM.

If a file is given without an extension,the LOAD command will assume the extension /CIM if START is specified, or the extension /CMD if not.

The {PROMPT=switch} parameter will allow you to load programs from other than a system diskette using the system drive. You will be prompted to mount the proper diskette in the drive. Pressing ENTER will cause the system to proceed with the load. If the program is to be executed, you will then be prompted again to re-mount the system disk in the drive before the program is executed.

The {RUN=switch} tells the system that you want the file to be executed after loading.

The {START=address} parameter informs the system where to start loading a core-image file. These files do not contain loader codes which tell the system where in memory they are to load, so the load address must be supplied by the user.

The {TRA=address} parameter tells the system where the entry point of a core-image file is, and is generally given with the RUN parameter. After the file has loaded into memory, control will be transferred to the address supplied with the TRA parameter. This parameter can also be used to override the normal entry point address of a /CMD file.

Since programs may also be saved as core-image files without loader codes, the LOAD command will also allow you to specify a transfer address if you wish to run such a file after loading. This address is specified by the TRA parameter.

There is a special format for program files which allow them only to be loaded, and not run, when their filespec is typed in at the DosPLUS II prompt (These file types are defined under TRSDOS). These are files which contain the proper loader codes, but end with an ASCII 03H. If you type in the name of such a file it will be loaded as though with the LOAD command, but will not execute. However you can force such files to execute by using the LOAD command with the RUN or TRA parameters if you know where the entry point of the program is.

Examples:

LOAD TEST/CMD:YA

This command will load the program file TEST/CMD from disk drive :YA into memory. The locations into which it loads will be determined from the special loader codes within the file itself. Control is passed back to DOSPLUS II after the file is loaded.

LOAD TEST/CMD:YA {RUN}

The file TEST/CMD is loaded into memory from drive :YA. As soon as the file is loaded, control is passed to it and it will begin executing. This is the same as typing TEST:YA.

LOAD FOOBAR/CMD:0 {P,R}

The program file FOOBAR/CMD is to be loaded from disk drive :0. The system will prompt the user to mount the correct disk containing FOOBAR/CMD in drive :0 before it begins the load. The user should insert the disk in drive :0 and press <ENTER>.
As soon as the file is loaded, you will be prompted to reinsert the system disk. Then FOOBAR/CMD will execute.

LOAD MEMTEST {START=7C00H}

MEMTEST/CIM will be loaded into memory starting at address
7C00H (31744 decimal). Control will return to DOSPLUS II upon completion of the load. Note that a default extension of /CIM is assumed by the LOAD command.

PAUSE

This command will pause execution until a key is pressed. It is generally used to temporarily halt the execution of DO files to permit the operator to perform some task, such as inserting required disks into the proper drives.

==========

The command syntax is:

PAUSE [message]

"message" is an optional string. It must fit on the command line.

==========

The PAUSE command provides a convenient way to temporarily halt execution of DOSPLUS II to give the operator a chance to perform some necessary task. It is generally used inside a DO file. The command may optionally be followed by any string of characters which the user wants displayed at the PAUSE. When the command is executed, the word "PAUSE" will be displayed followed by the string. Execution will then be suspended until the user presses any key on the keyboard. If the BREAK key is pressed, the the DO processing will terminate.

Note that if PAUSE is inside a DO file which is executed with the BREAK key disabled, pressing the BREAK key in response to PAUSE will have no effect.

Examples:

Suppose a DO file contains the following commands:

```
CLOCK ON
PAUSE Please insert diskette MGPDATA.
LOAD MGP/CIM {S=5500H}
MGP
```

When this DO file is executed, the real-time clock display will first be turned on. Then the PAUSE command will be executed, displaying the line:

PAUSE Please insert diskette MGPDATA.

At this point execution will be suspended. The user should then insert the proper diskette in a drive and press any key. As soon as he presses any key execution will continue with the next command.

If this DO file was executed with BREAK=NO, or if the BREAK key had been disabled with the SYSTEM command (see below) then all keys EXCEPT the BREAK key could be used to cancel the PAUSE condition. Pressing the BREAK key, however, would not cause execution to proceed.

PROT

This command allows you to change diskette information. A new disk name and date may be assigned, as well as new disk passwords. In addition, the protection status of files in the directory may be changed by assigning the disk master password to them. This command can also be used to erase unused directory entries.

==========

The command syntax is:

PROT drivespec {PARAM=exp,PARAM=exp ....}

"PARAM" is the optional parameter whose value is to be altered. "exp" is the value which will be assigned to "PARAM."

The parameters for the PROT command are as follows:

PW="string1"	Supplies the current disk master password to the system.
MPW="string2"	Supplies the new password to the system, if the password is to be changed.
NAME="diskname"	Specifies the new name for the diskette.
DATE="mm/dd/yy"	Specifies the new date for the diskette. This field can be entered in free format.
LOCK=switch	Determines whether the disk master password is to be assigned to, or removed from, all the files in the directory or not.
ACC=switch	If LOCK=Y, this will cause the disk master password to be assigned to the ACCESS password of all files. If LOCK=N, this will cause the ACCESS password of all files which have them to be removed.
UPD=switch	If LOCK=Y, this will cause the disk master password to be assigned to the UPDATE password of all files. If LOCK=N, the UPDATE password of all files which have them will be removed.
CLEAN=switch	Specifies whether unused slots in the directory are to be zeroed.

Each parameter may be abbreviated to its first letter.

The defaults for the parameters are as follows:

PW	None. The disk master password must be specified.
MPW	None. A new master password will not be assigned unless explicitly stated.
NAME	None.
DATE	None.
LOCK	None.
ACC	YES.
UPD	YES.
CLEAN	NO.

==========

The PROT command allows you to change diskette attributes which were assigned at FORMAT or BACKUP time. These attributes include the diskette name, date, and master password. In addition, you can use the PROT command to assign the diskette master password to all the files in the diskette directory, or, conversely, remove all passwords from user files (system files will not be affected).

The {PW="string"} parameter supplies the diskette's current master password to the system. The password is a string of valid characters enclosed in single or double quotes. Any alphabetic characters in the strings are evaluated in a case-independent fashion, that is, upper and lower case letters are treated equally. To use the PROT command, the diskette's master password must be specified using this parameter unless it is null or nonexistent.

The {MPW="string"} parameter assigns a new diskette master password. The password must consist of a string of valid characters enclosed in quotes. Either single or double quotes may be used.

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

The {DATE="mm/dd/yy"} parameter allows you to change the diskette date. Normally this date is assigned at FORMAT or BACKUP time, but you may change it using the PROT command. The date may actually be any string up to 8 characters in length which the user wishes to place in this field.

The {LOCK=switch} parameter affects the protection status of the files on the diskette. LOCK=Y assigns the disk's master password to the Access and Update passwords of all user files on the diskette (unless used with the ACC and UPD parameters, see below). Conversely, LOCK=N removes all Access and Update passwords from all user files on the diskette. System files (that is, files with a file protection level of 6) are not affected.

The {ACC=switch} and {UPD=switch} parameters are used in conjunction with LOCK to control the assignment or removal of file passwords. For example, if ACC=NO was specified in conjunction with LOCK=Y, then only the UPDATE password of each user file would have the diskette's master password assigned to it. The Access passwords would be left untouched. Similarly, if UPD=NO was specified together with LOCK=N, then only ACCESS passwords would be removed from user files.

The {CLEAN=switch} parameter will determine whether unused slots in the diskette directory will be zeroed out or not. Some unused directory slots may contain information pertaining to KILLed files. If the directory slots are zeroed out, then no trace of any killed files would remain, and consequently it would be impossible to attempt the recovery of any killed files.

Examples:

```
PROT :AA {PW="secret",MPW="CIA"}
```

This command will change the master password of the diskette in drive :AA from "secret" to "CIA". Note that the case-independent evaluation of alphabetic characters would have allowed you to specify PW="SECRET" or MPW="cia" and still obtain the same results.

```
PROT :5 {LOCK=N,UPD=N}
```

This command will result in the access passwords of all user files being removed. Update passwords, however, would not be touched.

```
PROT :X1 {N="Fiscyr83",D="01.01.83"}
```

The name of the diskette in drive :X1 would be changed from whatever it was originally to "FISCYR83", and the diskette date changed to 01.01.83.

```
PROT :K2 {N="New$disk",CLEAN}
```

The diskette in drive :K2 would be renamed to "New$disk" and all unused slots in its directory would be zeroed out.

```
PROT :XX {DATE="KEEPOUT"}
```

The DATE field may contain any string, not just the date.

RENAME

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

==========

The command syntax is:

RENAME [FROM] channel/drivespec1 [TO] channel/drivespec2

"channel/drivespec1" indicates the current logical name of a disk file, device, or disk drive. "channel/drivespec2" indicates the new logical name.

==========

The user may rename any device, file or disk drive under the DOSPLUS II system. Names must conform to the conventions described in the Operations section of this manual. Briefly, a device name consists of an @-character followed by one or two valid characters; a disk drive name consists of a : (colon) followed by one or two valid characters. A filespec consists of a one to eight character file name, and a one to three character extension preceded by a slash ("/"). A file's password cannot be changed by the RENAME command. However, if a file to be renamed has a password, it still must be entered in order for the RENAME to execute properly.

Duplicate device or file names are not allowed. Wildmask specifications may NOT be used with the RENAME command.

The logical device and/or disk drive names are entered into the system's device table. Thereafter that particular device should be referred to by the new logical name until it is again changed by the RENAME command. New filenames replace the old ones in the diskette directory. If the same filename exists on more than one diskette directory, only the first one is changed if no drivespec is specified.

Examples:

RENAME @KI TO @KB

This command renames the @KI device to @KB.

RENAME :0 :ME

This command renames disk drive :0 to :ME.

RENAME FOO/BAS TO FOOBAR/BAS

This command renames the file called FOO/BAS to FOOBAR/BAS.

RESET

The RESET command will restore a device or disk drive to its normal power-up state. It will cancel all LINKing and ROUTing. RESET may be used to reset a single device or drive or to reset all drives; however it cannot be used to cancel a NIL state.

==========

The command syntax is:

RESET
RESET [FROM] devicespec/drivespec

"devicespec/drivespec" is the current logical name of the device or disk drive to be RESET.

The RESET command has no parameters.

==========

RESET without any device or drive specification will perform a GLOBAL reset of all devices and disk drives. If a devicespec or drivespec is included on the command line, then only that device or drive will be reset. Any linking or routing of the device will be cancelled, and the device will be restored to its normal power-up setting. However, any active translation (that is, the device is FILTERed) will not be affected. Also, the current logical name of the device or drive will NOT be changed.

If a device was linked or routed to a disk file, RESET will close the disk file when the link or route is cancelled.

Examples:

RESET

This command will perform a global reset of all devices. Any devices which were linked or routed will be restored to their powerup condition. Disk files which were the target of linking or routing will be closed.

RESET @PR

This command will restore the @PR device to its normal condition if it had been linked or routed to another device. If no linking or routing had been done, this command would not have any effect.

ROUTE

This command allows you to change the I/O path of the system's devices from their default settings. You may reroute the I/O of a logical device to another device or to a disk file. ROUTE without any device or channel specifications will display the current device settings.

==========

The command syntax is:

ROUTE
ROUTE [FROM] devicespec [TO] channel

"Devicespec" is the logical name of the device which is to be rerouted. "Channel" is the target device or file for the route.

==========

The ROUTE command provides the DOSPLUS II user with the ability to redirect the I/O paths of the system's devices. This provides unparalleled operational flexibility with a minimum of effort. With this command, lineprinter output may be sent to the display, or display output sent to a disk file. This avoids the need to rewrite programs in the event that, for example, a lineprinter should fail; all lineprinter output could merely be rerouted to the display, or to a disk file for later printing.

Unlike the LINK command, which links two I/O devices together so that data goes to the two devices simultaneously, ROUTE actually forces data intended for one device to another device or to a disk file. ROUTE is mainly used for output devices.

If a device is routed to itself, then that device is RESET (see above). If ROUTE is entered without any device specification or channels, then the current device settings will be displayed.

Wildmasks may not be used when specifying a channel for the ROUTE.

Once a device has been ROUTEd, it may be restored to its original settings with the RESET command.

Examples:

ROUTE @DO TO @PR

This command will send all output normally going to the display to the lineprinter instead. Once this command is given, no new data will appear on the display screen.

ROUTE @PR PRINTFIL/TXT:3

All output to the lineprinter will be sent to a disk file called PRINTFIL/TXT on drive :3. If PRINTFIL/TXT previously exists, then any data going to the routed @PR device will OVERWRITE the contents of PRINTFIL/TXT. If PRINTFIL/TXT does not previously exist, it will be created on drive :3. The disk file will remain open until the routing is cancelled by means of the RESET command.

ROUTE @PR TO @PR

This will cancel any active routing or linking for the @PR device, in effect performing a RESET @PR.

ROUTE @U1 @CA

Any data output to user-defined device 1 will now be sent to Communications channel A (@CA) instead.

ROUTE @DO TO @CB

This command would send all data intended for the video display to the Communications channel B port. It may be used, for example, if a large screen monitor was connected to the computer via this port for viewing by an audience.

SCREEN

This command is used to output video data to another device or channel. When this command is issued, whatever is on the video display will be echoed to another channel. This will provide users with the ability to produce hard copy of their video display screens.

==========

The command syntax is:

SCREEN [TO] channel

"Channel" is the output device or file for the screen print. The default is @PR.

==========

The SCREEN command will take whatever is on the video display at the time it is issued and send it to an output channel. Normally, the default output channel is the lineprinter device, @PR. The user may specify other output channels, for example a communications line, or a disk file. While the SCREEN command is processing the video output, any other program operations will be suspended for a few seconds.

NOTE: This command may be generated directly by a CTRL-"-" (ctrl-dash). When this form is used, the screen contents will immediately be output to the @PR device. Since there is no means of setting an alternative output channel with this form of the command, you should perform any necessary LINKing or ROUTing of the @PR device before using it.

When Ctrl-dash is used, nothing will be echoed to the screen before the video contents are output to the printer. This may be preferable to using "SCREEN" in order to avoid destroying the display format.

This command provides you with a convenient way of maintaining copies of screen displays. For example, the SCREEN command can be embedded in BASIC programs, or executed from machine language programs to keep track of user input to particular prompts. It can also be placed at strategic points in user programs to maintain a log of the program's I/O operations.

Examples:

SCREEN

Everything that is on the video display will be sent to the @PR device. Any running program will be temporarily suspended until the operation is completed (that is, until all the screen data has been loaded into the @PR spool buffer -- NOT until the lineprinter has finished printing).

SCREEN TO SCRNFILE/DAT:P0

All characters currently on the video display will be sent to the file called SCRNFILE/DAT on drive :P0. If SCRNFILE/DAT does not previously exist, it will be created. If the file already exists, then the screen data will overwrite the previous contents of the file.

SCREEN @CA

Characters on the video display will be output to communications channel A.

SET

The SET command installs a driver program for a specified device. Driver programs may be installed to enable the system to use non-standard equipment. Other driver programs can also be established for the three user-definable devices in the system.

==========

The command syntax is:

SET [FROM] devicespec [TO] filespec
SET [FROM] devicespec [TO] devicespec
SET [FROM] devicespec2 [TO] devicespec

"Filespec" is the name of the driver program. "Devicespec" is the logical name of the device for which the driver program is intended. "Devicespec2" is any active device whose driver address will be copied into the DCB of "devicespec1".

==========

Filespec and devicespec must be specified. Wildmasks are not allowed. If a filespec is given without an extension, the system will supply a default extension of /DVR (driver).

Setting a device to itself is the same as performing a RESET of that device. A device previously KILLed (i.e., set to NIL) may be restored to an active state by setting it to itself.

New driver programs can be established to handle I/O to particular devices using the SET command. A driver program is a machine-language routine which processes the data intended for a physical device such as a lineprinter and does the actual sending of data to that device. Depending on the nature of the physical equipment, the driver program may also input data or control signals from it to govern its operation, as in the case of a lineprinter driver program. Such a program may stop sending data when it receives a "printer not ready" or "paper out" signal. The driver program, then, acts as the interface between a physical piece of equipment and the DOSPLUS II operating system.

Other driver programs may be established to control the action of INPUT devices such as the keyboard or a communications line. These driver programs may process incoming data from such devices.

The DOSPLUS II system comes with default drivers already established for the main system devices (@KI, @DO, @PR, @CA and @CB, as well as the disk drives). Generally, these default drivers will be suitable for a wide variety of situations. However, a situation may arise where the system's driver is inadequate, or incompatible with a particular piece of equipment. Such a situation may be when a lineprinter using non-standard symbols is connected to the system, or a modem which requires special signals is connected to one of the serial ports. In such a case, a new driver must be installed for the affected device. Such driver programs may be provided with the system, or they may be specially written for the user's purposes.

Driver programs are stored on disk along with other files and programs, and are loaded by the SET command. They are given the extension /DVR to distinguish them from other files, but the user may give them any name and extension he chooses. However, if a driver program file without an extension is specified in the SET command, DOSPLUS II will append the default extension of /DVR to it.

Driver programs other than the system's default drivers normally load into high memory, and HIMEM is adjusted downward to protect them from being wiped out by other programs.

When a new driver is installed for a device by means of the SET command, any previous linking or routing for that device is RESET. However, any filtering or translation established for the device will not be affected.

If a second device is to be set to the same special driver, using the normal SET command form will load a second copy of that driver from disk, thus wasting memory space. However, there is a way around this. For example, if disk drive :02 was SET to a special driver called 5INCH/DVR, and you wish to set drive :01 to the same driver, instead of saying SET 5INCH TO :01 (which would load a second copy of the driver) you may say SET :02 TO :01. This command will copy the driver address currently being used by :02 into the DCB of :01. The net result is that both :01 and :02 will now share the same copy of the driver program. After executing this form of the SET command, however, you must CONFIG the second device to correctly reflect any new characteristics provided by the driver routine (see CONFIG).

SETting a device to itself will restore it to its normal power-up condition. This form of the SET command is also used to restore devices which have been killed or set to NIL. Using this form of the command for logical disk drives 4 through 7 will activate the DOSPLUS II hard disk drivers, which support the Radio Shack hard drive and the QCS hard disk. Logical drives 4 through 7 will be assigned to the hard drive(s). However, once the hard disk drivers have been set up, the drive assignments can be changed to suit your needs.

Examples:

```
SET @PR TO DWII/DVR
```

The driver program called DWII/DVR will be loaded from disk and installed for the @PR device. Any previous linking or routing established for @PR will be cancelled. An equivalent form of this command would be SET DWII TO @PR (the /DVR extension would be supplied by the system).

```
SET @CA MYMODEM/NEW:KZ
```

MYMODEM/NEW will be loaded from disk drive :KZ and installed for the Communications channel A device.

SET FROM @KI TO DVORAK/DVR

A new driver for the keyboard device @KI will be installed using the driver program called DVORAK/DVR.

SET :01 TO :02

This will copy the driver address being used by disk drive :02 into the Device Control Block of disk drive :01 so that both disk drives will now share the same copy of the driver program.

SET @CB @CB

If @CB was previously KILLed, this command will restore it to an active condition.

PARAMETER PASSING WITH THE SET COMMAND

Certain specialized device drivers may require the user to provide optional parameters at installation time. What these parameters are will depend on the driver program itself. For example, a specialized driver program for a smart modem connected to communications channel A (@CA) might require the user to provide a telephone number for its autodial function. This could be passed to the driver as a parameter (e.g., SET SMART20 TO @CA {DIAL="555-4746"}).

The syntax for passing parameters with the SET command is identical to the other library commands. Parameters must be separated from the I/O field either by a comma or a left brace, and terminated either by a carriage return, an implied carriage return (that is, a ";"), or a right brace. It is up to the driver program to retrieve the parameters and interpret them.

SETCOM

The SETCOM command configures the serial ports of the Model II. Either channel A or channel B may be configured using this command. SETCOM without any devicespec or parameters will display the current settings of the serial devices.

==========

The command syntax is:

SETCOM
SETCOM [FROM] devicespec {PARAM=exp,PARAM=exp...}

"Devicespec" is the logical device name of the serial port. "PARAM" is the parameter to be set or altered; "exp" is the value (numeric or string) which PARAM is to be set to.

The parameters for the SETCOM library command are:

BAUD=value	Sets baud rate of serial device
WORD=value	Sets word length (number of bits per word)
PARITY="string"	Determines type of parity to be used by the SIO.
STOP=value	Determines the number of stop bits to be used by the SIO.
BS=value	Determines size of the serial device's spool buffer in bytes.
EMPTY=switch	discards conents of spool buffer.

All parameters except BS may be abbreviated to their first character.

The defaults for the above parameters are as follows:

BAUD	300
WORD	8 bits
PARITY	None (no parity)
STOP	1 stop bit
BS	Previous buffer size
EMPTY	NO

SETCOM without any devicespec or parameters will display the current settings for the system's two serial devices.

==========

SETCOM is used to initialize or change the values of various parameters of the Model II computer's two serial I/O ports. These ports are devices @CA and @CB (Communications channels A and B) in DOSPLUS II. They are generally used either for communications with remote terminals, or for driving a serial printer. I/O through the ports are controlled by a SIO chip. The SIO transmits and receives data in a serial fashion, that is, bit by bit. It can be set to transmit and receive data in a variety of formats. The speed of transmission, number of bits per word, type of parity, and number of stop bits all go into making up a data format. When communicating with another terminal or serial device, the data format used by the Model II must be one that the remote device can understand.

There are a number of standard formats which are widely used for serial communications, and which a wide variety of serial devices can accept. However, there are also certain serial devices, such as special printers, which can accept non-standard formats. The SETCOM command allows you to set your computer's SIO to conform to whatever peripheral device may be hooked up to the serial port.

The {BAUD=value} parameter determines the speed of data transmission through the port. BAUD is a term meaning "bits per second." The baud rates to which the serial port's SIO may be set to are: 110, 150, 300, 600, 1200, 2400, 4800, and 9600 bits per second. The default for the SETCOM command is 300 baud, which is a standard rate for communications over telephone lines.

The {WORD=value} parameter sets the number of bits in a data "word" and can be from 5 bits to 8 bits. Seven or 8 bits are generally used for communications, since they allow the full ASCII character set to be transmitted. Word lengths of less than 7 are generally used for sending control values to peripheral equipment rather than for communications. The default for the SETCOM command is 8 bits, which permits the values 0 through 255 (00H through FFH) to be transmitted.

The {PARITY="string"} parameter determines the type of parity that will be signaled by an extra bit transmitted with each data word. The parameter may take the values ODD, EVEN, or NONE, and defaults to NONE (i.e., the parity bit is ignored). If ODD parity is specified, the parity bit would be set to a 1 if the the total number of bits in the data word set to 1 is an even number. When EVEN parity is set the parity bit of a word will be set when the data word has an odd number of bits set to 1. Parity is used by the SIO to check for transmission errors and generally need not concern the user.

The {STOP=value} parameter determines the number of stop bits that will be used by the SIO. It may take the values 1 or 2.

In asynchronous serial communications, each word (from 5 to 8 bits) is framed by start and stop elements which the SIO uses to synchronize to the start of data elements. The start element is normally a bit set to logic 0 added to the start of each word. Following the word are 1 or 2 stop bits set to logic 1. The transition from the stop element to the start element (logic 1 to logic 0) signals the start of the next data word.

The number of stop bits used by the SIO will depend on the requirements of the peripheral equipment or the remote system being communicated with.

The {BS=value} parameter allows you to set the size of the spool buffer for the serial I/O devices. Normally, the buffer size is 2 bytes in length. But if you wish to set up a larger buffer, you can use this parameter to specify the buffer size you want. If a buffer size larger than 2 bytes is specified, it will be established in high memory, and the HIMEM pointer will automatically be adjusted so that other programs will not collide with it.

IMPORTANT: The buffer space actually allocated for either of the communications line channels is TWICE value specified in the BS parameter. This is because each data byte to be transmitted over the serial ports has an associated status byte which must also be spooled. Therefore if you say BS=400, the system will actually allocate 800 bytes in memory to hold both the data and status bytes.

The {EMPTY=switch} parameter permits you to reset the device's spool buffer at any time, rather than waiting for it to be emptied normally. The action is identical to the EMPTY parameter of the FORMS command (see above).

Examples:

```
SETCOM @CA {BAUD=300,WORD=7,BS=512}
```

Communications channel A is set up for 300 baud transmission using 7-bit words, 1 stop bit (default) and no parity (default). A 1024-byte spool buffer is established for the device in high memory (512 bytes for the data bytes, and another 512 bytes for the status bytes).

```
SETCOM @CB {B=1200,W=7,P="EVEN",S=1}
```

Communications channel B is initialized for 1200 baud, 7-bit words with even parity and one stop bit.

```
SETCOM @CA
```

Channel A is set to the default conditions of 300 baud, 8 bit words, no parity, and one stop bit.

```
SETCOM
```

The current settings of the two serial ports will be displayed.

SYSTEM

This command allows you to configure certain aspects of the DOSPLUS II system to your requirements. Once the configuration has been established, it may be saved on disk as a program file. Any number of configurations may be saved. When a particular one is required, all that is needed is to type the name of that file at the DosPLUS II prompt to establish it in the system.

The command syntax is:

SYSTEM
SYSTEM {PARAM=exp,PARAM=exp ...}

"PARAM" is the parameter to be set or altered. "Exp" is the value (switch value, numeric or string) to be set for a given parameter.

The parameters for the SYSTEM command are:

BREAK=switch	Enable/disable BREAK key recognition.
LOMEM=value	Set address for LOMEM pointer.
HIMEM=value	Set address for HIMEM pointer.
ALIVE=switch	Turn ALIVE character on/off.
TRACE=switch	Turn program counter trace display on/off.
SAVE="filespec"	Save the current configuration to disk as "filespec".
DATE=switch	Enable/disable date prompt on bootup
TIME=switch	Enable/disable time prompt on bootup.
LOGO=switch	Enable/disable display of the DOSPLUS II logo on bootup.

All parameters except TRACE and LOGO may be abbreviated to their first character. TRACE and LOGO cannot be abbreviated. The parameters do not default. If not explicitly declared in the command, the previous value for that parameter will remain in force.

The SYSTEM command permits the user to customize his DOSPLUS II configuration according to his needs. This command can alter certain conditions within the system but its real power is in the "SAVE" parameter. Use of the SAVE parameter allows you to store entire configurations on disk. Saved configurations include not just the conditions settable by the SYSTEM command, but also all Device Control Blocks, the drive code table, and any driver routines or buffers located in high memory (above HIMEM). All LINKing and ROUTing, along with the current logical device names, are also preserved by the SAVE parameter. In addition, any commands given on the same line (using the multiple command syntax) will be preserved.

When you wish to restore the configuration, you simply type in the name of the file it was saved under, as though you were running a program. The configuration will be loaded and restored. Any commands that followed the original SYSTEM {SAVE="FILESPEC"} command on the same line will be executed. Thus it is possible to save any number of configurations on disk, and call them up as required. This means that each configuration needs to be built only once.

The SYSTEM command given without any parameters will display the current settings of HIMEM (top of free memory), LOMEM (bottom of free memory) and PHYMEM (top of physical memory).

The {BREAK=switch} parameter determines whether the system will respond to the BREAK key or not. When BREAK=NO, DOSPLUS II totally ignores the BREAK key. This condition will also apply to application programs (such as BASIC) which use the operating system's routines to detect the BREAK key. Such a condition may be desirable if you do not wish users to interrupt running programs by hitting the BREAK key. This condition will remain until BREAK recognition is reset by a SYSTEM {BREAK=Y} command, a DO command with BREAK=Y, or a reboot of the system.

The {LOMEM=address} parameter allows you to set a memory pointer called LOMEM which defines the lowest address of FREE MEMORY in the system. Properly written application programs which check this pointer will avoid using memory below that pointed to by LOMEM. LOMEM cannot be given an address less than 2800H. This is the region occupied by the resident DOSPLUS II system and is restricted.

The {HIMEM=address} parameter will allow you to set another memory pointer, called HIMEM. This defines the highest free memory address in the system. Properly written application programs should not use memory above that pointed to by HIMEM. When DOSPLUS II creates a buffer or loads a routine into high memory, it will always use memory below HIMEM and then adjust HIMEM downward to protect the areas now in use.

Due to the fact that DOSPLUS II will use high memory for certain purposes, you will not be able to use the SYSTEM {HIMEM=address} command to specify an address GREATER than its current value. SYSTEM {HIMEM=address} can only be used to adjust HIMEM downward. This restriction ensures that any areas of high memory being used by DOSPLUS II will not be inadvertently destroyed. Also, the HIMEM pointer cannot be less than the current value of LOMEM.

The {ALIVE=switch} parameter will turn the "alive" character on or off. This is a moving graphics character in the upper right corner of the display screen. Its movement is determined by a background task routine (that is, a routine normally "transparent" to the user, similar to the routine which updates the clock display), and helps determine whether or not the interrupt task processor is working properly. Used in conjunction with the TRACE parameter, it can help an experienced systems programmer diagnose a malfunction in a running program.

The {TRACE=switch} parameter turns the trace display in the upper right corner of the screen on or off. This displays the current value of the Z-80 CPU's program counter. Like the ALIVE and the CLOCK display, this is executed as a background task by the interrupt processing routines of DOSPLUS II. This display can be used by an experienced programmer to diagnose any problems which may appear in a running program.

The {SAVE="filespec"} parameter tells DOSPLUS II to save the current configuration to disk under the specified "filespec." The configuration will include the current settings of the parameters under the SYSTEM command (BREAK key recognition, LOMEM and HIMEM values, etc.) along with all Device Control Blocks (DCBs) in their current condition, the Drive Code Table (DCT), and any routines or buffers residing in high memory (the memory region above the HIMEM address). All LINKing and ROUTing of devices will be preserved, along with their current driver routines, if any. Logical device names will also be preserved.

The filespec under which the configuration is to be saved should be given the extension /CMD. This will permit rapid re-installation of the configuration at some future time by simply typing in the name of the file from the DosPLUS II prompt.

You should be extremely careful NOT to save any configuration when a disk file is in an OPEN condition, as when a device is ROUTed to a disk file. The reason for this is that there is no guarantee that the actual disk space that was being used by the routed device will STILL be available when the configuration is reloaded at some future date. This could easily result in the destruction of valuable data on the disk.

Also, when reloading a configuration file from disk, any commands which follow the terminator (the ";") in a multiple-command line will not be executed, since the keyboard buffer will be totally replaced by that saved in the file. That is, if CONFIG1/CMD is a saved configuration file, then the command

CONFIG1;DIR;RESET;ROUTE @DO @PR

will go only so far as loading the CONFIG1 file; any following commands will not be processed since, as soon as CONFIG1 loads, the command line buffer will already have been replaced by the one in the CONFIG1 file. To execute a string of commands upon loading a configuration file, the commands must be saved in the file itself. For example, if you wish to execute the commands RESET, DIR and ROUTE @DO @PR immediately after loading a configuration file, you would use this command line:

SYSTEM {SAVE="CONFIG1"};RESET;DIR;ROUTE @DO @PR

When the file CONFIG1 is now loaded from disk, the following commands RESET, DIR and ROUTE @DO @PR will be executed in sequence.

The .{DATE=switch} and {TIME=switch} parameters will turn the date and time prompts at bootup on or off. When these parameters are set to ON, you will be asked to supply TIME and/or DATE whenever the system is booted. Pressing ENTER will cause the registers for that parameter to be zeroed out. Pressing BREAK in response to the prompt will cause the system to determine whether the time or date registers contain valid data. If valid data is found, it will be retained. Otherwise the registers will be zeroed out. Note that time and date are preserved through a reset of the entire system, although the time may be off by several seconds after a reboot.

The {LOGO=switch} parameter will turn the DOSPLUS II logo at bootup time on or off.

The condition of the TIME, DATE and LOGO parameters are saved onto the system disk independently of the rest of the configuration. It is not necessary to do a SYSTEM {SAVE="filespec"} to preserve the settings of these three parameters.

Examples:

SYSTEM {BREAK=NO}

This command disables recognition of the BREAK key. The BREAK key will be completely ignored by DOSPLUS II until a SYSTEM {BREAK=ON} is issued, a DO file is executed with BREAK=Y, or the system is rebooted.

SYSTEM {HIMEM=EF00H,TRACE=ON,SAVE="CONFIG1/CMD"}

HIMEM is set to EF00H (61184 decimal). The trace display is turned on, and the system's current configuration is written out into a disk file called "CONFIG1/CMD."

SYSTEM {LOGO=ON,TIME=NO,DATE=Y}

The LOGO display and date prompt at bootup time are both enabled. The time prompt is disabled. The settings of these three parameters will be written out to the disk independently of a SYSTEM {SAVE=...}.

SYSTEM {T=OFF,TRACE=ON,A=ON}

The time prompt at bootup is disabled. The TRACE and ALIVE prompts are turned on. Note that TRACE cannot be abbreviated.

TIME

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

==========

The command syntax is:

```
TIME
TIME hh:mm:ss
```

The TIME command by itself will display the current time in the system's real time clock in the format HH:MM:SS.

==========

When setting the clock with the TIME command, the time can be specified in a variety of ways. Allowable separators are colons (:), commas (,), dashes (-), slashes (/), periods (.) and spaces. This flexibility allows you to specify the time in whatever format is most comfortable to you.

The time is maintained by the system in 24-hour format. That is, the hours go from 0 to 23. Midnight is 00:00:00, and one p.m. is 13:00:00.

Examples:

```
TIME 3:5:30
TIME 03:05:30
TIME 3-05-30
TIME 03 5.30
TIME 3.05/30
```

All of the above are equivalent and set the system's clock to 03:05:30.

```
TIME 9.00
TIME 9
```

The system clock is set to nine o'clock. If minutes and seconds are not specified, they default to 00.

```
TIME
```

DOSPLUS II will print the current time on the video screen.

VERIFY

The VERIFY command causes DOSPLUS II to read back whatever is written onto the disk in order to verify that it was written correctly.

==========

The command syntax is:

VERIFY switch

"Switch" is either ON (Yes) or OFF (No).

==========

This command will enable automatic read-after-write on all disk I/O. It will ensure that data written to the disk can be read back without error. This will slow disk I/O down slightly but might be desirable when writing critical data to the disk.

Examples:

```
VERIFY
VERIFY YES
VERIFY Y
VERIFY ON
```

These forms of the VERIFY command are all equivalent and enable the read-after-write function. If VERIFY is already on, then these commands will have no effect.

```
VERIFY OFF
VERIFY N
VERIFY NO
```

These forms of the VERIFY command will turn off the read-after-write function. If the function was already disabled, then these commands would have no effect.

UTILITIES

DOSPLUS II Utility programs

The following utility programs are included with your DOSPLUS II. They will be executed from the DOS command mode by entering the name of the utility program and pressing ENTER. Some of them have additional parameters that may be input from the DOS command line. Consult the individual program documentation for the filenames and additional parameters.

BACKUP	(Duplicate a floppy diskette)
CONV	(Copy files to/from TRSDOS disks)
DIRCHECK	(Directory check utility)
DIRFIX	(Directory repair utility)
DISKZAP	(Sector orientated disk editor)
DRAW	(Graphics editor program)
FORMAT	(Floppy disk formatter - includes RFORMAT)
HELP	(Quick reference utility)
MAP	(Display which sectors a file resides in)
OFFSET	(Relocate file's location in memory)
PATCH	(Install patches into disk files)
SVCINT	(TRSDOS SVC simulator)
SYSGEN	(Install system files on non-standard media)

BACKUP

This program is used to copy all data from one floppy disk to another. It will make exact or "mirror-image" copies only. To use a "copy by file" method to backup your disk, use the library command COPY. You will also use COPY to backup the hard disk.

==========

The command syntax is :

BACKUP [FROM] :sd [TO] :dd {param=exp...}

":sd" is the drive that you will be copying FROM. If this information is not provided in the command line, BACKUP will prompt you for it later.

":dd" is the drive that you will be copying TO. As above, if this is not specified in the command line, it will be prompted for.

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

"exp" is the optional expression that will supply any additional input needed by the parameter.

Your parameters are :

DATE=string	Allows you to set the date directly from the command line when you are aware that the system date is NOT set and you do not wish to be prompted.
USE=switch	Allows you to indicate that you wish to over-write any existing format on the destination disk WITHOUT being prompted during the backup.

Abbreviations :

DATE	D
USE	U

==========

The backup utility enables you to backup your floppy diskettes. It is recommended as a good computing practice to use this utility to make frequent copies of your important data diskettes.

With DOSPLUS II's BACKUP utility, it is not necessary to pre-format your destination diskettes. If the diskette is blank, DOSPLUS II will format it automatically. Even if the diskette was previously formatted, DOSPLUS II will offer you the chance to format it again before using it in the backup.

BACKUP allows you to optionally specify the source and destination drive from the command line using the syntax shown above. You, of course, do not need the FROM and TO delimiters unless you are specifying the destination drive first or only. BACKUP will assume that the first drivespec encountered is the source drive unless it finds a TO delimiter. It will likewise assume the second drivespec encountered to be the destination drive unless it encounters a FROM delimiter.

If you do not specify the source and destination drives at the command line, BACKUP will prompt you for them. If you specify one without the other, BACKUP will prompt for the one that is missing.

Also, in addition to specifying the source and destination drives from the command line, you have the option of specifying the date and whether or not you wish to use the disk if it contains data.

To set the date, all you must do is use the statement "DATE=string", where "string" is a quoted string up to eight characters in length. When inputting the backup date, either with this line or in response to the prompt, you are not limited to numeric input.

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

You may, if you wish, operate BACKUP from within a DO file. This can allow you to use BACKUP as a menu option from a BASIC program and then return to the menu. The procedure is :

(1) Have the first statement of the DO file exit BASIC and return to DOS.

(2) Execute the BACKUP.

(3) Have the DO file re-load BASIC and run your menu.

This is made simpler by virtue of the fact that you can specify all information needed for BACKUP right from a command line. True "hands off" operation. The computer operator doesn't even need to respond to a "Diskette contains data" prompt. Prompting messages -

Source drivespec ?

Reply to this question with the drivespec of the drive that contains the disk you wish to backup. Do not include the colon (":"). It is only necessary to provide BACKUP with the one or two character drive name.

Destination drivespec ?

Reply to this question with the drivespec of the drive that contains the disk you wish to backup to. As above, you do not need to include the colon. This drivespec may be the same as the source drivespec if you wish to execute a single drive backup.

Backup date (MM/DD/YY) ?

If the system date has not been set and you have not entered it from the command line, BACKUP will prompt you for the date. When it does, you have three options :

(1) Press BREAK and abort the backup.

(2) Press ENTER and default to a date of "00/00/00".

(3) Type in up to any eight ASCII characters you wish for the date and press ENTER. You are not restricted to numeric characters.

If the diskette is not blank and you have not specified the "Use" parameter from the command line, you will receive the prompt :

Diskette contains data, Use or not ?

You may reply in one of three ways to this prompt :

(1) Press BREAK and abort the backup.

(2) Type "Y" or "U" and press ENTER. This will cause BACKUP to attempt to use the existing format.

(3) Type "F" and press ENTER. This will cause BACKUP to re-format the destination disk first.

Once all of these questions have been answered, BACKUP will proceed with the copy of the disk. The destination disk will bear the same name and Disk Master Password as the source disk. The date on the destination disk will be either the current system date or whatever characters you entered when prompted.

Single drive vs. Multiple drive -

If the source and destination drivespec are not the same (in other words, you are backing up between two separate disk drives), BACKUP will proceed with the copy after all information has been provided with no further operator intervention.

If, on the other hand, the source and destination drivespecs are identical (in other words, a single drive backup), BACKUP will procced with the copy but will prompt you for the source, destination, and system disks as they are needed.

Pay close attention to these prompts and insert the proper diskette. If you were to accidentally insert the wrong disk at the wrong time, you could corrupt the data.

Examples:

```
BACKUP
```

This will execute the backup program and have it prompt for all information.

```
BACKUP FROM :0 TO :1
BACKUP TO :1 FROM :0
BACKUP :0 :1
```

These three examples are all equivalent. They instruct the BACKUP program to backup the disk in Drive 0 to the disk in Drive 1. Note that if you ARE going to use the drive specifiers from the command line, you will need to hae both disks (source and destination) in place before executing BACKUP.

```
BACKUP FROM :0 TO :1 {DATE="Sept 24",USE}
BACKUP :0 :1 {D="Sept 24",U}
BACKUP :0 :1,D='Sept 24',U
```

All of these commands will acoomplish the same results. They will backup the disk from Drive 0 to the disk in Drive 1. They will set the backup date to "Sept 24" (note the use of non-numeric characters) and instruct BACKUP to use the destination disk even if it contains data.

Finally:

BACKUP will not backup between two disks of dissimilar format. For example, you can't backup a single sided to disk to a double sided one or vice versa. For those applications, you should use the COPY command to perform a "copy by file" type of backup. BACKUP also will not backup between rigid and floppy disk drives.

As BACKUP is making the backup, it will ONLY copy those cylinders that have allocated data on them and it will ONLY copy as much data from each cylinder as it contains. Do not be alarmed if you see BACKUP skip several cylinders or it BACKUP seems to copy some cylinders faster than others. If you wish to verify, make note of the cylinders at which this occurs. Then use the library command FREE to display a free space "map" of that disk. The cylinders skipped should show no "x"s at all and cylinders that seemed to backup faster than others should have open space (i.e. not solid "x"s). (See the library command FREE)

BACKUP attempts to make "mirror-image" copies of the source disk. If it cannot for any reason do this (a granule allocated on the source disk is locked out on the destination), BACKUP will report an error and abort to the DOS command mode. You may at that time either re-format the destination disk and try again or resort to a "copy by file" backup.

One important note. After the "Diskette contains data" prompt is on the screen, you may NOT switch the source disk. This will cause incorrect information to be written to the destination disk that will later corrupt data. You may switch the destination disk at that time, if you wish.

Obviously, if you are going to invoke BACKUP with all questions answered from the command line, you had better have the disks to be backed up all mounted and ready. This is doubly true if you have specifed the "Use" parameter.

As a rule of thumb, if you are going to backup two disks that are not currently mounted and ready to go it is best to just type "BACKUP" and allow the program to load and ask you all needed questions. Once the program is loaded, you may remove all disks and proceed. It will tell you when it needs a system disk again.

Although BACKUP is a great deal faster than the same program in TRSDOS, we want to inform you that it IS doing a full verify of all copied data. There is no loss of accuracy, simply a great gain in speed.

CONV

The CONV utility provides a convenient means of passing files from a TRSDOS disk to a DOSPLUS II disk and vice versa. CONV also includes facilities for viewing the directory of a TRSDOS disk, which would normally be unreadable from DOSPLUS II.

The command syntax for invoking CONV is:

```
CONV [FROM] :DR [TO] :DR [USING] wildmask {param,param...}
CONV wildmask {param,param...}
```

where channel is a filespec or drivespec, wildmask is a file wildmask, and param is an optional parameter.

The parameters for the CONV utility are:

QUERY	Query the user before moving a file
OVER	Query the user if the file already exists on the target disk whether or not to overwrite it.
BACK	Move file from DOSPLUS II to TRSDOS
DIR	Display directory of TRSDOS disk
CAT	Same as DIR
ECHO	Display filenames as they are moved
SYS	The file to be moved from a TRSDOS disk has the TRSDOS "system file" bit on. Treat the file as a system file.
PW="password"	Declares Disk Master Password to CONV for use with password protected files.

All parameters can be abbreviated to their first letter.

Since DOSPLUS II disks are formatted differently from TRSDOS disks, one cannot be directly read from the other. The CONV utility provides a means whereby files can be moved from one system to the other with a minimum of effort.

CONV will copy over a single file, a class of files, or all of the user files (visible/invisible) on a disk. DOSPLUS II system files may NOT be moved over to a TRSDOS disk; however TRSDOS system files can be moved over to DOSPLUS II. When copying program files from TRSDOS to DOSPLUS II, CONV will append the /CMD extension to the file. When copying program files from DOSPLUS II to TRSDOS, the /CMD extension will be stripped, and the program bit in the TRSDOS directory entry will be set for that file.

Technical note : CONV will recognize TRSDOS 1.2 and TRSDOS 2.0 disks. It does NOT support TRSDOS 4.0 disks or hard drive systems. If you wish to move files from TRSDOS 4.0 to DOSPLUS II, you must first use the TRSDOS FCOPY utility to place the files on a floppy disk in TRSDOS 2.0 format, and then transfer them to DOSPLUS II using the CONV utility.

TRSDOS passwords are ignored, but DOSPLUS passwords are not. If an affected DOSPLUS II file is password protected, the disk master password must be declared to CONV with the PW parameter. TRSDOS passwords are ignored by CONV when moving files.

If a TRSDOS password to be moved to DOSPLUS II is password protected, the passwords and protection status will be transferred as well, and the file will be protected on the DOSPLUS II disk. However, DOSPLUS files which are password protected will have their passwords stripped when transferring them to TRSDOS disks. The reason for this is that DOSPLUS encodes passwords in a different manner in the directory entry.

The {QUERY} parameter will cause CONV to display the name of the file being moved and ask the user whether to go ahead or not. If you reply with a "Y", then the transfer will proceed; otherwise it will be cancelled. This is useful when doing a group transfer using a wildmask, and only some files of those which fit the wildmask are actually to be moved.

The {OVER} parameter will force CONV to pause if the file already exists on the destination disk, and query the user whether that file is to be overwritten or not. If the user replies with a "Y", then the transfer will take place. Otherwise it will be cancelled.

The {BACK} parameter is used to indicate a DOSPLUS II-to-TRSDOS transfer. Normally, CONV assumes that the direction of transfer is from a TRSDOS disk to a DOSPLUS II disk. This will force the reverse.

The {DIR} and {CAT} parameters will display a directory listing of the files on a TRSDOS disk. Only the filenames will be displayed.

The {ECHO} parameter will display the name of each file as it is copied over. This is particularly useful when performing a group transfer using a wildmask. Note that the QUERY parameter, if present, will override ECHO.

The {SYS} parameter is used when transferring TRSDOS "system" files over to DOSPLUS II. TRSDOS identifies system files by setting a bit in the directory entry. Normally such files would be ignored by CONV, but this will force it to execute the transfer. This parameter is needed to transfer BASIC from TRSDOS to DOSPLUS, since TRSDOS identifies BASIC as a system file.

Examples:

```
CONV :1 :AE,Q,OVER
```

All the user files on drive :1 will be transferred to drive :AE. You will be prompted with the name of each file, once to approve the transfer, and then again if a file by that name already exists on drive :AE.

```
CONV */TXT:1 :02 {ECHO,BACK,PW="M02B1"}
```

Files with the extension /TXT will be moved from the DOSPLUS disk in drive :1 to the TRSDOS disk in drive :02. Each file's name will be displayed as it is moved. If any of the files on the DOSPLUS disk are password protected, the passwords will be overridden by the disk master password "M02B1." Such files will have their passwords stripped as they are moved to the TRSDOS disk.

```
CONV BASIC:4 :S0 {SYS,O,PW="Mydisk"}
```

BASIC will be moved from the TRSDOS disk in drive :4 to the DOSPLUS disk in drive :S0. Since TRSDOS considers BASIC to be a system file, it is necessary to use the SYS parameter. The user will be prompted if a file called BASIC/CMD already exists on drive :S0. If such a file does exist and the user orders CONV to proceed, the disk master password "Mydisk" will override any password protection it might have.

DIRCHECK

DIRCHECK is a utility program for testing the integrity of a DOSPLUS II disk directory. It can be used to check a single directory entry or the entire directory for errors.

==========

The command syntax for invoking DIRCHECK is:

DIRCHECK filespec
DIRCHECK :dr

where filespec is the file whose directory entry is to be checked, and :dr is the drivespec whose directory is to be checked.

==========

DIRCHECK will scan a disk directory for errors which may result in the possible destruction of valuable data unless repaired. It will inform you of any errors it finds, and its companion utility, DIRFIX, can be used to repair the errors.

A DOSPLUS II disk directory can be thought of as being divided into three sections: the Granule Allocation Table (GAT), which resides on the first sector of the directory track, the Hash Index Table (HIT), which resides on the second sector of the directory track, and the file records, which make up the remainder of the directory. All three sections relate to each other very closely, and an error in one can propagate to the others.

The GAT can be thought of as a "map" of the disk, showing which granules are allocated to files, which are available for use, and which are locked out for some other reason. When creating a new file on a disk, DOSPLUS II uses this table to determine where it can be placed. An error in this table may result in the same sectors on the disk being assigned to more than one file, with potentially disastrous results.

The HIT, or Hash Index Table, may be thought of as a map of the directory itself. In this table are kept one-byte codes (called directory entry codes, or DECs) of each file in the directory, positioned in such a fashion as to pinpoint a file's directory entry in the succeeding sectors. Thus the system can access a file's directory entry in a minimum of time and with a minimum of disk accesses. All files have at least one DEC. If a file is assigned more than four extents, an extended directory entry is created for it, with a corresponding DEC.

Errors in the HIT table may consist of DECs which point to empty directory records or records containing inactive files, or DECs which do not correspond to the file entry in the corresponding position (the encoded value, which is derived from the filespec, is incorrect or zero). Such errors can also be potentially dangerous and may result in the system accessing the wrong file, or simply failing to find a file.

The directory records maintained in the remainder of the directory track contain information about each of the files on the diskette. Among other things, their status (active/killed), the file name, encoded passwords if any, and the number and length of each extent are coded for each entry. Errors here can consist of invalid characters in file names, an end-of-file pointer specifying a position beyond the last allocated sector for the file, an incorrectly linked extended directory entry, etc.

DIRCHECK will test for these types of errors in a diskette directory and list any errors found on the screen. If errors exist, they can be repaired using the DIRFIX utility.

When DIRCHECK is typed with a filespec, for example, DIRCHECK BASIC/CMD, it will display the directory information for that filespec and report any errors it finds. When DIRCHECK is typed with a drivespec, for example, DIRCHECK :1, it will scan the entire directory for errors and list them on the screen. If not enough memory is available to scan an entire directory (perhaps as a result of a large chunk of high memory being allocated) it will inform the user of the fact and abort the operation.

If DIRCHECK is typed without a drivespec or filespec on the command line, it will return with an asterisk ("*") prompt, and you may type in the necessary information at that point, or press BREAK to terminate the program.

Errors in the directory records are reported first, followed by GAT table errors, if any, and finally, HIT table errors, if any. A DIRCHECK display might look like this:

```
DIRCHECK - DOSPLUS II DIRECTORY CHECK UTILITY - SERIES A.00
(c) (p) Copyright 1982 by MicroPower, Inc.

00239 Free grans, Name = DOSPLUS , DATE = 08/25/82

DEO/TXT    @ Directory sector 002, Relative byte 20H
Invalid filespec
HELP/CMD   @ Directory sector 004, Relative byte 40H
End of file sector beyond allocated sectors
TABLE/TBL  @ Directory sector 006, Relative byte C0H
Multiple files assigned to granule
TABLE/TBL  @ Directory sector 006, Relative byte C0H
Extended directory record has a Zero HIT byte

Cylinder 016 has an invalid GAT Table byte
Cylinder 052 has an invalid GAT Table byte
HIT Byte at 74H Invalid or Extraneous
HIT Byte at 75H Invalid or Extraneous

00008 TOTAL ERRORS
```

In the first part of the DIRCHECK report, the offending files are displayed along with the position of their entries in the directory records. The next line contains the actual error message containing that file. For example, the first file displayed has been found to contain an invalid character in its filespec, which is displayed as an underscore. DIRCHECK reports an invalid filespec for this entry.

The second part of the display reports GAT table errors. Several cylinders (tracks) on the diskette in question have been found to be erroneously mapped in the GAT table. These cylinders may be assigned to particular files, but the fact is not reflected in the GAT table; or, these cylinders may in fact not be in use, but are mapped in the GAT as being allocated to a file.

Finally, HIT table errors are reported. An invalid or extraneous HIT byte may reflect an incorrectly coded HIT byte, or a HIT byte present with no file entry in the corresponding position.

DIRCHECK does not do anything about these errors. It will merely scan a directory or a file entry for errors and report them to the user. If you suspect a problem with a disk directory, this utility can verify the fact. If problems are found, then the companion utility, called DIRFIX can be used to repair most of the errors.

DIRFIX

DIRFIX is a utility program for repairing disk directory errors found by DIRCHECK. DIRFIX will repair most types of directory errors with little user intervention.

==========

The command syntax for invoking DIRFIX is:

DIRFIX :dr {param,param...}

where :dr is the drivespec whose directory is to be checked, param is an optional parameter.

Optional parameters for the DIRFIX utility are:

GAT	Repair GAT Table errors
HIT	Repair HIT Table errors
FILES	Repair File Entry errors
LOCK	Lock out track above directory track (for rigid disk systems only)
PROMPT	Prompt for additional drive information.

These parameters may be abbreviated to their first letter, and may be specified singly or together in any order.

==========

DIRFIX is a utility which will repair most directory errors with very little input from the user beyond specifying which section of the directory is to be repaired. Errors found by DIRCHECK can generally be fixed by DIRFIX. It is not necessary to run DIRCHECK before running DIRFIX, although it is a good idea to do so in order to verify that errors really exist.

When repairing a directory, DIRFIX will use the information in the Drive Code Table for the drive in question in order to determine the size and other characteristics of the diskette. However, you may also supply this information to DIRFIX by including the PROMPT filespec on the command line. It will then return with a "#" prompt, and you may enter the drive number, the number of cylinders (tracks) on the disk, the density of the disk (1 for single, 2 for double) and the number of surfaces on the disk (1 or 2). For example,

:3A,77,2,1

specifies that drive :3A contains a 77-track diskette formatted in double density, and that it is a single-sided diskette.

The parameters GAT, HIT and FILES (or G, H, and F) define which section of directory DIRFIX is to repair. More than one section may be repaired at the same time. In fact, if errors exist in the file entries, DIRFIX will NOT repair the GAT or HIT tables until the file entries have been repaired first. If file entry errors do exist, and the FILES parameter is not specified, DIRFIX will abort with an error message.

It is important to note that repairing file records may result in some file entries being truncated. This will happen if a file has an extended directory entry which is improperly linked to the primary entry, or is linked to by more than one primary file entry. In this case, DIRFIX will simply terminate the file records at the primary entry. Also, if the EOF byte of a file entry points to a sector beyond the last sector allocated to the file, it will be reset to point to the last allocated sector rather than additional sectors being assigned to the file.

After file entry errors have been repaired, DIRFIX can proceed to repair the GAT and HIT tables. When repairing the GAT table, DIRFIX assumes that errors are not confined to the GAT table alone but may affect the rest of the sector. Therefore it will also repair the disk name field, the creation date field, and will insert the hash code for "PASSWORD" (in upper case) in the master password bytes. The disk name will be changed to *DIRFIX*, and the date will be changed to the current date to reflect when the disk was repaired.

If any cylinders (tracks) are found allocated in the GAT table that are not assigned to any of the files, these will be freed up and will become available for use (NOTE: On a hard disk system, any flawed granules may be freed up, since flawed granules are allocated without being assigned to any file). The user should note that if either of the two file entry errors discussed above occur, forcing files to be truncated, tracks may be flagged as "available" in the GAT which may have belonged to a file. In this case it is up to the user to rescue as much data as he can from those tracks before reusing the disk.

When the HIT table is repaired, it is totally reconstructed using the data in the file directory entries. This will result in any incorrect Directory Entry Codes being reconstructed, and any unassigned DECs being removed from the table.

There are two errors which DIRFIX will not repair. The first is when multiple files are assigned to the same granule. Since DIRFIX has no way of knowing which file is actually using that particular granule, it will leave this type of error alone. The user may fix it by killing the extraneous files after determining which file the granule should be assigned to, and then re-running DIRFIX a second time.

The second error which cannot be fixed is the presence of extraneous extended file directory entries. Since such directory entries do not include the file name in them (they are considered to be extensions of the primary entry only) there is no way of knowing which file such an entry belongs to if (a) no primary file entry links to it, or (b) more than one file links to it, and its reverse pointer (which points back to the primary entry) has been altered or destroyed. DIRFIX will not repair this error.

Because of these possibilities, it is a good idea to rerun DIRCHECK after DIRFIX, in order to verify that no errors remain. If any errors remain after DIRFIX has done its job, the user should COPY all valid files on the repaired disk to a new one (do not use BACKUP, or the errors will be transferred as well) and then reformat the first disk.

Diskzap - Disk Editor Program

Introduction

This portion of the DOSPLUS II manual is the "Diskzap Operators manual". Diskzap is our disk editing program included with DOSPLUS II. Please bear in mind as you read this that this portion of the manual is designed to thouroughly acquaint you with the operation of the Diskzap program, NOT to be a tutorial on disk editing procedures. This manual will assume some fundamental knowledge of diskette structure. Some of the more unique features of Diskzap are :

Diskzap allows you to "set" each drive independently of the other, and will remember a drive's configuration, eliminating repeated asking of such questions when executing sub-options.

When modifying a sector, it displays the byte position within the sector and the contents of the byte the cursor is covering.

The display mode also has "wrap around". That is, when modifying a line and the last byte is typed in, we will now move down to the beginning of the NEXT line instead of the beginning of the same line.

The verify option displays the track and sector that it is reading as it verifies the disk.

The format option offers user selectable disk interleaving.

The fill option allows you to set WHAT byte it will fill the sector with instead of arbitrarily zeroing it.

Diskzap is organized very logically into menu options. By first displaying the menu and then covering each option in detail, we hope to thoroughly acquaint you with basic program operation.

The Menu

After entering the filename for the Diskzap program at the DOS command level, it should load in and display the following :

```
DISKZAP - DOSPLUS II Disk Editor - Series A.00
(c) (p) 1982, by MicroPower Inc.


  Set
  Fill
  Copy
  Print
  Verify
  Format
* Display
```

This is the MAIN MENU. It lists all the sub-options and allows you to move between them. The top line is the HEADER. It tells you which version of Diskzap you are dealing with. Diskzap will default to the standard parameters for the machine it is on. The following are the default parameters :

77 cylinders per disk
No skip option
Sector offset of 1
Eight inch drive
27 sectors on track 0
Single density track 0
128 byte sectors on track 0
30 sectors on all remaining tracks
All remaining tracks double density
All remaining sectors 256 bytes long

Any of these may be altered via the "Set" sub-option. The asterisk that appears to the left of the "Display" option on power-up is the "control cursor". Whichever sub-option it is positioned next to is the one that will be invoked when the ENTER key is pressed. It may be moved up and down the list by pressing the <up arrow> and <down arrow> keys. To exit Diskzap, from the main menu press "O" (as in "Out").

Set [Alter disk drive parameters]

Diskzap powers up with the control cursor positioned for this sub-option. If the disk you are working with is a standard DOSPLUS II system diskette, then the configuration corresponds to the default parameters described on the preceding page, then you may proceed directly to the sub-option of your choice and begin the desired operation. If this is NOT the case, then you need to use the Set sub-option to alter them. To function with a DOSPLUS II data diskette, all that must be altered are the parameters for track 0. Configure track 0 for 30 double density sectors 256 bytes in length.

To invoke this sub-option, as with any of the sub-options, simply position the control cursor to the left of the word "Set" and press ENTER. The first question to be asked will be :

Drive ?

Respond to this with the drivespec of the drive that you are configuring. After setting the drive, you will be asked :

Track count ?

Answer this question with the number of cylinders on the disk that you are configuring. Enter the true cylinder count for the diskette. Answer according to media, not hardware. This parameter is interested in how many tracks are on the DISK, not how many your drive is capable of. You will have opportunity to set the skip parameter later.

Technical note : Double headed disks are viewed as two separate disks. Sector numbering repeats itself on the secondary side. When displaying the secondary side of a double headed disk, you must append a "B" to the drive number (i.e. 1B would be the secondary side of drive one). Do not fall prey to the common trap of entering the track count as double what it actually is when configuring double headed disks. An 77 track double headed drive is still 77 track, it simply has a second side.

After configuring the track count of the disk to be edited, the next question asked will be :

Skip option ?

Reply to this question with either a "Y" for yes or "N" for no. The default at power-up is no. The skip option will allow you to read/edit 40 track diskettes in 80 track drives. This parameter is designed for use with the five inch drive adapters currently under development. Although it will function with the eight inch drives as well, there is no practical application for an eight inch, 35 cylinder disk.

Technical note : There are certain inherent dangers and restrictions when using the skip option on 40 track disks in 80 track drives. Not only is the stepping distance different between the two, but the track width itself is slightly narrower on the 80 track units. Because of this, writing to a 40 track diskette in an 80 track drive may cause irreparable harm that will manifest itself as CRC read errors when the diskette is taken back to a 40 track drive. Because of this, MicroPower strongly recommends that you restrict "skipped" operation to reading only.

Once you configured the skip option, you will be queried :

Sector offset ?

Respond to this with a value (0-1) that will be used as the starting sector number for each track. The remaining sectors will be numbered sequentially from there on.

The only real use for this parameter will be for those users who wish to use Diskzap with diskettes formatted by DOS' other than DOSPLUS. For example, Model II TRSDOS formats its diskettes with each track numbered beginning with 1. If you attempt to use the default setting of 0 when you are, say, verifying a diskette formatted by TRSDOS, you would have a "Sector not found" error at the beginning of every track. By setting the sector offset to 1, you would avoid all this and be able to proceed with the verify unencumbered by non-existant errors.

The sector offset is not designed to allow Diskzap to read diskettes with protected formats utilizing random sector numbering and should not be used as such. It will seek to number every sector sequentially beginning with the offset value.

Once you have entered the starting sector number, you will be asked :

Five or eight ?

This questions sets the eight inch option on Diskzap. Diskzap will read/edit five inch diskettes if the proper hardware is present. Answer this question with "5" for a five inch diskette or "8" for an eight inch disk. The default value set at power-up is "8".

Technical note : The standard Model II/Model 16 hardware as supplied by Radio Shack does NOT have the needed hardware to support five inch disk I/O. DOSPLUS and Diskzap will operate with five inch drives, but only if certain hardware is present. If you are not certain whether or not you have such hardware, you probably do not. If you are not certain whether or not the hardware that you have is SUPPORTED, contact MicroPower Technical support division for a list of currently supported five inch disk controllers.

After setting these parameters, you will be asked several questions regarding track zero. In many systems, including DOSPLUS, track zero will differ from the rest of the tracks on the disk regarding density, number of sectors, sector length, etc. To avoid having to stop when accessing track zero and reset Diskzap, we allow you to configure track zero separately.

The first question you will be asked is :

TRK 0 sec/trk ?

This is requesting the number of sectors on track zero. The standard for data disks is 18 sectors single density, 30 sectors double density. For system disks, this is 27 sectors single density. However, when viewing disks formatted by alien systems such as CP/M, there could be variances. Answer the prompt with a value (0-255) to indicate how many sectors there are on track zero. Remember, pressing ENTER at one of these prompts leaves the parameter UNCHANGED. If it was set to the default, then all is well. But if it is not, you will need to reset it to the desired value.

The ability to configure the number of sectors on track zero separate from the rest of the disk would be, by itself, of only limited usefulness. However, Diskzap allows two other configuration parameters. The next one is :

TRK 0 density ?

This parameter allows you to configure the density of track zero separately from the sector count and length. This can be extremely useful, again, in the case of alien systems differing sector counts but the same density. Admittedly, this is rare, but you must still compensate for it.

Reply to this with an "S" for single density or a "D" for double density.

Technical note : DOSPLUS II system disks use a single density track zero. This is required by the ROM bootstrap loader. DOSPLUS II data diskettes, however, format track zero as double density so that the granules not actually USED by the bootstrap can be freed to the system for data storage. In order to avoid confusion, a good rule to remember is :

System disk - Single density (S to S)
Data disk - Double density (D to D)

The next and final parameter to set for track 0 is :

TRK 0 sec len ?

This option will allow you to use Diskzap with systems that format with sector lengths of 128 or 256. 128 byte long sectors are the standard on Model II DOSPLUS and TRSDOS system disks. However, certain systems may use sector lengths other than 128 (such as DOSPLUS II data disks). You may adjust for that here. Reply with the desired value (128 or 256). It will reject all other values.

Following your definition of track 0, you will be given the opportunity to configure those same three parameters for all other tracks on the diskette.

The first query is :

Sectors/track ?

Answer this query with a value (0-255) to indicate how many sectors there are on all the remaining tracks. The standard, of course, will be 18 sectors per track in single density and 30 sectors per track in double density.

However, there is the chance that some system could be using more or less sectors on the track without altering the density that the floppy disk controller works in. This parameter gives you the ability to compensate for that.

After configuring for the number of sectors, you will be queried as to the density of the remaining tracks. You will be asked :

Track density ?

Respond to this question with either "S" for single density or "D" for double density, depending, of course, on the density of the disk.

Once again, remember that the while the power-up default parameter is double density, pressing ENTER leaves it unchanged. This means that if you have been working with a single density disk and you switch to double density you MUST reset this to "D". Simply pressing ENTER will leave it still set to single density.

The final parameter you will be asked for is :

Sector length ?

This option will allow you to use Diskzap with systems that format with sector lengths of 128 or 256. 256 byte sectors are the standard on Model II DOSPLUS and TRSDOS, and all TRSDOS like operating systems will more than likely use them. However, certain systems may use sector lengths other than 256. You may adjust for that here. Reply with the desired value (128 or 256). It will reject all other values.

A final note on Set

When using the set option, you only need to look at as many prompts as are pertinent to you. For example, if all you wanted to do was change the diskette's track count, you could go to the set option and alter the track count. Then you could press BREAK and return the command mode immediately. There is no need to step through prompts that are irrelevent.

It is with this in mind that we have designed the set option. The parameters we felt you were going to use the most (track count, skip option, and sector offset) are close to the front, where you can alter them easily and then avoid the rest of the prompts with the BREAK key.

For the most part, when using set, you will reply with ENTER to questions like "Sector length" and "Sectors/track". These parameters are only needed to *work with diskettes foreign to your system.*

Because pressing ENTER leaves the parameter unchanged instead of re-loading the original default, you do not need to re-enter a parameter that is set the way that you want it. Set will retain this drive configuration for as long as Diskzap is in operation, but must be re-configured upon each new entry of the program.

Fill [Fill sector with specified byte]

This option will allow the user to fill a sector with any particular byte that may be desired. This is useful when it is desired to erase completely old data from a sector without re-formatting the entire disk.

To invoke this sub-option, place the control cursor to the left of the word "Fill" in the main menu and press ENTER.

The first question to be asked is :

Drive ?

Reply to this with the number of the drive that contains the diskette to be operated on. Any valid drivespec will be allowed here. Remember that to work with the secondary side of a double headed diskette, you must append a "B" to the drive number. Appending an "A" to a drive number is valid, but is also assumed and therefore superfluous.

After answering that question, you will be asked :

Track ?

Answer this question with the track number that contains the first (or only) sector to be filled. As with all input prompts in Diskzap, you may respond in any number base. Decimal is the default and will be assumed. Be certain to append the proper type specifier to any other base entries (e.g. "H" for hexidecimal, "B" for binary, and "O" for octal).

Once the track is entered, you will be asked :

Sector ?

Reply to this query with the number of the first (or only) sector to be filled.

The next prompt will be :

Sector count ?

Respond to this with a value that represents the number of sectors, beginning with the sector specified in the preceding questions, to be filled.

The final question will be :

Fill data ?

Answer this question with the byte that you wish to have the sector filled with. This can be a one or two byte value. Pressing ENTER at this prompt will use the default fill value, which is zero.

Example -

If you wanted to fill tracks 4 and 5 of a particular double density diskette with the hexidecimal value "E5", you would answer the questions in the following manner :

Drive ? 0
Track ? 4
Sector ? 0
Sector count ? 60
Fill data ? E5H

After inputting all data and pressing ENTER on the last prompt, the drive will run and Diskzap will display the track and sector number as it fills each sector.

Copy [Copy sectors]

This function will allow you to copy sectors from one disk to another or from one part of a disk to another. To invoke this command, place the control cursor to the left of the word "Copy" in the main menu and press ENTER. The first question to be asked is :

Drive ?

Answer this with the drivespec of the SOURCE drive. Next, you will be asked :

Track ?

Answer this with the track number that contains the first (or only) SOURCE SECTOR. This is the sector that is to be copied (or the first of many, whichever you desire). After answering that question, you will be asked :

Sector ?

This is prompting you for the number of the first (or only) SECTOR TO BE COPIED. Once you have input that, you will be prompted for :

Drive ?

This time it is seeking the drivespec of the DESTINATION DRIVE (the drive to which you wish to copy).

The next prompt is :

Track ?

Answer this with the number of the track on the destination drive that contains the first (or only) DESTINATION SECTOR.

After answering that, you will be queried :

Sector ?

This is prompting you for the number of the first (or only) SECTOR TO COPIED INTO. It does NOT necessarily have to be the same as the source sector (i.e. you can copy the last two sectors of track 4 on drive 0 into the first two sectors of track 7 on drive 1).

The last piece of data required will be :

Sector count ?

This prompt is seeking the number of sectors that you wish to copy.

Technical note : Please remember that when you are using COPY, you are defining a "block" of sectors. You specify the starting point of this block on both the SOURCE and DESTINATION drives. The "sector count" prompt allows you to define the length of the block. Pressing ENTER will copy only a single sector. But, it must be a CONTIGUOUS block. You are copying sequentially from the source sector to the destination sector for the number of sectors you specify. What this means is, if you wish to copy 50 sectors, skip 200, and copy 50 more, you will have to copy each block of 50 separately. You may, if you wish, locate them beside each other on the destination drive, but they must be copied independantly.

Example -

If you wished to copy track 2, sector 5 of drive 0 into track 3, sector 12 of drive 1, you would answer the prompts in the following manner :

```
Drive ? 0
Track ? 2
Sector ? 5
Drive ? 1
Track ? 3
Sector ? 12
Sector count ? 1
```

If you wished to copy an entire DOSPLUS II, 77 track data diskette from drive 0 to drive 1, you would answer the prompts in the following manner :

```
Drive ? 0
Track ? 0
Sector ? 0
Drive ? 1
Track ? 0
Sector ? 0
Sector count ? 2310
```

After answering the "sector count" query and pressing ENTER, Diskzap will begin the copy. When copying sectors, Diskzap will seek to read in as many sectors as it can (up to one complete track) before writing them, as opposed to reading and writing a single sector at a time.

When copying a single sector, there will be no operational difference. However, when copying more than a track (especially an entire disk), it makes LARGE difference. Diskzap will also displays the track and sector number of each sector as it is copied (both the SOURCE sector as it is read and the DESTINATION sector as it is written).

If Diskzap encounters an error during the sector copy routine, it will pause and display the error discovered. It will also ask if you wish to continue. It would then write as much of the source sector as it could read into the proper destination sector and proceed from there. This will allow you to copy as much data as is absolutely possible from a disk without having to work around known bad sectors. This "proceed after error" feature becomes a key one in repairing blown diskettes. If you can copy a complete track save one sector, then you have only lost 256 bytes of data as opposed to potentially much more.

Print [Print hardcopy of selected sectors]

This command will create printed copy of the contents of specified sectors. You may, of course, obtain hard copy of one sector from the display mode by using your particular machine's screen print function (<control "-">). This can be tedious for multiple sectors, however, and that is where print is used.

To invoke this option, position the control cursor to the left of the word "Print" in the main menu and press ENTER.

The first question asked will be :

Drive ?

Answer this with the drivespec of the drive that contains the first sector to be printed. Next you will be asked :

Track ?

Answer with the number of the track that contains the first (or only) sector to be printed. Following that, you will be queried :

Sector ?

Enter the number of the first (or only) SECTOR TO BE PRINTED. Finally, you will be prompted :

Sector count ?

Reply to this with the number of sectors that you wish to print. Remember, just as with copy, you are dealing with contiguous blocks ONLY! You may not print 5 sectors on track 0 and then 5 on track 11 without printing them both independantly of one another.

Example -

In order to print out all the directory sectors (assuming the directory was on track 26 or 1A hex) from the double density diskette in drive 0, you would :

```
Drive ? 0
Track ? 26
Sector ? 0
Sector count ? 30
```

As each sector is printed, it will be displayed on the screen. You may tell by examining the track and sector indicators in the upper left hand corner of the screen which sector is currently being printed.

Technical note : Diskzap does NOT check for printer ready status. If you engage the print option and there is no printer available, Diskzap will simply "lock up" and force you to either make a printer available or reset the machine.

Verify [Read and check specified sectors]

This option will allow you to read and verify any specified setors on the disk. It will check each sector for accuracy by verifying the CRC byte. If it encounters an error, it will pause with the correct error message. Pressing ENTER will cause it to continue verifying.

To invoke this option, as with any other, position the control cursor to the left of the word "Verify" and press ENTER. The first question asked will be :

Drive ?

Reply to this question with the drivespec of the drive that contains the diskette that you wish to verify. The next question asked will be :

Track ?

This is prompting you for the track number that contains the sector you wish to begin verifying at. When verifying an entire diskette, you may press ENTER at this prompt to select track 0. After answering that, you will be asked :

Sector ?

This is asking you for the sector number on the above specified track that you wish to begin verifying at. This would allow you to begin verifying with the last two sectors of track 5. Following that, you will be prompted :

Sector count ?

This is seeking the number of sectors you wish to verify. Remember, if you specify more sectors for a disk than you have configured for in "Set", it will wrap around from the last configured track and begin again at track 0, sector 0 and continue from there. That is why it's important to configure for the correct track count before beginning with any diskette.

For your convenience in verifying entire diskettes, here are some full disk verify sector counts :

DOSPLUS II System disk - 2307 sectors
DOSPLUS II Data disk - 2310 sectors

While you are verifying a diskette, you may abort and return to the main menu by holding down the BREAK key.

Examples -

If you wanted to verify your Model II DOSPLUS system disk in drive 1, you would answer the prompts in the following manner :

Drive ? 1
Track ? 0
Sector ? 0
Sector count ? 2307

If you wanted to verify your Model II DOSPLUS track data diskette in drive 2, you would answer the prompts in the following manner :

```
Drive ? 2
Track ? 0
Sector ? 0
Sector count ? 2310
```

Once you have answered the final question and pressed ENTER, Diskzap will begin reading the specified sectors. It will display the track and sector number as it verifies each sector. As each sector is read, the CRC value is calculated and checked and any errors reported. If it detects any non-standard data address marks, Diskzap will pause on that track and sector and print the message "AM/WRITE FAULT!". This message serves to indicate that a non-standard data address mark was found.

Pressing ENTER will proceed with the verify. Even though Diskzap paused with the data address mark, the sector's CRC byte was checked and a complete verify was done. The CRC byte and its function is explained in the "General notes and conventions" section of this manual.

Format [Format a selected track or tracks]

This sub-option allows you to format a track or series of tracks. You may, if you wish, use it to reformat a track somewhere in the middle of a disk to repair a non-readable sector. To invoke this option, position the control cursor to the left of the word "Format" in the main menu and press ENTER.

The first question is :

```
Drive ?
```

This is prompting you for the drivespec of the drive that contains the disk you wish to format a track on. After answering that, you will be queried :

```
Track ?
```

Respond to this with the number of the track at which you wish to begin formatting. The next question is :

```
Track count ?
```

This is seeking the information as to how many tracks you desire to format. Pressing ENTER at this prompt will default to one track.

The final prompt is :

```
Interleave factor ?
```

This is prompting you for the sector interleave factor. Diskzap, if formatting more than a single track, will implement true DISK interleving.

Technical note : Disk interleaving is simply a factor that controls how the diskette is formatted. It concerns the sequence in which the sectors are numbered on the track. MicroPower has done extensive research into disk interleaving and arrived at the optimum value for the interleave factor on standard hardware (interleave factors can be affected by processor speed). We recommend an interleave factor of 3 in double density and an interleave factor of 2 in single density. These are the values used by our standard floppy disk formatter utilities. However, for special applications, you may desire to change this. This parameter allows you to do that.

Examples -

If you wanted to repair track 10 of your Model II double density data diskette, you would first copy what sectors were readable onto either another diskette or a free track on the same disk. Then you would answer the prompts in the following manner :

```
Drive ? 0
Track ? 10
Track count ? 1
Interleave factor ? 3
```

That would format the track. After re-copying the old data onto it, the track would be as close to repaired as it was going to get. This technique can often be used to rescue a file, losing only a sector or two (at most an entire track), and possibly saving thousands of records and hundreds of man-hours.

After answering the last prompt and pressing ENTER, Diskzap would begin formatting. It will display the track number it is formatting as it proceeds. We elected not to increment the sector number on the display because the formatter proceeds so fast that it only slowed it down to display such a number. Therefore the sector number will always be zero and only the track number will increment.

Technical note : Diskzap's formatter does NOT write system data (a boot-strap and a directory) to the disk. This formatter was designed for use within Diskzap. A diskette formatted by Diskzap is NOT ready to receive data in most standard applications. For that, you must use the FORMAT utility included with DOSPLUS.

Display [Display or modify diskette sectors]

This is perhaps the most often used option in Diskzap (followed closely by Verify), because this is the heart of any disk editor, the edit mode. Diskzap uses a full screen editor that has cursor wraparound.

To invoke this sub-option, position the control cursor to the left of the word "Display" in the main menu and press ENTER.

The first question you will be asked is :

Drive ?

Answer this query with the drivespec of the drive that contains the diskette with the sector you wish to display/modify. The next question is :

Track ?

This is prompting you for the number of the track on the disk that contains the sector you wish to examine.

The final question is :

Sector ?

Reply to this with the number of the sector you wish to display.

After typing in the sector number and pressing ENTER, you should see a display that looks something like this :

```
270200: C1CD 831D C03E 1BB7 C93E FF12 0E00 4679 ....>.7.>....Fy
270210: CD9E 1D28 071A C620 120C 18F3 79CD AB1D ..(... ....y.+.
270220: B677 1D1A 3C20 027D 121C 1A3C 12C1 0BC5 6w..< .}...<....
270230: 78B1 C2F0 1CC1 3E00 47E6 E06F 261B CB66 xl....>.G..o&..f
270240: 2004 3E18 B7C9 DD4E 10CD 401E C0CD 2703  .>.7..N..@...'..
270250: C9E6 0707 0707 F640 32A9 1DCB 40C9 E607 .......@2)..@...
270260: 0707 07F6 C732 B71D AFCB C7C9 DD4E 10CD .....27./....N..
270270: 541E C0DD 7E11 CD10 1E3E 1EC0 457D 320B T.......>..E}2.
270280: 1E54 DD5E 111A 77CD 661E C0CD 2403 C036 .T.^..w.f...$..6
270290: 902C C53A 841D 772C 0614 3600 2C10 FBE5 ,.:.w,..6.,...
2702A0: 060A 36FF 2C10 FBD1 13C1 CD27 03C0 3A01 ..6.,......'..:.
2702B0: 7EFC 1D84 1D47 CD24 03C0 7DC6 1E6F 36FE ...G.$..}..o6.
2702C0: 2C36 00CD 2703 C96F CD17 1EC8 2E00 7EB7 ,6..'..o......7
2702D0: C83E 2085 6F30 F7FD 7E11 D603 2CBD 30EE .>o0.....,=0...
2702E0: F6FF C9C5 D5E5 0600 1E00 2100 203E 06D7 ..........!.....
2702F0: E1D1 C1C8 3E14 C9C5 D5E5 0600 1E00 2100 ....>.........!.
```

The first item to notice is in the left hand column of the display. You will see six characters followed by a colon. The first two characters are the TRACK NUMBER (displayed in hexidecimal format). The second two characters are the SECTOR NUMBER (also displayed in hexidecimal format). The final two characters are the BEGINNING BYTE INDICATORS. Each one of those indicates the number of the first byte in that row. Then there are rows of 16 bytes each (10 hex). This is the HEXIDECIMAL DISPLAY AREA. These are set in groupings of two bytes, such that you have eight columns of two separated by spaces. Immediately to the right of the hexidecimal portion of the sector display is the ASCII DISPLAY AREA. There are 16 ASCII characters on a row corresponding to the bytes in the hexidecimal display row immediately to its left. Non-ASCII characters will be displayed as periods.

Options -

At this point, you have several options, each of which is controlled by a single keystroke. They are :

Key	Function
;	Increment display position one sector
+	Increment display position one track
-	Decrement display position one sector
=	Decrement display position one track
BREAK	Return to main menu
M	Enter modify mode

If you select "M" to enter the modify mode, the display will change slightly and you will have several other options.

If Diskzap encounters an error during a sector read in the display mode, it will pause and display the error discovered. It will also ask you if you wish to continue. If you respond "Y", it will display as much of the sector as it could read. You may then enter the modify mode and make any corrections possible before re-writing it. The sector will be re-written to the disk reflecting any corrections you may have made. That means there will no longer be a read error from system level. It does not mean that the data is now 100% correct. It is correct only to the level that were able to repair it, but it will read as it is now without an error. This "continue after error" feature will allow you to rescue bad sectors in part or in whole, where otherwise you would have had no chance of recovering the data.

Modify mode -

When you enter the modify mode, a reverse video cursor will appear over the byte in the upper left hand corner. You move this within the sector by using the arrow keys. Whatever byte is currently highlighted in reverse video, that is referred to as the CURRENT CURSOR LOCATION. This is the byte that will be affected should you enter a change.

Options -

At this point, you have several options, each of which is controlled by a single keystroke. They are :

Key	Function
right arrow	Increment cursor position one byte
down arrow	Increment cursor position one row
left arrow	Decrement cursor position one byte
up arrow	Decrement cursor position one row
BREAK	Aborts modify mode and returns you to the main menu without re-writing the sector. Restores original contents.
ESC	Aborts modify mode and returns you to the display mode without re-writing the sector. Restores original contents.
ENTER	Complete modification and returns you to the display mode after writing the modified sector to the disk.
Z	Fills sector from current cursor position to the end of the sector with zeros "00".

When in the modify mode, only valid hexidecimal characters or motion commands (the arrow keys) will be accepted, all others will be ignored (except for "Z"). All hexdecimal letters (i.e. A-F) must be entered as capitals.

When you finish modifying one byte, the cursor will move onto the next. If that was that last byte of a row, the cursor will move onto the first byte of the NEXT row. The only exception is the last byte of the last row. After modifying it, the cursor will stay right where it is. To begin with the next sector, write this one back to the disk with ENTER, advance to the next sector with ";", enter the modify mode again with "M", and return to modifying.

General notes and conventions

When Diskzap prompts you for a track number or a sector number, these values must always be entered in any acceptable base. Simply remember to append the proper base specifier to the end of any non-decimal entries. Consult the DOS Operations section for further details on this. Before you begin editing your diskettes directly, you should at very least be familiar with hexidecimal notation.

When Diskzap prompts you for track count, sector count, or something similar, these values can also be entered in any base.

Any prompts that are seeking data such as "Fill byte" will require a value either one or two bytes in length. For the sake of simplicity, you should enter these in hexidecimal format. However, you may use any valid base here also.

Always, when you are prompted for "Drive ?", if you wish to operate on the secondary side of a double headed diskette, append a "B" on to the drive number. For example, the secondary side of drive 1 would be 1B. The same sector numbers are used again on the second side, therefore you must have some method of indicating to Diskzap with side you are referring to.

Whenever Diskzap prompts you for anything, it will always have a default value. You obtain this default value by simply pressing ENTER in response to the prompt.

In the case of "Set", the default value is the current setting. In other words, there is no consistent default value. It will simply leave the current setting unchanged. This allows you to "skip ahead" and change only the desired parameters by pressing ENTER when the currently displayed parameter doesn't need to be altered. This is explained in the "Set" portion of the manual.

In the case of all the other parameters, the default value will always be one of two things, depending upon the type of question.

In the case of a "count" question, such as "Track count" or "Sector count", the default will be one (1), the lowest possible unit. For example, if you are formatting a disk using "Format" and you press ENTER at the track count prompt, it will format one track. If you are copying sectors and press ENTER at the sector count prompt, it will copy over one sector.

In the case of a "location" question, such as "Track" or "Sector", the default will be whatever the beginning of first number would be. For example, when it asks for the drive and you press ENTER, you get drive 0. If you are asked for the track number and you press ENTER, you would get track 0. If you are asked for sector number and you press ENTER, you would get whatever the first sector number is (0 or 1, depending upon the sector offset).

The one exception is the "Fill data" prompt in "Fill". It is neither a count or a location question. Its default value is "00".

CRC bytes -

CRC (cyclic redundancy check) is calculated by running all 256 bytes of information in a sector through an algorithm which produces a two byte value for that data. This is stored in the sector ID field during a write. During a read, the same calculation is done. If the calculated value does not match the stored value, then an error has occurred. Because of the way a 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.

DRAW

DRAW is a utility program for generating screen displays using the Model II's graphics characters.

==========

The command syntax for invoking DRAW is:

DRAW

==========

DRAW is a graphics screen editor utility which will allow you to generate screen displays of mixed text and graphics and save the displays to disk. Screen displays saved to disk may be recalled at any time for further editing.

DRAW uses the alphabetic keys of the keyboard to generate each of the graphics characters. The correspondence of keys to graphics characters may be viewed by invoking the "help" frame with CONTROL-9. Pressing any of the alphabetic keys will place the corresponding graphics character on the screen at the current cursor position. If the key is held down, it will begin to repeat. Movement of the repeating keys is always left to right.

The cursor is positioned by means of the numeric keypad. The numeric keys are also used to invoke DRAW mode, which will draw a line on the screen as the cursor is moved; ERASE mode, which will reset whatever graphics are on the screen that the cursor moves over; and SKIP mode, which will allow the cursor to move across the screen without affecting any graphics designs in its path.

Cursor direction is determined from the following pattern (on the numeric keypad):

```
    8
4   5   6
    2
```

That is, holding 8 will move the cursor in an upward direction (toward the top of the screen), 4 will move it left, 6 will move it right, and 2 will move it down. Pressing 5 will cause the cursor to be moved to the "home" position in the upper left of the screen.

The other numeric keys are used as command keys, as follows:

7	Switches DRAW mode on. Graphics blocks beneath the cursor will be turned ON and left on.
9	Switches SKIP mode on. The status of any graphics blocks (on/off) that the cursor moves over will be left unchanged.
1	Switches ERASE mode on. If any graphics blocks are turned on, moving the cursor over them while in ERASE mode will turn them off.
3	Toggles between TEXT and GRAPHICS modes. While in TEXT mode, the keyboard functions normally, and alphanumeric text will be placed on the screen at the current cursor position.

The following keys are used to save and load the graphics screen to disk:

<	Loads graphics screen from disk.
>	Saves graphics screen to disk.

The graphics screens are saved onto a file called DRAWSAVE on drive 0.

FORMAT

This utility allows you to organize a diskette and prepare it to receive data.

==========

The command syntax is :

FORMAT drivespec {param=exp...}

"drivespec" specifies the drive containing the disk to be formatted. If this is not given at the command line, FORMAT will prompt for it.

"param" is the optional action parameter that modifies the effect of the command.

"exp" is the optional expression that indicates the function of the parameter.

Your parameters are :

DATE="string"	Allows you to set the format date from the command line. This should be expressed as a quoted literal up to eight characters in length. You are not restricted to numeric input. If this is not given in the command line and the system date is not set, FORMAT will prompt you for it.
PASS="string"	Disk Master Password. This parameter allows you to specify the Disk's master password from the command line. This should be expressed as a quoted literal up to eight characters in length.
NAME="string"	Allows you to specify the disk name from the command line. This should again be expressed as a quoted literal and may be up to eight characters in length.
CYLS=value	Number of cylinders. This allows you to specify the number of cylinders to format the disk to. This should be expressed as a numeric value, not a quoted literal.

SIDES=value	Number of sides. This allows you to specify single or double headed format from the command line. This should also be expressed as a value (either 1 or 2), no quotes are needed.
TDEN="string"	Track density. This parameter allows you to specify the format density from the command line. It should be expressed as a single character quoted literal. Either an "S" for single density or a "D" for double.
USE=switch	This allows you to override the prompt "Diskette contains data, Use or not?" that appears when the disk to be formatted is not blank. This is your only warning, so using this parameter can be dangerous if used without caution. This parameter should be expressed as a switch (either ON, OFF, Y, or N.). Since it defaults to "Y", though, simple inclusion of the word "USE" or the letter "U" in the command line is enough to override the prompt.

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

DATE	01/01/80
PASS	No password
NAME	No name
CYLS	77
SIDES	1
TDEN	Double density
USE	No

Abbreviations :

DATE	D
PASS	P
NAME	N
CYLS	C
SIDES	S
TDEN	T
USE	U

The FORMAT utility is used to organize the diskette into tracks and sectors and prepare it to receive data. You will use this both in formatting new disks and in "starting over" with a clean slate on old ones. All disks must be formatted before they can be used by the system. If is NOT, however, necessary to format a disk before backing up to it (see the utility program BACKUP). BACKUP will format the destination disk if it is blank.

The disk to be formatted may be either blank or contain data. If you format a disk that already contains data, any data on that disk will be permanently lost. When you format a disk, DOSPLUS II will check the disk for flawed granules. If it discovers any areas of the disk during format that are bad, it will "lock out" those areas and prevent the system from attempting to use them.

To format a disk, type "FORMAT" from the DOS command mode and press ENTER. The first message to appear will be :

Target drivespec ?

Enter the drivespec of the drive that contains the disk you wish to format. You will then be asked :

Diskette name ?

Enter the name you wish to assign to that disk. Any characters are legal (numeric or alphabetic). You have a maximum of eight characters. Following that, you will see :

Format date ?

Enter today's date. You may, if you wish, use this field for something else. It will be displayed whenever you execute a CAT, DIR, or FREE upon that disk. DOSPLUS II doesn't use the disk date for anything, so this area is free for you to use. Eight characters maximum. May be alphabetic or numeric. After entering this, FORMAT will prompt you :

Master password ?

Enter the desired Disk Master Password. This password will be used for a variety of functions later. Pressing ENTER will default to "null password", but from then on you will not be able to assign effective file protection. The Disk Master Password will always override the file password. If a Disk Master Password is NOT set, then specifying no password will ALWAYS get you into a file. We therefore recommend that the Disk Master Password always be set. Maximum of eight characters. Once you have answered that prompt, you will see :

Number of cylinders (35-96) ?

Enter the number of cylinders you to which you wish to format the disk. The standard floppy disk drives will be 77 cylinders. If you are using RFORMAT to format the hard disk (discussed later), this value will vary greatly. Enter the number of cylinders desired or press ENTER to default to 77. After that, you will be queried :

Number of sides ?

Enter "1" for single sided (Model II) drives or "2" for double sided (Model 16) drives. Remember that single or double sided is limited by your drive hardware. Simply answering this prompt "2" in the Model II is NOT going to give you a double sided disk. After answering this, you will be asked :

Single or double density ?

Enter "S" for single density or "D" for double. Both Model II and Model 16 use double density disk drives. Pressing ENTER defaults to "D". DOSPLUS II formats 18 sectors per track in single density and 30 in double.

After you have answered all these questions, DOSPLUS II will proceed with the format. If the diskette was not blank, you will be warned :

Diskette contains data, use or not ?

Enter "Y" to proceed or "N" to abort. Pressing BREAK will also abort.

If the disk was blank, or you have signalled FORMAT to use it anyway, you will see the track number displayed as first they are formatted and then verified. When the procedure is complete, you will see :

Insert SYSTEM disk (ENTER)

flashing on the screen. Make certain that a system disk is inserted and then press ENTER to return to DOSPLUS II.

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

RFORMAT

DOSPLUS II includes a utility called RFORMAT that is used to format the rigid drives (hence the name RFORMAT). It functions identically to the standard floppy disk formatter with two exceptions.

First, instead of limiting you to 96 cylinders, it will allow values from 35 to 200. 200 cylinders is the maximum allowed for any single volume of the hard drive. Therefore, you will enter the correspondingly larger number.

Second, the question "number of sides?" is now "number of surfaces?" and is referring to the number of read/write surfaces available. The standard Radio Shack 8.4 megabyte hard disk (Cat. # 26-4150/51) has 4. Also, the question regarding density will not be asked. There are no varying densities in rigid drives.

Examples:

```
FORMAT
```

This will call the format utility without specifying any parameters. FORMAT will load and prompt the operator for all information.

```
FORMAT :1
```

This accomplishes the same as the above example, except that it furnishes FORMAT with the number of the drive to be formatted. Therefore, this information will not be prompted for.

```
FORMAT :2 {DATE="OCT",PASS="PW",NAME="D1",CYLS=77,SIDES=1,TDEN="D",USE=Y}
FORMAT :2 {D="OCT",P="PW",N="D1",C=77,S=1,T="D",U=Y}
FORMAT :2,D='OCT',P='PW',N='D1',C=77,S=1,U
```

These three commands will produce the same results. They will format the disk in drive 2 without prompting the operator for any further information, even if the disk is not blank.

```
RFORMAT :4 {N="DRIVE A",D="10/19/82",P="PASSWORD",C=200,S=4,U}
```

In this example, we are formatting the hard disk ":5" for 200 cylinders using 4 surfaces. The drive will be formatted without further input regardless of whether or not it contains data already.

Finally:

Remember, when using FORMAT and RFORMAT, any data that exists on a disk is lost when the disk is re-formatted. Please be careful not to accidentally destroy valuable data.

RFORMAT is used to format the rigid drives ONLY. FORMAT is used to format the floppy drives ONLY. Please only use the correct program for your drive. For further, more detailed instructions on installing the DOSPLUS II system on a hard disk, see the utility program SYSGEN.

When using RFORMAT, the parameters given at the start of this utility for FORMAT are still valid. Simply use the "Size" parameter for the number of surfaces and specify the correct number of cylinders with the "Cyl" parameter.

HELP

HELP is a utility which displays information about the DOSPLUS II library commands.

==========

The command syntax for invoking HELP is:

HELP library-command

==========

HELP provides on-line information about syntax and parameters for the DOSPLUS II library commands. When entered with a library command for an argument on the command line it will display several lines of information about that command. The command syntax will be displayed, followed by the parameters. Abbreviations for the parameters will also be given.

To display a list of the commands for which HELP is available, simply enter HELP without any argument.

MAP

MAP is a utility program which will display the sectors occupied by a particular file when given a filespec, or the free/allocated status of each granule on a specified diskette when given a drivespec. It will also display which file a particular sector is assigned to.

==========

The command syntax for invoking MAP is:

```
MAP filespec
MAP :dr
MAP :dr,xx,yy
```

where "filespec" is the file to be MAPped, and :dr is the drivespec where the diskette to be mapped resides. "xx" and "yy" are cylinder and sector numbers, respectively.

==========

MAP is a utility program which can locate files on a diskette or logical drive (on a hard drive system). Given a filespec, it will display directory information for that file, as well as the cylinders and granules that the file occupies. Given a drivespec, it will display a map of the free and allocated granules on that disk. If a cylinder and sector number are given with the drivespec, then MAP will display the name of the file which that particular sector is assigned to, if any.

When a filespec is given as an argument to MAP, the following information will be displayed for that file: the file's directory entry code (DEC), its logical record length (LRL), its end-of-file byte (EOFB) and end-of-file sector (EOFS), followed by a display of the track numbers and granules assigned to it. For example:

```
TEST/CMD - DEC=96H  LRL=256 EOFB=00H EOFS=0010
44, 20- 24   44, 25 -29
```

The first line displays the file name and directory information regarding the file. The second line indicates which cylinders and sectors are occupied by the file.

When MAP is used with a drivespec alone, then a map of the GAT table is displayed. Granules allocated to files are marked with a lowercase "x" beside them, free granules are marked with a "." (period), and the directory cylinder is indicated with a "D." Any locked-out granules are marked with an "L." This is very similar to the FREE library command's map.

MAP can also be specified with a drivespec, cylinder number and sector number, for example, MAP :0,17,0. This tells MAP to examine drive :0, cylinder 17, sector 0 and determine which file it belongs to. If the sector in question is assigned to a file, it will display:

BASIC/CMD @ Relative Sector 00024

indicating that the sector in question is assigned to the file BASIC/CMD and is the 24th relative sector of that file.

If the sector has not been allocated to any file, MAP will display the message:

Sector NOT ASSIGNED.

MAP is useful for determining disk utilization as well as for locating a particular file on disk prior to examining it directly using the DISKZAP utility.

OFFSET

OFFSET is a utility which can identify the load addresses of a particular file, or offset a file so that it loads into another memory location from where it normally goes. Optionally, an appendage may be added to such a file to move it back to its old location prior to executing.

==========

The command syntax for invoking OFFSET is:

OFFSET [FROM] channel1 [[TO] channel2] {param, param}

where channel1 is the file to be read or offset, channel2 is the optional filespec for the offset file, and param is an optional parameter.

The parameters for the OFFSET utility are:

READ	Read only mode. Do not alter specified file.
SHOW	Display load blocks of specified file.
OFFSET=address	Offset specified file so it loads into "address."
APPEND	Add appendage routine to block-move an offset file to its old location after loading.

The parameters may be abbreviated to their first letter.

==========

OFFSET is a utility which can display the address locations where a machine language file loads into, and also change the load addresses of a machine language file so that it loads at another location. This is useful when two files which occupy the same memory locations must be loaded together. OFFSET can also add an appendage routine which will block-move the file from its new load point to its former load address for execution.

When changing the load addresses of a file, OFFSET will scan the command line for the presence of two filespecs. The filespec in the FROM field is then used as the source file, and the relocated program is written into the filespec in the TO field of the command line. If no extension is given, /CMD is assumed for the destination file. If only one filespec is given on the command line, OFFSET will write the relocated program back into the same file.

If no filespecs are specified, then OFFSET will return with an asterisk prompt ("*"). At this point you may type in the required information.

The {READ} parameter places OFFSET in read-only mode. In this mode, OFFSET will read a file and display its starting address, ending address, and the number of "load bytes" -- bytes actually loaded into memory. If the file has a transfer address, that is, it can be executed by the system after loading, a second line on the display will show the this address. If the file does not have a transfer address (it is not executable by the system immediately after loading) this second line will not be displayed. All address values are given in hexadecimal.

The READ parameter inhibits OFFSET from writing a file back out to disk. Therefore if the command

OFFSET BASIC/CMD TO NEWBASIC/CMD,R

is given, the second filespec "NEWBASIC/CMD" will be ignored.

The {SHOW} parameter displays the starting and ending addresses of each load block of a file. Load-module format files usually consist of several blocks of 256 bytes or less, preceded by a loader code and a two-byte address which tells the system loader where that block is to go in memory. OFFSET will read a file and display the addresses for each load block. For example,

BLOCK LOADS FROM 2200H TO 22FEH
BLOCK LOADS FROM 22FFH TO 2350H
BLOCK LOADS FROM 2475H TO 256EH

If the SHOW and READ parameters are given together, the READ display as described in the preceding paragraph will appear after all the block addresses have been displayed.

The {OFFSET=address} parameter allows you to specify a new starting address for the file. All the load block addresses will be changed to reflect the new starting address. The transfer address, if present, will also be changed. If a destination filespec was given, the new program will be written out under that name. Otherwise it will be written out under the same name.

Note that the OFFSET utility will NOT alter the program itself. It only alters the locations where the program loads. Thus, if a machine language program contains branches to absolute memory locations (e.g., JP 2204H), these will remain unchanged, and executing the program in its new location will more than likely have unexpected results.

The {APPEND} parameter will add a 22-byte routine, called an appendage, to the offset file. This routine will save all the primary Z-80 registers on the stack, block-move the program back to its original load address, then jump to the program's transfer address. This permits the creation of a file that will load at a different location from its execution address, and still execute properly.

The OFFSET utility can also be used to correct the EOF offset of certain program files. On some files, the EOF byte in the directory entry does not point to the actual last byte of the file, but to some point beyond it. An example of this is a file whose EOF pointer in the directory is on a sector boundary but whose actual last byte is in the middle of the sector. The command OFFSET filename without any parameters will cause a file to be read in and written back out under the same filename, with the EOF byte in the directory entry now pointing to the proper location. It may be necessary to do this to some files before the APPEND library command of DOSPLUS II can be used properly.

Examples:

```
OFFSET MEM/CMD:0,R,S
```

OFFSET will read the file called MEM/CMD, display the addresses of each load block, and at the end display information about starting and ending address of the file, the number of bytes loading into memory, and the transfer address, if any.

```
OFFSET BASIC/CMD,OFFSET=9000H,A
```

The load addresses of the file BASIC/CMD will be changed so that it starts loading at 9000H. In addition, the block-move appendage will be added to the file so that after loading at 9000H, it will be relocated to its normal load address and executed. The offset file will be written back out as BASIC/CMD, replacing the old file.

```
OFFSET TBAR/CMD 2BAR {O=571FH}
```

The load addresses of TBAR/CMD will be offset so that the program loads at 571FH. The relocated file is written back out to disk under the filename 2BAR/CMD (OFFSET supplies a default filename of /CMD).

```
OFFSET
```

The OFFSET program will return with an asterisk prompt ("*") and wait for the user to type in the necessary information before proceeding. If the BREAK key is pressed, the program will terminate.

PATCH

PATCH is a utility program which is used for making minor changes to disk files.

==========

The command syntax is:

```
PATCH file1 file2 [{PULL}]
PATCH file1
```

where file1 is the filespec of the file to be patched, file2 is the filespec of the file containing the patch data, and PULL is an optional parameter.

==========

PATCH allows you to make changes to disk files in a convenient manner. It is normally used for making changes to command files (files in load module format); however it can be used to change any kind of file. PATCH will assume an extension of /CMD for the target file if one is not provided.

PATCH data is usually placed in a separate file which may be prepared with the BUILD command or with a text editor. This file may be given any valid file specification; PATCH will assume an extension of /PAT if one is not explicitly given.

PATCH Command Syntax:

The syntax for the Patch utility is as follows:

```
PATCH file1 file2
```

where file1 is the target file (file to be patched), and file2 is the /PAT file containing the patch information. After the patches have been successfully applied, PATCH will display the message:

```
Patch(s) installed
```

to inform you that the patch information has been placed into the target file. If file2 is not present on any mounted disks, Patch will abort with an error.

PATCH File Syntax:

Patch files may have three kinds of lines: Comment lines, null lines, and patch data lines.

A comment line is one which begins with a period (".") as its first character and terminates with a carriage return. PATCH will ignore such lines, and they are provided for the user's benefit, so he can document his patches. For example,

```
.This is a comment line
```

A null line is a line which consists of nothing but a carriage return. PATCH will also ignore such lines, and they may be used to separate patch data lines into blocks for better readability.

Patch data lines contain the information that the PATCH utility uses in changing the target file. There are two kinds of patch data lines:

1) ADDRESS-ORIENTED lines, and
2) SECTOR-ORIENTED lines.

ADDRESS ORIENTED Patch Lines

This type may be used only to patch files in load module format. The syntax of the patch line is:

A=aaaa D=bb cc dd ....

The "A=" characters must begin each patch line of this type. They are followed by the address to be patched. The address may be specified in any of the numeric bases accepted by DOSPLUS II so long as they follow the correct format (ie, trailing "H" for hexadecimal numbers, trailing "O" for octal numbers).

Following the address and separated from it by a space are the characters "D=", followed by HEXADECIMAL digit pairs or strings enclosed in single or double quotes which represent the actual patch data. Digit pairs may be strung together or may be separated by spaces for readability. For example, if we wanted to patch a program at address 7CBFH with the digit pairs C3 08 5B, the patch lines

A=7CBFH D=C3085B and
A=7CBFH D=C3 08 5B

are equivalent. Note that the digit pairs following the "D=" do NOT have the trailing "H" character. Since only hexadecimal numbers are recognized, the trailing "H" is not used and, if present, will result in an error.

A string patch line might look like this:

A=7CBFH D='"Hello," she said.'

Note that if single quotes are used to enclose the string, double quotes may be embedded in the string, and vice versa. Whichever type is used to enclose the string, they must match or an error will be generated.

This type of patch line will result in an additional load block of code being appended to the target file which will contain the /PAT file's name (8 characters, no extension), and the patch data itself. When the program loads into memory, the patch bytes will overlay the data normally loading into the specified address. This is the reason why this type of patch line can only be used on files in load module format.

Address-oriented patches can be removed by means of the PULL parameter. The syntax for removing a patch is as follows:

PATCH file1 file2 {PULL}

where file1 is the target file, and file2 is the name of the patch file to be PULLED from the target file. For example, if the file XLOADER/CMD had originally been patched with a file called XL1/PAT, the XL1 patch can be removed by typing PATCH XLOADER XL1 {PULL}. When a patch is PULLed, the patch block is physically removed from the file, and any succeeding patch blocks are bubbled up into the space it occuppied. A successful patch removal will be signaled by the message,

Patch(s) removed

This syntax has certain implications. The first is that if more than one patch is to be applied to a file, the patches should be given different names in order to facilitate removal. The second implication, which is not obvious from the syntax, is that when PULLing a patch, the original patch file on disk need NOT be present. The PATCH utility will simply scan the target file for a load block that has the correct name and then PULL that block. If it fails to find a block with the specified name, then Patch will return the message,

Patch not found

and terminate. The target file will not be changed in any way.

SECTOR-ORIENTED Patch lines

The second type of patch line is a SECTOR-ORIENTED patch. This type of patch places the data directly into the specified locations on the disk file instead of appending it as a separate load block, and therefore may NOT be PULLed. The syntax is:

R=ss B=bb D=dd dd dd ...

The sector-oriented patch consists of a line starting with the characters "R=" followed by the SECTOR NUMBER for the patch. The sector number is followed by "B=" and the relative byte number within the sector where the patch is to begin, and finally, "D=" followed by the patch data (hexadecimal digit pairs or strings).

Sectors within a file are numbered starting from 0, and are assumed to be full 256-byte sectors IRRESPECTIVE of the file's logical record length. The sector number may be given in any numeric base acceptable to DOSPLUS II, as long as proper syntax is followed.

Bytes within a sector are also numbered starting from 0. A relative byte number greater than 255 is invalid and will generate an error. Relative bytes within sectors may also be specified in any acceptable numeric base.

The patch data must consist of either HEXADECIMAL digit pairs without the trailing "H" or strings enclosed in single or double quotes.

Patches of the sector-oriented type cannot be used to ADD sectors to a file. That is, if the record number specified in the R= field is greater than the last sector of the target file, an error will result.

Technical note : You may NOT intermix Address-Oriented patch lines and Sector-Oriented patch lines within the same /PAT file.

Patches may also be entered from the keyboard by typing PATCH followed by the filespec of the target file alone. When PATCH does not find a filespec for a patch file on the command line, it will print an asterisk ("*") on the screen. At this point you can start entering patch lines using either of the forms described above. The two types of patch lines may not be intermixed.

After typing in the last patch line from the keyboard, pressing the BREAK key will apply the patches to the target file and control will be returned to DOSPLUS II. Care should be taken when typing patches in from the keyboard, as there is no way to edit a patch line after the ENTER key has been pressed to terminate that line. Also, entering an address-oriented patch in this manner will cause it to be appended to the target file without a name, so that the {PULL} parameter may not be used to remove it should the need arise later.

Examples:

```
PATCH BASIC/CMD NEWBAS
```

The file BASIC/CMD will be patched using the contents of the file NEWBAS/PAT.

```
PATCH MONITOR XDIS,PULL
```

The patch load block with the name XDIS will be PULLed from the file MONITOR/CMD if it exists.

```
PATCH STORY/TXT
```

The PATCH utility will return with an asterisk prompt, so that the user can type patches to the file called STORY/TXT.

EXAMPLES OF PATCH FILES

```
.This is a patch to LOADER/CMD
.
A=3700H B=C3 50 50
A=5050H B=3AE5CD0208C30337
                                        <--- Carriage return here
.End of patch


.Patch to CONTRACT/TXT
R=07 B=3CH D="on this 12th day of August"
R=27 B=17H D="John B. Harrelson"
```

SVCINT SVC Interceptor

SVCINT is a utility program which ensures that all DOSPLUS II Supervisor Calls conform to TRSDOS specifications.

==========

The command syntax for invoking SVCINT is:

SVCINT

No parameters are needed.

==========

SVCINT, the Supervisory Call Interceptor, is a short program which loads into high memory. It will intercept certain supervisory routines in DOSPLUS II and make sure that they conform to TRSDOS standards. Some supervisory routines in DOSPLUS II do not act in exactly the same way as the corresponding routines in TRSDOS, and must be changed for use with certain programs. SVCINT must be run before the programs are executed.

In particular, PROFILE II+ requires the SVC interceptor routine. This is true even after it has been patched for DOSPLUS II execution. Also, VISICALC will require the presence of the SVC interceptor IF the DOSPLUS patches are NOT INSTALLED.

SYSGEN

This utility will allow you to install the DOSPLUS II system onto non-standard media such as double sided floppy diskettes or rigid drives.

==========

The command syntax is :

SYSGEN :td

":td" is the target drivespec. This informs DOSPLUS II which drive contains the disk you wish to SYSGEN.

There are no parameters with this command

==========

DOSPLUS II is shipped on an 8 inch single sided double density diskette. However, many of you are using Model 16s and hard drives and would like to install the system on either a double sided floppy, a rigid drive, or both. The SYSGEN utility allows you to do this easily.

Double sided drives can function as single sided, but the reverse is not true. The double sided drives must also have a special option that allows them to read single sided disks. All Radio Shack Model 16 drives are equipped with this option. Therefore, you can boot and use your DOSPLUS II in your Model 16.

However, in order to truly function in double headed mode, you must have a double sided system disk. Creating a double sided data disk is a simple as answering the FORMAT utility's "Number of sides" prompt with "2". Once you have created the double sided data disk, it is a simple thing to use SYSGEN to install the system on it.

We will walk you through first the double sided floppy procedure and then the rigid disk procedure.

Creating a double headed floppy diskette -

First, insert the disk you wish to install the system on into the double sided disk drive (you MUST have a double sided disk drive!). Call in the FORMAT utility (see the utility program FORMAT) and format the disk, answering the "Number of sides" prompt with "2".

After the formatter is finished with the disk, you will have a double sided data diskette. The next step is to install the system modules on that disk. You will accomplish by the statement :

SYSGEN :td

where ":td" is the drivespec of the drive containing the disk you just formatted. The SYSGEN program header will appear and the program will proceed to copy all the system modules to the specified drive.

Technical note : In order for SYSGEN to be able to copy the system modules to the specified disk, these system modules must be available on the current system disk (i.e. the one you used to boot the computer). If you have deleted any of the files with the "/SYS" extension, this program will not function.

Once the program has completed installing the system modules on the new disk, you will be returned to the DOS command mode. Now that you have installed the system modules, the next step is to install the utility files. This step IS optional. You may create a disk with just the system files on it if you wish. This disk will boot and run machine language programs perfectly well, and in some instances may be the desired item (for maximum free space).

However, if you intend to use this disk as a "working" DOSPLUS, you will more than likely need things such as BACKUP, FORMAT, and BASIC. These are all utility files and are not copied over automatically during the SYSGEN procedure. To move these files from your current master disk to the new disk just SYSGENed, you will use the COPY command (see the library command COPY). The statement will look like this :

```
COPY :sd :td {I,SPW="PASSWORD",E}
```

where ":sd" is the drivespec of the current system disk and ":td" is the drivespec of the disk you just SYSGENed. The "I" parameter tells COPY to copy even invisible files. The "SPW" parameter gives COPY the source Disk Master Password so that it can copy even the protected utility files. If you have not changed the password on your DOSPLUS master disk from when it was sent to you, then the password is "PASSWORD". The "E" parameter tells COPY to echo the filenames of the programs that it copies to the screen, so that you may see which files have been moved.

When COPY is done moving the programs, your task is complete. The disk is now a complete DOSPLUS II system diskette with all modules present. The procedure is NOT difficult or complicated. It follows three simple steps :

Format the diskette

SYSGEN the diskette (install system modules)

Copy the utility files (optional)

Any diskette that you can format, you can turn into a system disk with this procedure.

Installing the DOSPLUS II system on a rigid disk -

The procedure for installing the DOSPLUS II system on your rigid disk is much the same. The first step is still to format the drive. To accomplish this, you will use the RFORMAT utility. This program is described in the same area as FORMAT. Please note that if you are unsure of attempting this procedure, we have supplied some DO files that contain the proper instruction sets to install the system for you. The description and use of these will be discussed after we cover the manual installation technique.

Once you have formatted the drive, the next step is to install the system modules. This will be done with a statement like this :

SYSGEN :td

where ":td" is the drivespec of the hard disk you just formatted. The SYSGEN program header will appear and the program will proceed to copy all the system modules to the specified drive.

Technical note : In order for SYSGEN to be able to copy the system modules to the specified disk, these system modules must be available on the current system disk (i.e. the one you used to boot up the computer). If you have deleted any of the files with the "/SYS" extension, this program will not function.

Once the program has completed installing the system modules on the new disk, you will be returned to the DOS command mode. Now that you have installed the system modules, the next step is to install the utility files. This step IS optional. You may create a disk with just the system files on it if you wish. However, on a hard disk where space is no consideration, you should copy all the utilities to the system drive.

To move these files from your current master disk to the new disk just SYSGENed, you will use the COPY command (see the library command COPY). The statement will look like this :

COPY :sd :td {I,SPW="PASSWORD",E}

where ":sd" is the drivespec of the current system disk and ":td" is the drivespec of the disk you just SYSGENed. The "I" parameter tells COPY to copy even invisible files. The "SPW" parameter gives COPY the source Disk Master Password so that it can copy even the protected utility files. If you have not changed the password on your DOSPLUS master disk from when it was sent to you, then the password is "PASSWORD". The "E" parameter tells COPY to echo the filenames of the programs that it copies to the screen, so that you may see which files have been moved.

When COPY has finished, your task is complete. You have installed the DOSPLUS II system and utilities on the rigid drive. It can be broken down into three easy steps :

Format the drive

SYSGEN the drive (install the system modules)

Copy the utility files to the new disk

If you are using the standard Radio Shack 8.4 megabyte hard disk drive (Cat. # 26-4150/51), then all you need to do is reset the machine. The internal modification installed by Radio Shack with the hard disk will cause the system to be booted directly from the rigid drive. After that, you will simply configure each of the remaining drives to the correct settings, save your configuration file, and you are done.

If you are using a drive other than Radio Shack's, then you will need to manually configure the rigid drive as the system drive and use the floppy disk to boot up. Then you would load the proper configuration file to transfer control to the hard drive. You will accomplish this by using CONFIG. Please consult the CONFIG section of this manual for a more detailed breakdown of CONFIG in general. Specifically, the action you would take is :

```
CONFIG :0 {R,PD=0}
```

where ":0" is the drivespec of the system drive. If you have renamed this, use whatever drivespec is currently correct. The "R" parameter tells DOSPLUS II that the drive being configured is a rigid drive. The "PD=0" parameter tells DOSPLUS II that it is the rigid drive in the first position (because you CAN have more than one rigid drive attached).

All this statement has done is inform the DOS that the system drive (e.g. the drive where the system modules are located) is the first rigid drive. Although this takes effect at once, it is only a temporary change. The next time the system is reset, the configuration will return to the default. In order to make this change permanent, you need to create a "configuration file". But first, go through each drive with CONFIG and make certain that the parameters set are the correct ones.

For example, the standard Radio Shack hard disks (Cat. # 26-4150/51) have 256 cylinders. DOSPLUS II will only allow hard disk "volumes" to have a maximum of 200 cylinders. That means that your Radio Shack hard drive must be divided into at least two logical drives. You may wish to split it into four or more. At the moment, Drive 0 is the first hard disk volume. If you are going to have further volumes on the hard disk, you may wish to assign them drivespecs sequentially beginning from 0 to avoid later confusion.

If you decide to partition your hard disk into two volumes of 128 cylinders each, you could have the first volume as ":0" and the second volume as ":1". You might then call the first floppy drive ":2", etc., etc. This allows you to, when doing a global operation, access all the hard disk volumes FIRST. Then, if the situation warrants (i.e. the program or file searched for was not found), DOSPLUS II would move back and operate on the floppy drives.

Once all the drives in your system are configured correctly, the next step is to create the configuration file that stores all this. You will accomplish this via the use of the SYSTEM library command's "Save" parameter (see the library command SYSTEM). You might, for example, choose the filename "Cfig" for your configuration file. The command would then look like this :

```
SYSTEM {SAVE="CFIG"}
```

this command will search all drives looking for a file called "CFIG/CMD". If it locates one, it will overwrite it. Otherwise, it will create this file on the first available drive. In this case, that will be the first volume of the hard disk.

With the Radio Shack drive, all is well. You will be booting up directly off the hard disk, so the configuration file can and should be located on the first hard disk volume. With the other drives, this is a bit of a problem. You will be booting up from the floppy disk and loading this file to transfer control to the hard disk. Therefore, you will want this file on the floppy disk you will be using to boot up. Use the COPY command move it to that disk.

Then, immediately after re-booting the system, simply load that configuration file by executing it (e.g. typing "CFIG" and pressing ENTER). This will transfer control from the floppy disk to the hard disk. Or, in the case of the Radio Shack drive, it will properly configure the rest of the drives.

The two procedures are essentially the same. Outlined, they are :

Radio Shack drive

Reset the machine
Configure other drives correctly
Create configuration file

Other drives

Set system drive to hard disk
Configure other drives correctly
Create configuration file
Copy configuration file to floppy disk

The procedure for booting up is :

Radio Shack drive

Reset the machine and it boots from hard disk
Load configuration file

Other drives

Boot machine from floppy
Load configuration file to transfer control

The only real difference in this entire procedure is that the Radio Shack hard drive (Cat. # 26-4150/51), when installed on your machine, comes with an internal modification to allow you to boot directly from the hard disk. With the other drives, you will have to boot the computer from the floppy disk first.

Example:

SYSGEN :1

This will install the DOSPLUS II system modules on the disk located in the drive currently denoted by the drivespec ":1".

Finally:

As stated earlier, there have been some DO files provided on the disk for you that will intialize the hard disk (e.g. Format it, Sysgen it, and copy the utilities to it), and then after resetting the machine, configure the system for you and leave you only to create the configuration file.

These files will only apply under the following two conditions :

(1) You must have a Radio Shack hard disk
(2) You may only partition it into two or four logical drives

The files are :

Init2/txt	Initialize two volumes
Init4/txt	Initialize four volumes
Config2/txt	Configure two volumes
Config4/txt	Configure four volumes

You will execute these via the DO command (see the library command DO). Let's assume that you have a Model II with one internal floppy drive and one hard disk. If you desired to partition your hard disk as two volumes, you would first enter :

DO INIT2

this would call the instruction set stored in the file "Init2/txt", which would then format the hard disk and install the system on it. After the file finished, you woud be instructed to press ENTER to reset the system and then enter :

DO CONFIG2

this would call the instruction set stored in the file "Config2/txt", which would configure the remaining volumes of the hard disk and set the floppy disk drive as Drive 3. Once this file has finished you would be instructed to create the configuration file with the SYSTEM command.

If you wanted to divide the hard disk into four volumes instead of two, the same instructions apply. Simply use the "Init4" and "Config4" files instead. Even if you choose to do the procedure manually, these files provide an excellent example of the procedure. You may use the DOSPLUS II LIST command (see the library command LIST) to examine them. Viewing them may prove helpful in understanding the procedures.

Warning:

DOSPLUS II and TRSDOS do NOT use the same format on the hard disk. DOSPLUS II will not be able to read a hard disk formatted by TRSDOS. Therefore, before re-formatting the drive with DOSPLUS II, you must offload all data that you wish to preserve by using TRSDOS. Later this may be copied to the hard disk under DOSPLUS II.

Since DOSPLUS II's CONV utility cannot read TRSDOS 4.1 disks, only 1.2 and 2.0, you will have to use TRSDOS' FCOPY command to move everything to floppy disks formatted by THOSE systems (1.2 or 2.0). Do NOT move the files to 4.0 or 4.1 disks only! DOSPLUS II will not read that media later.

Remember, remove <u>all</u> important data before initializing the drive under DOSPLUS II. Once the drive is re-formatted, any previous data is <u>lost</u>.

BASIC Enhancements supplied with DosPLUS II

DOSPLUS II does not have its own BASIC interpreter. We have elected instead to patch and use the current program from your TRSDOS 2.0 series Master diskette. In order to make this transition as easy as possible for you, we have included a DO file called "BASIC/TXT" on your DOSPLUS II Master diskette to accomplish both the transfer and the patch. The actual procedure is very simple :

(1) Place the DOSPLUS II Master diskette in Drive 0 and press the reset button, booting the system.

(2) Place your TRSDOS 2.0 series Master diskette in Drive 1.

(3) Type "DO BASIC" and press ENTER.

The DO file will use CONVERT to move th file from TRSDOS to DOSPLUS II, OFFSET to set the end of file byte correctly, and APPEND to place the patch file "BASICP/CMD" on to the end of the file "BASIC/CMD". When it is finished, you will be returned to DOS. Your BASIC has been patched. A a point of reference, BASIC has been assigned the password "BASIC" during this conversion procedure.

In the case of a user who has DOSPLUS II installed on their hard disk, the procedure is a little different. You must first make certain that the drivespec ":1" is addressing the first internal floppy drive and the drivespec ":0" is addressing the first volume of the hard disk. You may accomplish this easiest by using the RENAME command.

(1) Rename whatever drive device is currently set for the first floppy drive to ":0" if it is not that way already. If the drivespec ":1" already exists in your system, you will have to rename that to something else temporarily.

(2) Make certain that the first volume of the hard disk, the one with the system installed on it, is currently named ":0". If it is not, alter it so that it is. If there is already a drive device in your system named ":0", you will have to assign it another name temporarily.

(3) Place your TRSDOS Master diskette in the first floppy disk drive (the one we named ":1").

(4) Type "DO BASIC" and press ENTER.

DOSPLUS II will do the rest, converting the file from the floppy disk to the hard disk and then processing it as stated above. The results will be the same. When you are returned to DOS after executing the DO file, BASIC will be patched.

Technical note : The BASICP/CMD file is a permanent patch. That is, the file itself is merged with the BASIC/CMD file. Therefore, once the above patching procedure is complete you may remove the file BASICP/CMD from the disk in order to provide more disk space. This file is ONLY needed at the time that you wish to patch a BASIC.

Enhancements -

The file "BASICP/CMD", in addition to simply patching BASIC to operate on DOSPLUS II, provides you with several enhacements built right in to BASIC. We provide you with several more in the form of external utility files that are used with and from BASIC. All together, they are :

Enhancement	Function
BASICP	Allows you more free memory, shorthand edit commands, lower case entry, and label addressing.
REF	Comprehensive BASIC program reference utility. Allows references of line numbers, variables, or keywords. Optional printer output.
SR	Global program editor. Allows you to make changes to all occurences of a particular string literal or expression. Optional display without alteration.
SORT	BASIC array sort program. Allows you to sort BASIC arrays of any type (integer, single or double precision, string...). Capacity for a total of THIRTY arrays. Only single dimension arrays allowed.

Internal enhancements from the BASICP patch file

Increased memory -

Once you have patched your BASIC with the DOSPLUS II enhancement package, there will be an additional 4,000 bytes of memory (approximately) available to the user.

Because of problems in TRSDOS, Radio Shack was forced to "lock out" the top area of RAM from BASIC, or there would have been conflicts with the system. DOSPLUS II does not have these problems, therefore we release this area of memory for your use.

Label addressing -

This function allows you to use indirect addressing within your BASIC programs. To accomplish this, we replaced the NAME function of BASIC with our own. To rename files from BASIC under DOSPLUS II, use the SYSTEM"RENAME" capacity.

==========

The command syntax is :

NAME label
GOTO label
GOSUB label

"NAME" assigns whatever line number it appears on the label you specify. After that, you reference the label EXACTLY as you would a line number using GOTO and GOSUB statements.

==========

Labels may now be used in place of line numbers. This frees you from having to remember the exact line number that a particular subroutine was located at. Simply assign the subroutine a unique name and reference it by that.

The only restrictions are : (1) labels may NOT contain any reserved words and (2) labels may not exceed 240 characters in length.

We have also altered BASIC's RENUM function such that it will not regard labels when renumbering a program.

Please note that a label may appear anywhere in a line. For example :

```
10 X=1:NAME TEST:FOR A=1 TO 10
20 Other program here ...
100 GOTO TEST
```

will work just fine.

Examples:

```
10 CLEAR 1000 : DEFINT I
20 NAME START
Other program lines ...
1000 GOTO START
```

In this example, line 20 has been assigned the label "START". Later, at line 1000, the program issues the command to "GOTO START". This would send program control back to line 20.

```
10 NAME DEFFNTESTER
```

This is an example of an invalid label. This label contains the reserved word "DEFFN". BASIC will reject this as a label.

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
left arrow	List first line of program
/ (Slash mark)	List last line of program
right arrow	List last line of program
down arrow	List next line of program
up arrow	List preceding line of program
! (excl. mark)	Abbreviation for SYSTEM (!"DIR")
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.

Technical note : 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.

REF

This utility provides you with a comprehensive, powerful BASIC program cross-referencer capable of referencing line numbers, keywords, or variables. It can also reference single occurences of the same and includes optional output to the line printer.

==========

The command syntax is :

SYSTEM"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) or any combination of the above including a single reference to a particular line number, variable, or keyword. For example :

SYSTEM"REF",K,L,V

This will reference the program for all three. If you specify a P also (SYSTEM"REF",K,L,V,P), it will do the same thing, but it will output the result to the line printer.

To display a single variable you use the "S" parameter. For example :

```
SYSTEM"REF",S,A
```

Every time the variable "A" occurs in the text will be listed for you. The single variable reference becomes as specific as you are. For example :

```
SYSTEM"REF",S,A$
```

will only hunt up references to the variable "A" when it is being used as a string variable. And still further :

```
SYSTEM"REF",S,A$(
```

will look up only references to "A" as an ARRAY string variable. It will also take complex variable names like :

```
SYSTEM"REF",S,FINDIT
```

This will look for all occurences of the variable "FI".

The same syntax applies for a single line number or a single keyword. For example :

```
SYSTEM"REF",S,PRINT
```

Will reference all the PRINT statements.

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.

SR

This utility program provides you with a method of displaying strings in text and optionally replacing them. This function works with string literals, string expressions, or a combination of both. You have the ability to specify the starting and ending line number in order to restrict the action to only a certain portion of the program.

==========

The command syntax is :

SYSTEM"SR",sexp,rexp,sln-eln

"sexp" is the search expression. This can be any valid string expression, a literal, or a combination of both string variables AND literals.

"rexp" is the replace expression. This can also be any valid string expression, literal, or combination.

"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 SYSTEM"SR" (for search and replace) followed by a literal ASCII string, OR any character string or other string variable.

After it alters a line, it will list that line showing the change. It operates in two modes : Search mode and Search and Replace mode.

For example, if you type

```
SYSTEM"SR","Test"
```

It will look through the text and list every line with the word "Test" in it. If you type :

```
SYSTEM"SR","Test","NewTest"
```

It will look through the text and every time that it finds the word "Test", it will replace it with the word "NewTest".

If you type :

```
SYSTEM"SR","Test","NewTest",100-200
```

It will confine this procedure to lines 100 through 200. 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 :

```
SYSTEM"SR",":",CHR$(10)+":"
```

This will go through the whole text and insert a line feed in front of every colon.

SORT

This utility program provides you with a fast, powerful BASIC array sort. It can sort up to thirty arrays, keying on up to ten. The arrays *MUST* be single dimension but array type (string or numeric) doesn't matter. You may define any of the key arrays as being sorted in ascending or descending order.

==========

The command syntax is :

SYSTEM"SORT",exp,+ or - AN(se)+KA-KA,TA,TA

",exp," expression to indicate number of elements to be sorted (integer)

"+ or -" indicates primary key array to be sorted in ascending or descending order. Optional, if omitted ascending order will be assumed.

"AN(se)" primary key array. Subscript indicates starting element number.

"+KA" next key array. Plus (+) indicates ascending order.

"-KA" next key array. Minus (-) indicates descending order.

",TA" First tag array.

",TA" Next tag array.

==========

A "key" array is defined as being an array that SORT will consider when sorting. A "tag" array, on the other hand, is simply "along for the ride". When SORT finds two elements of a key array that need to be swapped, it will swap the corresponding elements of all other key arrays and all tag arrays.

You 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, SYSTEM"SORT" 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.

Examples:

```
SYSTEM"SORT",100,A$(1)+B$-C$,D$,E$,F$
```

This command line would instruct SORT to sort 100 elements of string array beginning with the first element in the array "A$". If it finds a match there, it will attempt to sort by the corresponding two elements in "B$". If it finds a match there, it will sort by the corresponding two elements in "C$". However, "C$" is sorted in descending order. Any time that it swaps an element in any of the key arrays, it swaps it in all the other 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.

```
SYSTEM"SORT",N%,A$(1)
```

This will sort "A$" in ascending order starting at element one and proceeding for "N%" elements.

In DOSPLUS II's SORT, you may indicate ascending or descending order on a tag array. Also, SORT allows you up to ten key arrays (counting primary key array) and twenty tag arrays for a total of THIRTY arrays.

Technical note : This sort uses the "high" system overlay area of memory. Use caution so that you do not cause memory conflicts with other library commands and/or utilities. Please note that when calling SORT, you may not specify a starting element number for any array other than primary key array. All other key arrays will be assumed to start at the same relative position.

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$:RN%(I)=LOC(1):NEXT I
70 CLOSE
80 SYSTEM"SORT",EF,A$(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.

Introduction to the DosPLUS II Technical manual

The DOSPLUS II technical manual is not for everyone. It is designed to be a reference area for the programmer that needs specific information about the system in order to facilitate interfacing programs to it.

The technical manual is logically organized from simplest to most complicated. The first portion, **Introduction to hard disks,** is designed to acquaint anyone who is not familiar with DOSPLUS II to the method behind our hard disk organization.

Following that section are sections on **System organization** (containing a breakdown of the various system files and their functions), **Directory Structure** (giving a detailed breakdown of both the overall directory organization and a byte for byte discussion of a single directory entry), **Device Control Blocks** (giving a byte by byte description of the DOSPLUS II DCB organization), and **Drive Control Tables** (explaining what information is stored regarding the disk drives).

Those areas of information will prove to be interesting reading to the programmer or experienced user that is seeking to learn more about the manner in which DOSPLUS II functions internally.

The next area addressed is the DOSPLUS II **Supervisory Call System.** These Supervisor calls (SVCs), for the most part, are compatible with TRSDOS. In some areas, though, we have added a new call or improved upon an existing one. We strongly suggest that any machine language programmer planning on writing programs for DOSPLUS II go through these.

Wrapping up the technical manual is an explanation of the DOSPLUS II **Resident Jump Table** and the **Resident Disk I/O system.** The last item in the manual is a list of the DOSPLUS II error codes and messages. Included in the Resident Jump Table is a discussion of how to calculate the Logical File Number (LFN) for a particular directory entry.

This area of the manual is not intended to confuse the first time computer user. It is not intended to mystify the novice. We have carefully separated it from the User's manual so that the users who do not wish to concern themselves with technical information don't have to.

We hope that you find this manual both informative or useful. If there is anything that you feel we have left out, please don't hesitate to get in touch with our technical support teams and ask. Special thanks go to Dr. Renato Reyes of PowerSOFT for his excellent work in this area of the manual.

Table of contents

Introduction to hard disk operation

This section of the manual is designed to acquaint you with the principles of hard disk operation from the user's viewpoint. We believe that if you understand how your hard disk works, you will be able to configure yours for the optimum effeciency later on. Although this section of the manual is not required reading for anyone not using DOSPLUS II to operate a hard disk, we recommend that all users read it anyway to glean an understanding of hard disk operation and to see what the differences are between a cylinder and a track.

We will cover two areas :

(1) Operation of hard disks.

(2) Cylinder vs. Track.

Operation of hard disks -

DOSPLUS II operates with a variety of controller boards and hard disk drives, including (of course) Radio Shack's own unit. CONFIG allows you to set all the needed parameters for your particular drive, but we felt that a more detailed description would assist.

Each hard disk is comprised of "platters". The number of platters in your hard disk is referred to as that hard disk's "platter count". Picture each platter as something similar to a phonograph record. Each platter has two recording areas, top and bottom. These areas are called "surfaces".

Each surface has a certain number of tracks it can format and read/write to. This "track" is the same track that you have grown used to in floppy drives. For every track on the top surface, there is a corresponding track on the bottom surface. This is true for each platter. Therefore, a two platter hard disk would have FOUR track zeros. Track zero on the top surface of the first platter and track zero on the bottom surface, and again for the second platter.

Each surface has its own read/write "head". The read/write head is the part of the drive that actually travels over the surface and "reads in" or "writes out" data. The number of heads for each hard disk will be equal to the number of surfaces. And since we already know that the number of surfaces is equal to (platter count * 2), we know that head count can be acertained via the same formula. This would mean that the head count for a two platter drive is four. This will be important later because CONFIG will determine the size of the hard disk by its head count.

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 what we are attempting to achieve in data processing. This brings us to "cylinders".

Obviously, to read and write to all the tracks on one surface and then begin another surface would be terribly inefficient use of the drive and result in incredible head travel and a slow system. Since all heads step in and out together, when the head for the top surface of the first platter is positioned over track zero, ALL the heads of ALL the platters are over their track zeros also. To allow you reduce head travel, DOSPLUS II allows you to define cylinders.

A cylinder is defined as: all correspondingly numbered tracks starting with a user-defined head and proceeding sequentially for a user-defined number of surfaces. That is on a hard disk. On a double-headed floppy disk, the user may not define these values (because there are only two heads), and the definition becomes: all correspondingly numbered tracks on all available read/write surfaces.

Therefore, by defining our hard disk as "HC=4" (4 heads) and "HO=0" (no head offset or "start with the first actual head"), we define each cylinder as having four tracks within it. DOSPLUS II will write until it fills a cylinder before it advances to the next cylinder. In our example, this cuts head travel to 1/4 of what it was before.

Because these drive sizes and types vary so widely, DOSPLUS II has to be VERY flexible. That is why so much information is required with CONFIG. You must tell us how many heads your drive has, how many tracks there are on a surface, and how you want your cylinders arranged. As explained earlier in this manual, you have eight drive (file orientated) devices. You may assign as many of those as you wish to the hard disk.

When you CONFIGure a drive (see the library command CONFIG) as being a floppy disk drive, DOSPLUS II will restrict you from assigning any more drive devices to that physical drive. This is not the case with the hard disk. You may partition your hard disks with several or even all of your drive devices. This practice is recommended for those that need more directory space, because each partition has its own directory and may in that manner increase the number of actual disk files that you are able to store on your hard disk.

There are two methods of configuring the hard disk into several partitions under DOSPLUS II. They are :

(1) Fewer but larger cylinders.

(2) More but smaller cylinders.

There are advantages to each method. Generally, however, you may follow these guidelines for the most effecient disk usage and acess times. If you are going to be using many smaller files; use method (2). If you are going to have fewer files, but they will be larger, ever-changing data files; use method (1).

Method (1). In this method, you will define your cylinders to include ALL read/write heads. This will make the individual cylinders just as large as possible. You will leave the head offset parameter set to "0" and break up the drive by assigning each drive certain cylinders on the disk. Let's take an example. Say that we have a two platter, 153 cylinder hard disk.

To break it into two equal units using method one, first we select which two drive devices we wish to assign to the hard disk. Then, on each of those, we would set the CONFIGuration as rigid and point the PD (physical drive) parameter to whichever hard disk we are referring to. After that, again for each of them seperately, we would set the HC (head count) parameter on CONFIG to "4" (two platters = four heads). We would leave the HO (head offset) parameter set to "0" (begin with the first head). The next step is to decide how many cylinders we want each logical drive to have. Since 153/2 is 76.5, one drive will have 76 cylinders and one 77. Let's assign the front drive 76 cylinders. That means whichever of our two logical drives we select to be the first part of the hard disk will run fom cylinders 0-75 (76 in all). Therefore, the next partition will start at cylinder 76 and run to cylinder 152 (77 in all). On the first drive, CO (cylinder offset) should be set to "0" (as in start with the first cylinder). For the second drive, however, it should be "76" (as in start with cylinder 76).

When we call up RFORMAT to format the drives, we reference them by their drivespecs (which are still as they were even though both of them are now pointing to different areas of the same physical drive unit) and format the first logical drive for 76 cylinders and the second for 77. When the format is complete we have two equal (or very nearly so) partitions for the hard disk.

Method (2). In this method, you define your cylinders such that any one cylinder does not encompass ALL read/write heads. This can mean that each read/write head is a separate drive or that several read/write heads can be considered a single drive unit. This will make the individual cylinder small, but you will have a much greater number of cylinders. In this method we will make extensive use of the HO (head offset) parameter. Let's take our example again, a two platter, 153 cylinder drive and split it into two equal units. This time, though, we will use method 2.

To begin, first we select which two drive devices we are going to use for this hard disk. Using CONFIG, we define each as rigid and point the PD (physical drive) parameter at whichever hard disk we are referring to. We know that the drive has four read/write heads because it has two platters (remember, head count = platter count * 2). Since it is going to be split in two parts, each logical drive will get two heads. That's where the head offset parameter comes in. The first logical drive would get heads 0 and 1 (the first platter) and the second would get 2 & 3 (the second platter).

For the first drive, you would set the HO (head offset) to "0", telling DOSPLUS II to begin with the first head. Then set the HC (head count) parameter to "2", telling the system that this logical drive starts with head 0 and extends for two heads. For the second drive, you would set the HO (head offset) to "2", telling DOSPLUS II that you wish to begin this logical drive with the third head (head 2). The HC (head count) parameter would also be set to "2", indicating that this drive will extend for two heads also.

When we call up RFORMAT to format the drives, we reference them by their drivespecs (which remain unchanged from when we started) and format each logical drive for the full cylinder count of 153. Because each head is an independant drive, all cylinders are used. Once again, when the format is complete, we have two equal partitions for the hard disk.

The only difference is the manner in which DOSPLUS II will allocate space on the hard disk using each of the methods. As stated earlier, for maximum efficiency use method 1 for larger data files that will be ever-changing. When you have many smaller files that will remain static, use method 2.

Cylinder vs. Track -

From the above explanations, you should have a fair idea of what a cylinder is (at least in how it pertains to the hard drives). We will now cover the relationship between a "cylinder" and a "track" as it pertains to the floppy disk drives.

On a single headed floppy disk drive, a cylinder is the same exact thing that you are used to in a track. Those of you who are using single headed drives need not concern yourself with this other than to remember the new term and what it means.

However, on double headed, the cylinder functions as it did on the hard disk. It uses the corresponding track on the second side of the disk. DOSPLUS II views a double headed drive as a single volume. Which is to say that no special syntax for denoting which side of the drive you wish to address is needed with DOSPLUS II. When a diskette is formatted as double headed, it is formatted with the cylinders twice as large. By adopting this system, we have enabled you to easily make use of the increased storage of double headed disk drives with a minimum of software adaptations and headaches.

Whether or not a disk is double headed is set during the time you format the disk. When you call up FORMAT, it will ask you whether the disk is single or double headed. You respond with whatever the case is.

When FORMAT writes the system information to the disk, it writes what we call a DCT or <D>rive <C>ontrol <T>able. This DCT contains information about the disk that the DOS will use when it is accessing it. For the most part, this is absolutely invisible to the user. However, if you are switching from a single to double sided (or vice versa) disk in the same drive, you will need to force DOSPLUS II to re-read the DCT and pick up the new configuration to prevent errors later. This done via the I command (see the library command I).

If you want to create a double headed system disk, use the SYSGEN utility (see the utility program SYSGEN).

Finally -

Please do not be intimidated by all this information. These items should be totally invisible to the end user of the system (if the programs are written correctly). DOSPLUS II will come with these various areas pre-configured to operate standard hardware in the most efficient manner possible. However, there is an ever growing number of excellent suppliers of "non-standard" hardware emerging.

We want to support just as many different types of these manufacturers as possible with DOSPLUS II. In order to do that, we had to make the system as flexible as possible. We have done that. Our purpose in explaining what we have explained above is to acquaint the experienced user, programmer, or consultant with the manner is which we handle drive operation. If you need it, the power is there.

System Organization

The DOSPLUS II operating system consists of a resident module, which is always in memory, and a number of overlays which are called in to perform particular functions as needed. The use of overlays minimizes the amount of memory required by the system without sacrificing power or flexibility.

There are two overlay regions in DOSPLUS II, called the low region (1C00H - 21FFH) and the high region (2200H - 27FFH). Most supervisory call routines execute from the low region, and most library command routines execute from the high region. Generally, user programs which do not perform any calls to a library command routine via the DOSCMD or RETCMD Supervisory Calls may make use of the high overlay region; however, it is not recommended that the low overlay region be occupied by user programs.

The resident system, SYS0, occupies memory from 0000H to 19FFH. This module contains the routines which are required for proper operation of the system, including the default driver programs for each of the system devices, the Resident Disk I/O system, and the overlay handler, which loads the correct overlay from disk into memory as needed.

At 1A00H through 1AFFH is a 256-byte buffer area which is available to user programs. Immediately following it at 1B00H is the system buffer area, which should NOT be used by programs external to the operating system.

The first eight system modules (SYS1 through SYS8) occupy the low overlay region and therefore cannot be in memory concurrently. These eight modules contain the Supervisory call routines and the file I/O routines, among other things.

The next 8 modules (SYS9 through SYS16) load into the high overlay region. These contain the routines necessary for executing the library commands of DOSPLUS II.

SYS1 contains the system command interpreter and filespec evaluator routines. This overlay is called in to parse and interpret commands typed in at the DosPLUS II prompt.

SYS2 contains the necessary routines for OPENing and INITializing files, RENAMing files, encoding file passwords and generating file hash codes.

SYS3 contains the routines for CLOSing, KILLing and PURGing devices and/or files.

SYS4 is responsible for allocating disk space to a file's extents. It contains routines for searching a directory and creating extended directory entries for a file.

SYS5 contains the system's error messages.

SYS6 contains the routines for the system's DEBUGger.

SYS7 contains code to initialize the system's devices along with routines for printer and communications channel control.

SYS8 contains the parser routine for evaluating command lines and wild card mask specifications.

SYS9 is the first of the high-region overlays and contains the following library commands: DIR, CAT and FREE.

SYS10 consists of routines for channel-directed I/O. The library commands contained in this module are COPY, APPEND and LIST.

SYS11 contains the device management routines. It embodies the following library commands: ROUTE, RESET, SET, FILTER and LINK. The routines for displaying device status are also in this overlay,

SYS12 contains miscellaneous library commands and ancillary routines. The commands in this module are VERIFY, TIME, I, ERROR, DEBUG (Debugger On/Off -- the actual DEBUG code is in SYS6), DATE, AUTO, CLOCK, LIB, PAUSE, and SCREEN.

SYS13 contains initialization routines for the communications channels and printer, along with the code for executing the DO command. The library commands contained here are SETCOM, FORMS and DO.

SYS14 holds the file protection management routines. The library commands here are PROT, ATTRIB and KILL.

SYS15 contains other miscellaneous file management subroutines and library commands, namely CLEAR, CREATE, DUMP, BUILD, LOAD, and RENAME.

SYS16 contains system setup routines. The library commands handled by this module are CONFIG and SYSTEM.

All of the modules must be present on the disk in drive 0 (or the currently defined system drive) for proper DOSPLUS II operation. If one or more are missing or damaged, operation of the system can be very unpredictable.

The following are fixed memory locations in the resident module (SYS0) which may be accessed by user programs:

DCBTBL - The Device Control Block location table

This table starts at 0100H and consists of 15 contiguous 4-byte blocks for each device. The first two bytes of each block contain the address of the device's DCB in LSB/MSB format. The next two bytes of contain the two-character device name ("KI", "DO", "PR", etc.)

RTCTBL - The Real-Time Clock Interrupt Service Table

This table is located from 0140H through 014FH and consists of 8 2-byte entries. Each entry will hold the address of the Task Control Block (TCB) of an interrupt task to be executed under control of the system's real-time clock. The first two slots (0140H-0141H and 0142H-0143H) are available for user-defined tasks. The others are used by the system and should not be pre-empted for user tasks.

Insertion and removal of user-defined tasks into the table are detailed under the Supervisory Call section (SVC 92, NMICTL). Use of this supervisory call is recommended over simply inserting data directly into RTCTBL.

TIME$ - Time/Date locations

This block is located at 0150H through 0156H and contains the time and date data for the system. They are arranged as follows:

0150H	ticks
0151H	seconds
0152H	minutes
0153H	hours
0154H	day of month
0155H	month
0156H	year (0 represents 1980)

FLAGK1$ - System flag byte

This byte is located at 0167H and is used as follows:

bit 7	unused	
6	unused	
5	unused	
4	unused	
3	verify	1=On, 0=Off
2	debug flag	1=On, 0=Off
1	force read	1=On, 0=Off
0	unused	

OSVER$ - Operating System Version

This byte is at 016AH and encodes the operating system version number. The version number is maintained in binary coded decimal. The high 4 bits will have the actual VERSION number, and the lower 4 bits will contain the RELEASE number. It currently contains 10H, for version 1.0 (ie., version 1, release 0).

LOMEM$ - Lowest free memory address

These two bytes at 0171H-0172H encode the lowest available free memory address.

HIMEM$ - Highest free memory address

These two bytes are at 0173H-0174H and encode the highest available free memory address in LSB/MSB format. User programs should not use memory above this address.

PHYMEM$ - Top of physical memory

These two bytes at 0175H-0176H encode the highest physical memory address. In a 32K Model II they will contain BFFFH; in a 64K Model II they will contain FFFFH.

DOSPLUS II Directory Structure

The necessary information for accessing files on a diskette, or on a logical drive in a hard drive system, is maintained in a directory. In DOSPLUS II, each diskette, and each logical drive on a hard disk, contains one directory cylinder. The directory can be a maximum of 34 sectors, and the maximum number of files it can hold information on is 256. The directory is always within one cylinder, and is never split up. Also, unlike TRSDOS, DOSPLUS II does not maintain an alternate directory.

The directory cylinder can be thought of as consisting of three distinct sections: the Granule Allocation Table (GAT), the Hash Index Table (HIT), and the directory records proper. All three sections are used by the operating system when accessing a file, or allocating space to a file.

THE GRANULE ALLOCATION TABLE (GAT)

The Granule Allocation Table, or GAT, is located in the first sector of the directory cylinder and contains information pertaining to the allocated and available space on the diskette or logical drive. It also contains information as to how many cylinders have been formatted.

Each disk is formatted into cylinders (tracks=cylinders on single-sided disks), containing a specific number of sectors. The sectors on each cylinder are grouped into arbitrary units called GRANULES, or grans. The size of each gran (that is, the number of sectors in each gran) varies according to the media that the operating system is dealing with. In any case, when allocating space to a file, DOSPLUS II will always allocate a minimum of one granule. Small files may be assigned more space than they require, but disk search time is minimized.

Each byte in the GAT table represents one cylinder. The byte value is actually a bit map of the granules in that cylinder, so that each cylinder then can contain a maximum of eight granules. Each bit that is set to 1 represents a granule that is in use or otherwise unavailable, and each bit set to 0 represents a granule available for allocation. Bit 0 represents the first granule on the cylinder, bit 1 the next, and so forth.

Depending on the formatting characteristics of the media in question, the number of bits in each GAT byte actually used by the system will vary. For example, a standard 8" double density, single-sided diskette has 6 granules of 5 sectors each. This means that only bits 0 through 5 of each GAT byte will be used, and bits 6 and 7 will remain permanently set to 1 to indicate that they are not available.

In floppy disk media, relative bytes 00H through 5FH (0 through 95) of the GAT table contain the free/allocated granule information for up to a maximum of 96 formatted cylinders. The relative byte numbers correspond to the cylinder numbers.

Relative bytes 60H through BFH (96 through 191 decimal) contain the lock-out information for each cylinder and granule on the diskette. The arrangement is precisely the same as that of bytes 00H through 5FH. In this table, any bit set to 1 indicates that that granule is unavailable for use. If the formatting process detected any flawed granules or cylinders, they will be mapped into this table. This table is also used by the system during the backup process to determine if both source and destination disks have the same capacity.

In hard drive configurations, bytes 00H through CAH (0 through 202) are all used for the free/allocated space table. Hard drive directories do not make use of a lockout table. Any flawed grans found during format time are simply allocated.

Relative byte CBH (203 decimal) of the Granule Allocation Table contains the version number of the operating system. In the current release of DOSPLUS II this will contain 10H, representing version 1.0. This number is kept in BCD (binary coded decimal) form. The digit represented by the leftmost 4 bits is the version number, and the digit represented by the right 4 bits is the release number.

Relative byte CCH (204 decimal) contains the formatted track count of the diskette in excess of 35 tracks, and is used to minimize the time spent in computing the number of tracks on a disk. On a standard 8" single-sided disk, this byte will contain 2AH, or 42 decimal (42+35 = 77).

Relative byte CDH (205 decimal) contains information pertaining to the format of the diskette. Bit 6 indicates density. If set to 1, then the diskette is formatted in double density. Bit 5, if set to 1, indicates that the diskette is double-sided. Bits 2 through 0 encode the number of grans per cylinder MINUS 1.

Relative bytes CEH and CFH contain the encoded master disk password. The encoded password is 12 bits in length.

Relative bytes D0H through DFH contain the diskette name and date information assigned to it at format or backup time, in two adjacent 8-byte fields. Note that DOSPLUS II does not require a date; therefore the second 8-byte field (the "date" field) can actually contain any ASCII information.

Relative bytes E0H through FFH are used to hold any AUTO command for the diskette in question. If no auto command is present then byte E0H will contain 0DH (13 decimal).

THE HASH INDEX TABLE (HIT)

The Hash Index Table, or HIT, occupies relative sector 01 of the directory cylinder. This sector contains hash codes of each active file in the directory, positioned in such a manner as to code the location of the file's directory record in the following sectors. This permits the operating system to locate a particular directory record in a minimum amount of time and with a minimum number of disk accesses.

When a file is created, its name and extension are processed by the operating system through a special hashing algorithm which produces a one-byte value in the *range 01H through FFH (the value 00H is used to indicate an available HIT position).* The hash code is then stored in the HIT table in a position which corresponds to the location of the main directory record. Since duplicate hash codes are possible, it may be necessary for the operating system to make several scans of the HIT table for the hash code of a file. After finding each, the operating system will go to the main directory record and compare the filename with the filename in memory for a match. If a match is found, then the scan stops. Otherwise the system returns to the HIT table for another try.

The position of a file's hash code in the HIT is called the Directory Entry Code (DEC) or Logical File Number (LFN). Each active file has at least one DEC. Files which are long enough to require extended directory entries will have a DEC for each extended entry. If the HIT table is arranged into 8 rows of 32 bytes each, then the position of each byte determines the position of the directory entry record. Each HIT row corresponds to the position of each 32-byte entry in the succeeding sectors, and each column position corresponds to the actual sector number minus 2 where the directory entry may be found. This design permits a maximum of 256 files on any one disk drive.

The position of the directory record may also be calculated from the relative byte position of its HIT code. The leftmost three bytes of the DEC (the relative byte position of the file's hash code) translates into the file's relative position within a sector. The rightmost 5 bits translates into the directory sector minus 2 where the record for that file may be found. Thus, a file with a HIT code at position 26H (001 00110) of the table will have its main directory entry at position 01 of sector 8 (6+2) of the directory. Relative position 01, of course, starts at byte 20H (32 decimal).

The number of DEC bytes actually used by the operating system for HIT entries will depend on the formatting characteristics of the media again. For example, a 5.25" minifloppy disk formatted in single density will only use the first 8 columns of each row of the HIT table, while a logical drive on a hard disk may make use of every possible entry in the table.

THE DIRECTORY RECORDS

The information for each file on a disk is contained in its directory records. DOSPLUS II uses 32-byte directory records, which means that each directory sector following the GAT and HIT sectors can contain 8 directory entries, at relative bytes 00H, 20H, 40H, 60H, 80H, A0H, C0H and E0H. Each active file has a directory record which holds, among other things, its name and extension, the hash codes of its access and update passwords, its protection level, the actual location of the file on the disk, and the number of extents the file has. Each file may have two types of directory record: a Primary record, also known as the File Primary Directory Entry or FPDE, and an extended record, known as the File Extended Directory entry or FXDE. An FXDE is created for a file when the primary entry runs out of space in which to code the file's segments.

The structure of a directory record in DOSPLUS II is as follows:

RECORD + 0	Flag byte 1	
	Bit 7 - 1=FXDE entry,	0=FPDE entry
	6 - 1=System file,	0=User file
	5 - 1=Non shrinkable,	0=Shrinkable
	4 - 1=Active file,	0=KILLed file
	3 - 1=Invisible file,	0=Visible file
	2 - 0 - Protection level, 0 - 7.	
+ 1	Flag byte 2	
	Bit 7 - Reserved for future use.	
	6 - 1=File modified	0=Not modified
	5 - 4 - Reserved for system use	
	3 - 0 - Month updated	
+ 2	Flag byte 3	
	Bit 7 - 5 - Year updated (1980 base)	
	4 - 0 - Day updated	
+ 3	EOF byte	
+ 4	Logical Record Length (0=256 bytes)	
+ 5-12	Filename, left justified and padded with blanks as necessary	
+13-15	Extension, left justified and padded with blanks as necessary	
+16	Update password hash code (LSB)	
+17	Update password hash code high nybble (bits 7 to 4)	
	Access password high nybble (bits 3 - 0)	
+18	Access password hash code (LSB)	
+19	EOF sector MSB	
+20	EOF sector LSB	
+21	EOF sector NSB	
+22-31	Segment descriptor list.	

The structure of the FPDE and FXDE are identical, except that an FXDE does not make use of bytes 2 through 21. Also, in an FXDE, byte 1 of the record is a backward linking DEC which points to the preceding directory record for this file. No filename or extension is encoded in an FXDE.

A detailed explanation of the directory record structure is given below.

RECORD + 00

This is the first of three flag bytes maintained for each directory record. Bit 7, if set to 1, indicates that this record is a File Extended Directory Entry (FXDE). This means that the file is large enough to require more than one directory record entry in order to properly code the position of all of its segments on the disk. Bit 6, when set to 1, indicates that this file is a SYSTEM file, i.e., part of the DOSPLUS II operating system itself, or a file which has been given system status.

Bit 5 indicates whether or not a file is of the non-shrinkable type. DOSPLUS II allows the user to create files with a certain minimum preallocated size, and indicate to the system whether the file can fall below the minimum or not. If this bit is set to 1, then the file cannot be reduced to a size below its minimum allocated size at creation.

Bit 4 flags whether the file is active or killed. DOSPLUS II does not remove the directory entries of KILLed files from the directory, but instead, merely resets this bit. This permits recovery of an accidentally killed file, if necessary.

Bit 3 indicates whether or not a file has the Invisible attribute.

Bits 2, 1 and 0 code the file's protection level, as follows:

7 - No access
6 - Execute only
5 - Read, execute
4 - Write, read, execute
3 - Unused
2 - Rename, write, read, execute
1 - Kill, rename, write, read, execute
0 - Full access

RECORD + 01

This is the second of the three flag bytes, and encodes the following conditions:

Bit 7 is reserved for future use.

Bit 6 indicates whether the file has been modified since its creation or backup. The COPY library command and the BACKUP utility will change the setting of this bit.

Bits 4 and 5 are reserved for system use.

Bits 3 - 0 encode the month the file was updated.

RECORD + 02

This is the third flag byte. Bits 7 through 5 encode the year that the file was updated, with 0 being equivalent to 1980.

Bits 4 through 0 encode the day of the month that the file was updated.

RECORD + 03

This is the EOF byte pointer. It points to the next available byte of the file proper in the sector pointed to by bytes 19,20 and 21.

RECORD + 04

This byte encodes the logical record length of this file, from 1 to 256 bytes. A zero value implies a 256-byte LRL.

RECORD + 05 through + 12

These eight bytes contain the filename. The name is left-justified in the field, and padded on the right with blanks if necessary.

RECORD + 13 through + 15

These three bytes contain the file's extension, left-justified in the field and padded on the right with blanks if necessary.

RECORD + 16, +17

Byte 16 plus the leftmost 4 bits (the high nybble) of byte 17 form the 12-bit hash code of the file's UPDATE password. Byte 16 is the LSB of the update password hash code, while bits 7 through 4 of byte 17 form the HIGH nybble of the hash code.

RECORD + 17, +18

The rightmost 4 bits of byte 17 (the low nybble) plus byte 18 form the 12-bit hash code of the file's ACCESS password.

The password hash codes are formed by passing the ASCII password (all upper case) through the the standard encoding algorithm used by TRSDOS and most other TRS-80 Models I and III disk operating systems. The algorithm produces a two-byte hash code. The final value of the code is arrived at by stripping the leftmost 4 bits of the hash code's MSB, to form the 12-bit code used by DOSPLUS II. This permits both update and access passwords to be encoded in two bytes (24 bits), thus freeing up one byte for use as part of the three-byte EOF record pointer.

RECORD + 19, +20, +21

These three bytes form the 24-bit pointer to the EOF sector of the file. The use of a three-byte pointer permits the creation of files larger than 65535 records in length. Byte 19 is the MSB of the pointer, byte 21 is the NSB, and byte 20 is the LSB. The maximum file size possible under DOSPLUS II is 16 million records.

RECORD + 22 through + 29

These bytes form the segment descriptor list for the file. Segment descriptors consist of 2 bytes each. The first byte contains the number of the cylinder where this segment starts. The low 5 bits of the second byte contains the quantity of contiguous granules in this segment in zero relative order (that is, a "0" implies 1 gran, "1" implies 2 grans, etc.). The high 3 bits of this byte contain the granule of the cylinder where this segment starts (files do not need to start at the first granule of each track every time). An FF byte in the segment descriptor list indicates the end of the list.

Using this scheme, then, each file segment can be thought of as consisting of up to 32 contiguous granules.

RECORD + 30, +31

These two bytes are used to flag the presence of an extended directory entry, as well as its position in the directory. If the value of byte 30 is FFH, then no extended directory records exist for this file. If the value of this byte is FEH, then an extended directory entry exists, and byte 31 contains the DEC of the extended directory entry.

File Extended Directory Entries (FXDE)

If a file occupies more than 32 contiguous granules, then a File Extended Directory Entry is created for it. The FXDE is very similar in structure to the primary entry, except that (a) no filename is entered for it; (b) only bits 7 and 4 of relative byte 0 of the entry are used; and (c) relative byte 1 of the entry is not a flag byte but a backward link to the previous entry. This byte will contain the DEC of the previous directory entry for this file. If a file has only one FXDE, then this byte will point to the primary entry.

Relative bytes 22 through 31 of an FXDE are identical to that of a primary entry, or FPDE. A file may have as many FXDEs as required to map it out in its entirety.

DEVICE CONTROL BLOCKS

The various physical devices recognized by DOSPLUS II as being a part of the computer system are interfaced to the operating system via Device Control Blocks, or DCBs. Each device is assigned a DCB. The DCB is a contiguous segment of memory which may be up to 40 bytes in length. The location of a particular DCB can be gotten by means of the LOCDCB routine in the Resident Jump Table. Users should not modify any DCB directly, however; there are Supervisory Calls for this purpose. Modifying a DCB indiscriminately may cause the system to crash.

The current release of DOSPLUS II has DCBs defined for the following devices: @KI, @DO, @PR, @CA, @CB, @U1, @U2 and @U3 (devices 0 through 7, respectively). The size of each DCB varies according to the requirements of the physical device itself.

Part of each DCB is used for data common to all devices. These locations are fixed relative to the start of the DCB. Additional bytes are used for data unique to the device, if needed. The common DCB data are defined as follows (@CTL, @GET and @PUT refer to character-oriented I/O operations):

DCB	+ 00	Bit 7: File DCB: 1=Yes, 0=No
		6: DCT/DCB: 1=DCT, 0=DCB
		5: DCB is Linked: 1=Yes, 0=No
		4: DCB is Routed: 1=Yes, 0=No
		3: DCB is NIL: 1=Yes, 0=No
		2: @CTL valid: 1=Yes, 0=No
		1: @GET valid: 1=Yes, 0=No
		0: @PUT valid: 1=Yes, 0=No
	+ 01,02	Driver address (LSB/MSB)
	+ 03,04	Linked/Routed DCB address, if any (LSB/MSB)
	+ 05	Flag byte
		Bit 7 through 1: dependent on device
		Bit 0 - Translate: 1=Yes, 0=No
	+ 06,07	Translation table address, if any (LSB/MSB)
	+ 08	Translation table length
	+ 09,10	Buffer start address (LSB/MSB)
	+ 11,12	Spool buffer size
	+ 13,14	Offset for adding data to buffer
	+ 15,16	Offset for taking data from buffer

A detailed explanation of each of these bytes follows.

DCB +00

This byte is a flag byte which identifies the type of DCB and also its condition. Bit 7 (the high order bit) indicates whether the DCB is a file DCB or not, that is, a DCB for a disk file in an OPEN condition. Bit 6 indicates whether the block is a DCB or a Drive Control Table (DCT).

If the DCB is linked to another, then bit 5 will be set to 1. If the DCB is ROUTed, then bit 4 is set to 1. If the DCB is in a NIL condition ("killed"), then bit 3 will be set to a 1 to indicate it.

Bits 2 through 0 of this byte indicate the types of I/O calls from the system which are valid for this DCB. If bit 2 is set to 1, then a @CTL call is valid, meaning that the driver for this device will return from a call only when a character is available.

If bit 1 is set to a 1, then @GET calls are valid. This means the device is capable of character-oriented input. Bit 0 set to a 1 would indicate that PUT calls are valid, meaning that the device is capable of character-oriented output. Some devices are capable of only one or the other, while other devices are capable of both input and output.

DCB +01, +02

These two bytes hold the address of the device I/O routine (the driver routine) in LSB/MSB format.

DCB +03, +04

If the device is LINKed or ROUTEd to another device (i.e., bit 4 or bit 5 of DCT +00 is set), then these two bytes will contain the address of the other device's DCB in LSB/MSB format.

DCB +05

This byte is a flag byte and its use varies depending on the device in question. Bit 0 will always indicate whether or not translation (filtering) is in effect for the device or not. See below for use with particular devices.

DCB +06, +07

If translation is in effect, that is, the device has been FILTERed, then these two bytes will contain the address of the translation table in LSB/MSB format.

DCB +08

If translation (filtering) is in effect, then this byte will hold the number of entries in the translation table pointed to by the preceding two bytes. The length of the table is the value in this byte times 2.

DCB +09, +10

These two bytes hold the starting address of the device's spool buffer, if any, in LSB/MSB format.

DCB +11, +12

These two bytes encode the size of the spool buffer pointed to by the preceding two bytes. Minimum size is 2 bytes.

DCB +13, +14

These bytes form the offset from the start of the device's spool buffer to the point where an incoming data byte may be added into the buffer (next available buffer location).

DCB +15, +16

These form the offset to the location of the next data byte to be taken from the buffer and spooled out to the physical device.

KEYBOARD DCB (@KI)

Flag Byte definitions (DCB + 5):

Bit 7 - unused
6 - BREAK OFF switch: 1=Yes, 0=No
5 - BREAK Processor active: 1=Yes, 0=No
4 - HOLD Processor active: 1=Yes, 0=No
3 - BREAK character waiting: 1=Yes, 0=No
2 - HOLD character waiting: 1=Yes, 0=No
1 - Active CHAINing: 1=Yes, 0=No
0 - Translation in effect: 1=Yes, 0=No

In addition to the common data bytes defined above, other bytes used by the @KI DCB are:

DCB	+ 17,18	FCB address of DO file, if executing.
	+ 19,20	Address of BREAK processor

Byte DCB +05 is used by the @KI device to flag conditions which involve the keyboard.

If Bit 6 of this byte is set to 1, then the BREAK key has been disabled and will not be responded to by the keyboard driver.

Bit 5 set to 1 means the BREAK processor (either the operating system's, or a user's routine) is active. Bit 4 set to 1 means the system HOLD processor is active.

If bit 3 is set to 1, it means that a BREAK character has been detected (break key pressed). How this character is processed will depend on the condition of bits 6 and 5.

Bit 2 set to 1 means a HOLD character has been detected. How it is dealt with depends on the condition of bit 4.

Bit 1 indicates that chaining from a DO file is active. This means that input from the DO file will be processed by the @KI device as though they were actually typed in from the keyboard.

Bit 0 set to 1 indicates that the device is being filtered by a translation table.

If chaining is in progress, then bytes DCB +19 and +20 will contain the address of the File Control Block (FCB) of the active DO file. If the BREAK processor is active (bit 5 of the flag byte set to 1) then relative bytes 21 and 22 will hold the address of the BREAK processor routine in LSB/MSB format.

DISPLAY DCB (@DO)

The Video Display DCB uses relative bytes 09 through 18 differently from the other DCBs. The use of these bytes are defined below:

DCB	+ 09	Character mode mask
	+ 10	Line length
	+ 11,12	Cursor location
	+ 13,14	Start of scroll area
	+ 15,16	End of scroll area + 1
	+ 17	Scroll count

DCB + 09

This is a mask byte which indicates the character modes in effect at the current time.

DCB + 10

This byte encodes the video line length (normally 80 characters, unless double-size characters are being displayed).

DCB + 11, + 12

These two bytes encode the current position of the cursor. This is an offset from the start of the video screen display.

DCB + 13, +14

These two bytes define the video scroll area, i.e., the first video position which is not scroll-protected, in offset form from the start of the video display.

DCB +15, +16

The end of the scroll area + 1 is encoded in these two bytes, also offset from the start of the video display.

DCB +17

This byte contains the scroll count.

LINEPRINTER DCB (@PR)

Flag byte (DCB + 5) definitions:

Bit 7	- Driver type:	1=Serial, 0=Parallel
6	- LF after CR:	1=On, 0=Off
5	- Unused	
4	- Print lowercase:	1=Yes, 0=No
3	- Transparent mode:	1=On, 0=Off
2	- Form feed type:	1=Real, 0=Translated
1	- Tab type:	1=Real, 0=Translated
0	- Translation:	1=On, 0=Off

Bit 7 encodes the type of driver routine being used with the printer. If set to 1, it means that the serial printer driver is in use, and all output is going out through serial channel B. If set to 0, then the default parallel printer driver is in use.

Bit 6 indicates whether the printer requires a linefeed (ASCII 0AH) after each carriage return (ASCII 0DH) in order to get correct spacing. If set to one, then the printer driver will issue a linefeed following each carriage return.

Bit 4 indicates whether the printer hooked up to the system is capable of producing lowercase or not. If so, then lowercase letters are sent out as is; otherwise they are translated to upper case.

Bit 3 indicates whether transparent mode is on or off. If transparent mode is on, then all data bytes are sent out to the printer without processing or translation.

Bit 2 determines the form feed type required by the printer. If this bit is set to one, then the ASCII formfeed code is transmitted to the printer (ASCII 0CH). Otherwise, the printer driver translates a form feed into the necessary number of carriage returns and/or line feeds to arrive at the next top-of-form position.

Bit 1 indicates whether tab codes (ASCII 09H) are to be sent out to the printer as is (the printer acts on the tab code) or whether they should be translated to the correct number of spaces to arrive at the next tab field.

Bit 0 indicates whether filtering is in effect or not.

Additional bytes used by the @PR DCB are as follows:

DCB	+ 17	Printed lines/page
	+ 18	Page length in lines
	+ 19	Line counter
	+ 20	Maximum characters per line
	+ 21	Character counter (current line)

The number of printed lines per page, page length, and line length are normally set through the FORMS library command or through the PRINIT Supervisory Call (SVC 17).

Communications Channel Serial Port DCBs (@CA and @CB)

Flag byte definitions, (DCB + 5):

Bit 7 - Unused
6 - Unused
5 - Unused
4 - Unused
3 - Unused
2 - Unused
1 - Channel initialized: 1=No, 0=Yes
0 - Translation: 1=On, 0=Off

Bit 1 indicates whether or not the port has been initialized or not (either through the SETCOM library command or the RS232C Supervisory Call (SVC 55). Bit 0 indicates whether filtering is in effect or not.

Additional bytes used by the two SIO DCBs are:

DCB	+ 17	Base Port SIO (command/status port)
	+ 18	CTC port #1
	+ 19	CTC port #2
	+ 20	Word length mask
	+ 21	Stop bits
	+ 22	CTC Register 5 mask (for PRCTRL calls)
	+ 23	Baud rate/word length/parity configuration

Bit 7 reserved
6 baud rate config (high)
5 baud rate config (next)
4 baud rate config (low)
3 word config (high)
2 word config (low)
1 parity even/odd: 0=Odd, 1=Even
0 parity on/off: 0=Off, 1=On

DCB + 17

This byte identifies the command/status port for this serial channel.

DCB + 18, +19

This byte identifies Counter-Timer controller ports #1 and #2 for each channel.

DCB + 20

This byte is a word length MASK. The number of bits set to 1 determine the length of the word, not the actual value of the byte. For example, 8-bit words would be represented by 0FFH (all bits set).

DCB + 21

This byte contains the number of stop bits being used in this channel.

DCB + 22

This is a mask byte used to set certain conditions in the serial port hardware via CTC register 5. This byte is reserved for system use only.

DCB + 23

This byte encodes the baud rate, word length and parity configuration of this channel. Baud rate is encoded in bits 6, 5, and 4 as follows:

0 - 110 baud	4 - 1200 baud
1 - 150 baud	5 - 2400 baud
2 - 300 baud	6 - 4800 baud
3 - 600 baud	7 - 9600 baud

The word length - 5 is encoded in bits 3 and 2. Bit 1 determines whether even or odd parity will be used. If set to 1, then even parity is used, otherwise odd parity is used.

Bit 0 determines whether parity is enabled or disabled. If set to a 1, then parity is enabled; otherwise parity is disabled, and the setting of bit 1 is ignored.

USER DEVICE DCBs (@U1, @U2, @U3)

These devices are normally unused in the standard system and are available to the user. The type flag (DCB +00) is set to 8, indicating a NIL device. It is up to the user who establishes a specialized driver for these devices to handle the DCB type.

Each user DCB is 13 bytes in length, and defined as follows:

DCB	+ 00	Type flag (NIL)
	+ 01,02	Driver Address
	+ 03,04	Linked/routed DCB address
	+ 05	Flag byte
	+ 06,07	Translation table address
	+ 08	Translation table length
	+ 09	Available
	+ 10	Available
	+ 11	Available
	+ 12	Available

Relative bytes 00 through 08 are used in the same fashion as the other DCBs. The remaining bytes are available to the user routine.

DRIVE CONTROL TABLES

DOSPLUS II also maintains current information about the eight logical disk drives (drives 0 through 7) that it supports. Information about these drives are maintained in contiguous blocks of memory called Drive Control Tables, or DCTs. Each drive active in the system is assigned a DCT. The system interfaces with the physical disk drive equipment through these tables, just as it interfaces to other devices through the device control blocks.

Certain equipment, such as high-capacity hard drives, may be subdivided into two or more logical drives within DOSPLUS II, and each logical drive will be assigned a DCT. Under this configuration, logical drives may start at any location on the hard drive (DOSPLUS II allows logical drives to start anywhere, not just at a platter or surface boundary). Information concerning the starting point of the logical drive is also maintained in the DCT for that drive number. For example, logical drive 2 may start at track 100, Sector 32 of a five-megabyte drive. This information will be in the DCT. If a RST 10H instruction is used to access the drive, the offsets will automatically be calculated. These instructions are detailed in the section on the Resident Disk I/O system.

The DCTs are normally 23 bytes in length, and their position in memory varies depending on the type of drive being interfaced. The address of the DCT for any currently defined drive can be gotten through the LOCDCT call in the Resident Jump Table. Note that drives are always accessed through this call using drive numbers 0 through 7, rather than absolute device numbers 8 through 15. The DCTs are heavily used by the system and the user should refrain from modifying them directly. Supervisory calls are available for this purpose.

The Drive Control Table bytes are defined as follows:

DCT	+ 00	Type Flag Bit 7 - File DCB: 1=Yes, 0=No 6 - DCT/DCB: 1=DCT, 0=DCB 5 - Reserved for future use 4 - Reserved for future use 3 - NIL: 1=Yes, 0=No 2 - Reserved 1 - Reserved 0 - Reserved
	+ 01,02	Driver address
	+ 03,04	Reserved for future use.

+ 05	Flag Byte	
	Bit 7 - 5"/8" flag:	1=5", 0=8"
	6 - Software Write Protect:	1=Yes, 0=No
	5 - Hard/Floppy:	1=Hard, 0=Floppy
	4 - Motor-on Delay:	1=Yes, 0=No
	3 - Head load:	1=Yes, 0=No
	2 - Skip drive:	1=Yes, 0=No
	1 - Fixed/Removable (Hard disks only):	1=Yes, 0=No
	0 - Log disk:	1=Yes, 0=No
+ 06	Step rate	
+ 07	Head offset	
+ 08,09	Cylinder offset	
+ 10	Sector offset	
+ 11	Head location	
+ 12	Binary drive number	

The contents of the following bytes (relative bytes 13 through 22) will change depending on the formatting of the disk placed in that drive.

DCT	+ 13	Bit 7 - Diskette density: 1=Double, 0=Single
		6 - Directory protected
	+14	Surface count
	+ 15	Cylinder count
	+ 16	Sectors/track
	+ 17	Directory length
	+ 18	Directory location
	+ 19	Sectors/granule
	+ 20	Granules/cylinder
	+ 21,22	Sectors/cylinder

A detailed explanation of the DCT table follows.

DCT +00

This is a type flag byte which identifies the type of device. Bit 6, if set to 1, identifies the block as a Drive Control Table. The condition of bit 3 will indicate whether the drive is active or has been "killed" (set to NIL).

DCT +01, +02

These two bytes contain the address of the I/O driver routines for this logical drive, in LSB/MSB format.

DCT +03, +04

Reserved for use in future implementations of DOSPLUS II.

DCT +05

This is a flag byte which defines certain characteristics of the actual physical hardware, along with some software conditions.

Bit 7 identifies the size of the disk drive. If set to a 1, the drive is a 5.25" drive. If set to a 0, it is an 8" disk drive. This applies to both floppy disk drives and hard drives.

Bit 6 indicates whether this logical drive has been write-protected from software or not. This is a software condition. If set to 1, then DOSPLUS II will make no attempt to write to this particular drive.

Bit 5 identifies whether the logical drive is assigned to a floppy disk or a hard disk. If set to 1, then the physical drive is a hard disk.

Bit 4 informs the system whether the physical drive needs a delay after turning on the motor to come up to speed. Certain minifloppy drives require a one-second delay before I/O can be attempted. If this bit s set to 1, DOSPLUS II will wait one second before performing any I/O to the drive.

Bit 3 informs the system whether a head load signal is required by the drive. Some drives require this signal, while others do not (the read/write heads are constantly in contact with the disk media).

Bit 2 is used for minifloppy drives only, and indicates whether the system should read the disk in "skip" mode. This entails reading every other track, and is used to read a 48-track-per-inch formatted diskette on a 96-track-per-inch minifloppy drive. This allows the standard 35/40 - track minifloppy diskettes to be read on an 80-track disk drive.

Bit 1 is used for hard drives, and indicates whether the drive media is fixed or removable.

Bit 0 indicates whether the DCT information for the diskette in this drive has already been read or not. If set to 1, then the next disk access for this drive will cause the DCT information on the diskette's boot track to be read in. If set to 0, it indicates that the DCT information is current for this diskette.

DCT +06

This byte encodes the track-to-track stepping rate that the operating system is to use with this particular drive. Minifloppy drives can step anywhere from 40 msec. to 6 msec track-to-track, depending on manufacturer. 8" floppies normally step at 3 msec track to track.

The next four bytes identify the starting location of a logical drive on hard disk media. DOSPLUS II does not require that a logical drive start on a particular platter, cylinder or sector; it can start anywhere.

DCT + 07

This byte encodes the head offset for this logical drive. This is used when more than one logical drive is assigned to the same piece of hardware, as in the case of high-capacity hard drives which are divided into two or more logical drives. Normally, hard drives consist of several platters, and one read/write head for each platter surface. This byte tells the system which head (and therefore which platter) this particular logical drive begins at.

DCT +08,+09

These two bytes contain the cylinder offset for this logical drive. A logical drive need not start on cylinder 0. The system can tell from these two bytes the starting cylinder of the drive.

DCT +10

This is the sector offset within a cylinder where a logical drive starts.

DCT +11

This byte codes the head location for this logical drive.

DCT +12

This is the binary drive number for this logical drive. It is the same as the physical drive number and can take values from 0 to 7.

The next ten bytes contain information which are characteristic of the diskette in the drive at the moment. In the case of floppy and minifloppy disk drives, the contents of these bytes may change whenever a diskette is changed.

DCT +13

The two highest bits of this byte contain the following information concerning the diskette format:

Bit 7 indicates the density of the diskette. If set to 1, then the diskette is formatted in double density.

Bit 6 indicates whether the directory track of the diskette is "read protected," that is, was written using "read-protected" data address marks, or DAMs. This does not actually read-protect the directory, but serves to separate the directory track from all the others for quick identification.

DCT +14

This byte encodes the number of surfaces on a disk assigned to this logical drive. A double-sided floppy disk would have two surfaces, for example, while a logical drive on a hard disk may be assigned more than two, depending on how it is configured.

DCT +15

This byte encodes the number of cylinders assigned to this logical drive. Minifloppy disks can have up to 96 cylinders, where each cylinder may be composed of one or two tracks (depending on whether the disk is formatted as single sided or double sided). An eight-inch floppy will normally have 77 cylinders. The cylinder count for hard drives will vary depending on how the logical drive has been configured.

DCT +16

The number of sectors per track is encoded in this byte. For non-hard disk drives, the sector count will vary depending on the density the media has been formatted in.

DCT +17

This encodes the length of the directory, in sectors, for a particular diskette or logical hard drive.

DCT +18

This encodes the location of the directory (cylinder number).

DCT +19

This byte encodes the number of sectors per granule for a particular diskette or logical hard drive. A granule is an artificial unit which is the minimum amount of space that will be allocated to a file by the system at any one time. The size of a granule will vary according to density and media. Floppy disks formatted under DOSPLUS will normally have 5 sectors per granule. Hard disks may have up to 32 sectors per granule.

DCT +20

This byte tells the system how many granules per cylinder are on the media. A standard single-sided 8-inch floppy disk formatted under DOSPLUS II, using 5-sector granules, will normally have 6 granules per cylinder.

DCT +21,22

These two bytes encode the number of sectors per cylinder formatted on the media. The standard 8-inch floppy disk formatted under DOSPLUS II will have 30 sectors; TRSDOS disks will have 26 sectors. Sector count for hard drives will vary.

DOSPLUS II Supervisor Call System

The system routines in DOSPLUS II may be accessed by user-written programs through the Supervisory Calls (SVCs). The SVCs permit user programs to take advantage of the operating system's routines to perform various functions such as device and file I/O and file access.

The DOSPLUS II system can have a total of 128 Supervisory Calls, although not all are defined in the current release. Each one is assigned a code number, from 0 to 127. DOSPLUS II Supervisory Calls are compatible with the TRSDOS 2.0 and TRSDOS 4.0/4.1 SVCs. Also, slots 102 through 127 are left open for the user to define his own SVCs.

All Supervisory Calls are executed by loading the Z80's A-register with the proper SVC number and executing a RST 8 instruction. Depending on the call, the other registers may need to be loaded with particular values before the RST 8 is executed. All SVCs return to the instruction following the RST 8 in the calling program. On return, the contents of all primary registers not defined in the explanation of the SVCs below are assumed unchanged; however, the alternate register set is used by the system and their contents must be assumed destroyed.

It is important to realize that while DOSPLUS II places very few restrictions on the user insofar as memory utilization goes, users should avoid those areas which are likely to be loaded by the system, especially when executing DOS-level commands from within a program. DOSPLUS II uses the following areas of memory:

0000H - 1BFFH	Resident system
1C00H - 21FFH	Low Overlay region
2200H - 27FFH	High Overlay region

This section details each Supervisory Call along with their entry and exit conditions. It is assumed that the reader is an experienced programmer familiar with the Z80 instruction set.

Notational conventions used in this section:

1) Hexadecimal numbers are represented with a trailing "H", e.g., 2BH, 31H.

2) The notation LSB/MSB is used to refer to the Z80 convention of coding address references with the least significant byte (LSB) coming first, followed by the most significant byte (MSB).

3) Standard ZILOG mnemonics and notation are used for Z80 machine instructions. The alternate register set of the Z80 is represented as the PRIME of the standard set, with a trailing apostrophe, e.g., AF'.

4) When a register pair is surrounded by parentheses, e.g., (HL), it indicates that this register pair contains an address reference to a location in memory (data outside the Z80 chip's registers).

SVC CALLING PROCEDURE

1) Load the Z80 A-register with the number of the Supervisory Call you wish to execute. Load any other registers required by the call.

2) Execute a RST 8.

3) The Supervisory Call routine will return to the instruction following the RST 8 with the Z flag set if the routine executed properly. The contents of register A will be 0 in this case. If the Z flag is not set (NZ condition), an error has occurred and the A register will contain the proper error code. Your routine should test the Z flag and carry flag immediately on return from the SVC and act accordingly.

EXAMPLES

A) Output a line to the lineprinter

```
            LD          HL,LINE     ;point to the line buffer
            LD          B,64        ;number of chars to print
            LD          C,13        ;Terminating character
            LD          A,19        ;SVC code number
            RST         8           ;Execute the SVC
            JP          NZ,ERROR    ;Execute error routine if NZ.
;------ continue here if successful ------
```

B) Convert a binary value to ASCII-coded hexadecimal

```
            LD          B,0         ;Set for binary-hex conversion
            LD          DE,NUMBER   ;NUMBER = 16 bit binary number
            LD          HL,BUFR     ;BUFR = 4 byte buffer for result
            LD          A,24        ;SVC code
            RST         8           ;Go perform the conversion
```

C) Get a character from the keyboard and display it on the video screen at the current cursor position

```
GETKEY      LD          A,4         ;Get-character SVC code
            RST         8           ;Execute SVC
            JR          Z,HAVKEY    ;Got a keypress, display it!
            CP          2           ;check contents of register A
            JR          Z,GETKEY    ;not an error, loop back
            JR          ERROR       ;Register A has an error code
;B contains character code if a key was pressed
HAVKEY      LD          A,8         ;Display-character SVC
            RST         8           ;display character on video
            JR          NZ,ERROR    ;Process if error condition
; -- otherwise continue with program here
;            -------
;            -------
ERROR       OR          80H         ;Set high bit of error code
            LD          B,A         ;put error code in B
            LD          A,39        ;load SVC number
            RST         8           ;display error message
;with high bit of error code set, DOSPLUS II will take an
;immediate exit to DOS Ready after displaying the error
;message.
```

==========

INITIO (SVC 0)
Initialize DOSPLUS II I/O System

This routine initializes the DOSPLUS II system I/O drivers. It calls initialization routines for the keyboard, video display device and printer. The video is cleared and set to 80 characters per line and normal (white-on-black) mode, the printer is set to 66 lines per page with 60 print lines and 132 characters per line. The keyboard type-ahead buffer and the printer spool buffers are emptied.

No parameters are required. This routine is called by DOSPLUS II and should not have to be called by the user unless a severe error condition has occurred.

ENTRY CONDITIONS
A = 0

EXIT CONDITIONS
Z = No error. Register A contains 0.
NZ= Error
A = Error code if NZ.

==========

KBINIT (SVC 1)
Initialize Keyboard Device

This routine initializes the DOSPLUS II default keyboard driver Calling this routine does not affect any translation which may be in effect. Also, any device linking involving the keyboard is not changed, and the location and size of the type-ahead buffer is also not changed. However, the type-ahead buffer will be cleared by this SVC, along with hardware keys.

ENTRY CONDITIONS
A = 1

==========

SETUSR (SVC 2)
Set Up User-defined SVC

This routine sets or removes a user vector in the system's SVC table. Under DOSPLUS II, any valid SVC code number (0 - 127) may be used to set up a user SVC. Normally, however, users should set up their routines using SVCs 102 through 127 which are reserved for this purpose. Once set up, the user program can then be called via the RST 8 instruction. User routines may reside anywhere in free memory. Previously defined SVC vectors may be redefined without first removing it from the table.

ENTRY CONDITIONS

A = 2
B = SVC code to be used, 0 to 127.
C = Set/Reset select. If C=0, remove the vector. If C <> 0, insert the vector.
HL = Transfer address of the User SVC (when C <> 0).

EXIT CONDITIONS

Z = No error. A register contains 0.
HL = Removed vector address (when C = 0).
NZ = Error.
A = Error code if NZ (parameter error, indicating a value in B greater than 127 when SVC was called).

==========

SETBRK (SVC 3)
Set Up a Break Key Processor Routine

This routine will allow you to intercept the BREAK key and transfer control to your own routine when BREAK is pressed. All the PRIMARY Z80 registers are preserved when control is transferred to your routine (your routine should not use the alternate register set if possible, or else save them and restore them before exiting).

If the break processor is ON, and if the debugger is not active, then the break character will not be returned by the keyboard, and the user break routine will be executed when the key is pressed . If the debugger is active, then it will have priority over the SETBRK user routine. If the break processor is OFF (default) then control will not transfer to the user vector, but the break character (03H) will be returned by the keyboard.

This routine will also permit you to disable the BREAK key by removing the address of the processing program from the interrupt vector table.

BREAK processor vectors may be changed without first removing the previously defined vector.

ENTRY CONDITIONS

A = 3
HL => Address of BREAK processor routine. When the BREAK key is pressed, control is transferred to this address immediately. If HL=0, then the vector to the previous processing program is removed, disabling the BREAK key vector.

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.
HL => Address of old BREAK processor routine if HL was 0 on entry.

Note that on exit, the break processor will return to the instruction which was interrupted. If the user's break processing routine saves the registers on entry and restores them prior to executing the RET instruction, the interrupted program can take up where it left off.

==========

KBCHAR (SVC 4)
Get One Character From the Keyboard

This routine will get one character from the type ahead buffer if any are available. It will return immediately whether or not a key was pressed. If a user BREAK processor was set, then the BREAK character will not be returned (see above). If the system BREAK processor is not active, then the routine will return to the calling program with the BREAK (03H) character. Similarly, if the system HOLD processor is set, then the HOLD key character will not be returned, and the system will interpret the HOLD. Otherwise, the HOLD (00H) character will be returned.

ENTRY CONDITIONS

A = 4

EXIT CONDITIONS

NZ = No key was pressed, or error.
A = 2 if no character available, otherwise error code.
B = Character found, if any. If no key was pressed, B is returned unchanged.

==========

KBLINE (SVC 5)
Get a Line of Input From the Keyboard

This routine will input a line from the keyboard into a user-defined buffer and echo each character to the video as it is typed. The input line is terminated when a carriage return is typed, or when BREAK is typed. If a carriage return (ASCII 13, 0DH) is used to terminate the line, it is stored in the buffer along with the rest of the characters. If the buffer fills before a carriage return is typed, then no other keys will be accepted EXCEPT a carriage return or break, which MUST be typed. The carriage return will be placed in the buffer along with the rest of the input line, if the buffer is not full.

These keys have specific actions with this SVC:

ENTER (13, or 0DH)	terminates the input line
ESC (27, or 1BH)	restarts input line. Previous input is erased.
BACKSPACE (08H)	Backs cursor one position, erases last character input.
BREAK (03H)	terminates the input line.

ENTRY CONDITIONS

A = 5

B = Maximum number of characters to input from keyboard. If B=1, then return on first key pressed. If B <> 1 then the line must be properly terminated.

HL => Starting address of user's input buffer.

EXIT CONDITIONS

Z = No error.

C = line termination flag. If C=0 then the buffer was filled before carriage return was typed. If C contains 0DH (carriage return) then the line was terminated before the buffer was filled, and the carriage return character is in the input buffer.

Carry = If the carry flag is set, then BREAK was used to terminate input. This is the only way to determine if BREAK was used to terminate the input line.

==========

DELAY (SVC 6)
Delay a User-Defined Interval

This routine provides a user-defined delay, returning control to the calling program when the delay interval has elapsed.

ENTRY CONDITIONS

A = 6
BC = Delay multiplier. Maximum delay is 426 milliseconds, if BC=0. If BC > 0, then the delay time is determined by the following formula:
Delay = 6.5 *(BC-1)+22 microseconds.

EXIT CONDITIONS

A = 0
Z = set

==========

VDINIT (SVC 7)
Initialize Video Display

This routine initializes the Video display and should be called once before starting any output to the video. The screen will be cleared and the cursor moved to the upper left position. This call will not affect any translation which is already active.

ENTRY CONDITIONS

A = 7
B = Character size select. If B=0, then character size is set to 40 characters/line. If B <> 0 then character size is set to 80 characters/line.
C = Mode select. If C<>0, then set normal (white-on-black) mode. If C=0, then set reverse video mode (black-on-white).

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

VDCHAR (SVC 8)
Display Character on Video Screen

This routine outputs a single character to the display in Scroll Mode. ASCII codes 0 through 255 are all valid for display.

The following control codes affect the video display:

Key	Hex Code	Function
F1	01H	Turn on cursor with SLOW BLINK
F2	02H	Turn cursor off
Ctrl-C	03H	Turn on cursor with FAST BLINK
Ctrl-D	04H	Turn on STEADY cursor
BACKSPACE	08H	Move cursor left one column position and blank out character there.
TAB	09H	Move cursor to the next tab field boundary (tab fields are 8 columns wide).
Ctrl-J	0AH	Move cursor down to next row, same column position.
Ctrl-K	0BH	Position cursor at the start of previous line if not scroll protected area.
ENTER	0DH	Move cursor to start of next line
Ctrl-T	14H	Home cursor to the first non-protected position on screen.
Ctrl-W	17H	Erase to end of line without moving cursor
Ctrl-X	18H	Erase to end of screen without moving cursor
Ctrl-Y	19H	Set Normal Display Mode (white-on-black).
Ctrl-Z	1AH	Set Reverse Display Mode (black-on-white).
ESC	1BH	Erase screen and home cursor to row 0, column 0.
Left Arrow	1CH	Move cursor left one column position.
Right Arrow	1DH	Move cursor right one column position.
Up Arrow	1EH	Set 80 characters/line mode. Display is cleared.
Down-Arrow	1FH	Set 40 characters/line mode. Display is cleared.

ENTRY CONDITIONS
A = 8
B = ASCII code for character to output to video. 0 to 255 are valid.

EXIT CONDITIONS
NZ = Error
A = Error code if NZ.

VDLINE (SVC 9)
Display Line on Video Screen

This routine takes characters from a user-defined buffer in memory and displays them on the video in Scroll Mode starting at the current cursor position. ASCII character codes from 0 to 255 are valid.

ENTRY CONDITIONS

A = 9
B = Number of characters to be displayed.
C = End of line character. This character is sent after the last character in the buffer. If C=0, then no terminating character is sent.
HL => Starting address of the buffer containing the display line.

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ

On exit, the cursor is positioned immediately after the last displayed character.

==========

VDGRAF (SVC 10)
Display Characters in Graphics Mode

This routine displays a buffer of characters in Graphics Mode, starting at a specified row and column on the video display. All character codes from 0 through 255 are valid.

Active control codes are as follows:

HEX CODE	FUNCTION
F9H	Sets white-on-black mode (Normal). Cursor is not moved. Display mode reverts to previous on exit.
FAH	Sets black-on-white mode (Reverse). Cursor is not moved. Display mode reverts to previous on exit.
FBH	Homes cursor to Row 0, Column 0
FCH	Moves cursor back one space. When column = 0, cursor moves to column 79 of preceding row.
FDH	Moves cursor forward one space. When column = 79, cursor moves to column 0 of next row.
FEH	Moves cursor up one row, same column. If Row = 0 then cursor wraps around to row 23.
FFH	Moves cursor down one row, same column. If Row = 23, then cursor wraps around to Row 0.

ENTRY CONDITIONS

A = 10
B = Row on screen to start displaying buffer, 0 to 23.
C = Column on screen to start displaying, 0 to 79.
D = Length of buffer, 1 to 255. If D=0, then cursor is positioned to the coordinates determined by the BC register pair.
HL => Starting address of text buffer.

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

On exit, cursor is left positioned at the graphics position immediately following the last displayed character.

==========

VDREAD (SVC 11)

Read Video Memory into User-Defined Buffer

This routine will read characters from the Video display into a user-defined buffer in Graphics Mode. When the read goes beyond column 79, it will wrap around to column 0 of the next row. When the read goes beyond column 79 of row 23, then it wraps around to column 0, row 0.

This routine can also be used to determine the present location of the cursor.

ENTRY CONDITIONS

A = 11
B = Row on screen where read is to start, 0 to 23. If B > 23 then B mod 24 is used to determine starting row.
C = Column on screen where read is to start, 0 to 79. If C > 79 then C mod 80 is used to determine starting column.
D = length of user's input buffer, 0 to 255. If D=0, then current cursor position is returned in BC register pair.
HL => Starting address of input buffer.

EXIT CONDITIONS

BC = Current cursor position (B=row, C=column).
Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

VIDKEY (SVC 12)
Get Input Line from Video With Prompt Message

This routine will output a prompting message to the video display at the current cursor position, then accept a line of input from the keyboard. The prompt message is kept in a user-defined buffer.

The Type-Ahead buffer is NOT cleared by this routine.

ENTRY CONDITIONS

A = 12
B = Number of characters in prompt message (0 - 255)
C = maximum length of keyboard input field (0 - 255)
DE => Start of input buffer where keyboard characters are to be stored.
HL => Start of buffer containing prompt message.

EXIT CONDITIONS

NZ = Error (invalid display character)
A = Error code
Z = No error.

If Z is set, indicating no error, then

B = Number of characters input from keyboard.
C = Length of displayed prompt string.
HL => Start of display prompt string
DE => keyboard input buffer

==========

SVC 13 and SVC 14 are UNDEFINED

==========

DISKID (SVC 15)
Read a Diskette ID

This routine reads the diskette ID from any or all drives 0 through 7. This is the ID placed on the diskette at FORMAT or BACKUP time. This routine can be used to determine if the operator has inserted the proper diskette.

ENTRY CONDITIONS

A = 15
B = Drive select code. B may take the values 0 through 7 and 255. If B=255 (FFH) then the IDs of ALL mounted drives are read.
HL => Address of the buffer for the Diskette IDs. If B = 0 through 7, then the buffer must be 8 bytes long. If B=255, the buffer must be 64 bytes long. The eight-byte IDs are placed into the buffer starting from relative drive 0. Blanks are inserted for drives not ready (no diskette, open door, etc.)

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = An error occurred.
A = Error code if NZ

==========

SVC 16 is UNDEFINED

==========

PRINIT (SVC 17)
Initialize Lineprinter Device

This routine initializes the DOSPLUS II lineprinter driver. It does not check the printer status (i.e, on/off, offline/online, etc.) or output any codes to the lineprinter itself.

ENTRY CONDITIONS

A = 17
B = Page length (maximum number of lines per page) 66 lines is standard.
C = Printed lines per page. May not be greater than B. 60 lines is standard.
D = Line length (maximum characters/line). 132 characters per line is standard. If D=0 then tabs (09H) will not be translated into spaces.

If B=C and both are NOT 0, then automatic top of form will not be done; however, a form feed (0CH) will be translated into the correct number of carriage returns or line feeds to get the paper to the next top of form.

If B and C are both 0, then form feed codes (0CH) and vertical tabs (0BH) are passed directly to the printer without translation. No automatic form feed will be done.

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error.
A = Error code if NZ.

The character counter and line counter registers of the printer DCB are set to 0. The user must set the top of form manually on his printer.

==========

PRCHAR (SVC 18)
Print Character

This routine will output a single character to the lineprinter device (Device 02). Depending on the printer used, the character may or may not print at the time it is output; many printers incorporate a print line buffer and will not print until the buffer fills or a carriage return is received. Also, depending on the size of the current printer spool buffer and the number of pending characters contained in it, there may be a lag between the time this SVC is called and the time the character actually appears on the printer.

ENTRY CONDITIONS
A = 18
B = ASCII code of character to be output.

EXIT CONDITIONS
Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

PRLINE (SVC 19)
Output a Line of Text to Lineprinter

This routine will send a line of text from a user-defined buffer to the lineprinter device (Device 02). Control codes may be included in the output line.

ENTRY CONDITIONS
A = 19
B = Buffer length, or number of characters to output
C = Terminator character, to be sent after last character in the buffer.
If C=0 then no terminator is sent.
HL => Starting address of buffer containing text line to output.

EXIT CONDITIONS
Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

RANDOM (SVC 20)
Random Number Generator

This routine will return a one-byte random number. The routine is passed a limiting value. The random number returned will be in the range 0 to limit value-1. For example, if the limiting value is 100, then the number returned will be in the range 0 to 99.

ENTRY CONDITIONS

A = 20
B = Limit value, in the range 2 to 255 (2H - FFH).

EXIT CONDITIONS

C = Random number in the range 0 to (B-1) except when B=0 or B=1, in which case C=0 on return.
Z = No error. Register A contains 0.

==========

BINDEC (SVC 21)
Binary to Decimal/Decimal to Binary Conversion

This routine will take a 16-bit binary number and convert it to ASCII-coded decimal in the range 0,65535 and vice-versa. The ASCII codes are placed into a 5-byte user-defined buffer. When converting from ASCII-coded decimal to binary, the ASCII digit codes are also taken from a 5-byte user-defined buffer.

ENTRY CONDITIONS

A = 21
B = Conversion select. If B=0, then convert binary to ASCII-coded decimal. If B <> 0, then convert ASCII-coded decimal to binary.

If B=0 (binary to decimal conversion):

DE = 16-bit binary number to be converted
HL => Starting address of 5-byte buffer to receive ASCII-coded number.

If B <> 0 (decimal to binary conversion):

HL => Starting address of 5-byte buffer containing the ASCII-coded decimal number to be converted.

EXIT CONDITIONS

Z = No error (binary to ASCII decimal will always return Z). Register A will contain 0.
NZ = Error (invalid character encountered).
A = Error code if NZ
HL => Starting address of 5-byte buffer
DE = Binary value if B <> 0

==========

STCMP (SVC 22)
Compare two strings

This routine will compare two alphanumeric strings and determine their sorted order. All or part of the two strings may be compared.

ENTRY CONDITIONS

A = 22
BC = Number of characters to compare
DE => Starting address of buffer containing first string
HL => Starting address of buffer containing second string

EXIT CONDITIONS
Flag register bits will indicate status as follows:

Z = strings are identical
NZ = strings are not identical
Carry = If set, first string (pointed to by DE) precedes second string (pointed to by HL) in sorted order.

Register contents on Exit:

Register pairs BC, DE and HL are preserved on exit.
BC' = Number of characters remaining after mismatch, including the first non-matching character
DE' => Address of first non-matching character in first string
HL' => Address of first non-matching character in second string

Note: BC', DE' and HL' are the ALTERNATE register pairs.

==========

MPYDIV (SVC 23)
Multiply/Divide Function

This routine will do multiplication or division of one 2-byte value with one 1-byte value.

ENTRY CONDITIONS

A = 23
B = Multiply/Divide Select. If B=0 then multiply; if B <> 0 then divide.

For Multiplication (B=0):

HL = Multiplicand
C = Multiplier

For Division (B <> 0)

HL = Dividend
C = Divisor

EXIT CONDITIONS

HL = Result of operation (Product if multiply, Quotient if divide)
A = Overflow byte if multiplication was performed
C = Remainder if division was performed

Status (Flag) bits affected by multiplication are:

Carry Flag = Set if overflow
Z flag = Set if result is zero

Status (Flag) bits affected by division are:

Carry Flag = Set if division by zero attempted (not performed).
Z flag = Set if quotient is zero

==========

BINHEX (SVC 24)
Binary/Hexadecimal Conversion

This routine will allow you to convert a 16-bit binary number to four ASCII-coded hexadecimal digits or vice versa.

ENTRY CONDITIONS

A = 24
B = Select. If B=0 then convert binary to ASCII-coded hexadecimal, otherwise convert hexadecimal to binary.

If B=0, then

DE = Contains 16-bit binary number to be converted
HL => Starting address of user-defined 4-byte buffer to hold ASCII-coded hexadecimal result.

If B<>0 then

HL => Starting address of user-defined 4-byte buffer containing ASCII-coded hexadecimal number to convert, left-filled with leading zeroes if necessary.

EXIT CONDITIONS

Z = No error (binary to ASCII-coded hexadecimal always returns with Z flag set). Register A contains 0.
NZ = Error (invalid characters encountered).
A = Error code if NZ.
DE = Binary result

==========

TIMER (SVC 25)
Timed Program Interrupt

This routine will permit a running program to be interrupted when a user-defined time interval elapses. The TIMER routine runs concurrently with the main program, unlike the DELAY SVC.

The user defines how many seconds the timer is to count down. The main program will epecute normally until the interval counts down to zero, or the timer is reset. When the time reaches zero, the timer is turned off and control is transferred to a processing routine at the next RTC interrupt.

ENTRY CONDITIONS

A = 25
BC = Number of seconds to count down
HL => Starting address of processing routine to which control is transferred when time runs out.

If the HL and BC register pairs are both 0, then the timer is turned off. If HL=0 and BC <> 0, then the counter is reset to the value in BC and timing continues.

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error.
A = Error code if NZ.

==========

CURSOR (SVC 26)
Set Cursor Controls

This routine will switch the cursor on or off. The system will keep track of the cursor position whether or not it is turned on or off.

ENTRY CONDITIONS

A = 26
B = Cursor function select.
If B=0 then turn cursor off
If B<>0 then turn cursor on

==========

SCROLL (SVC 27)
Scroll Protect the Video Display

This routine will permit you to protect a portion of the video display from scrolling. From 0 to 22 lines from the top of the display can be protected. Scrolling will take place only with those line below the protected area.

ENTRY CONDITIONS

A = 27
B = Number of lines to be protected (from 0 to 22).

EXIT CONDITIONS

Z = No error.
NZ = Error.
A = Error code if NZ.

==========

LOOKUP (SVC 28)
Table Lookup Routine

This routine will look up and locate an entry in a user-defined table in memory. The table consists of 3-byte entries with the following format:

Byte 1: Search Key
Byte 2-3: Data (for example, a vector address in LSB/MSB order)

The table is terminated by a single FFH byte. This means that search keys can only take the values 00H to FEH.

ENTRY CONDITIONS
A = 28
B = Search argument
HL => Starting address of table

EXIT CONDITIONS
NZ = Argument not found
A = Contains error code if NZ (not found).

If the Z flag is set, indicating that the argument was found, then:
HL => matching data set.

If the second and third bytes of the table entry is a jump vector address, then a JP (HL) instruction will transfer control to that address. For example,

```
            LD          HL,TABLE    ;point to table
            LD          B,07H       ;07 is "key"
            LD          A,28        ;load SVC code
            RST         8           ;execute the search
            JR          NZ,NOFIND   ;couldn't find it
;Table search successful, HL now contains the 2nd and 3rd
;bytes of the table entry.
            JP          (HL)        ;transfer control to
                                    ;vector address in table
;process the error condition here.
NOFIND      OR          80H         ;set high bit
            LD          B,A         ;code to B register
            LD          A,39        ;disp. error SVC
            RST         8           ;display error message and abandon ship
```

==========

HLDKEY (SVC 29)
Process the HOLD Key

This routine will enable or disable the HOLD key processor. It is also used for generating an indefinite pause terminated by a second press of the HOLD key.

ENTRY CONDITIONS

A = 29

B = Function select. If B=0, then the system HOLD key processor is shut off and pressing the HOLD key will return keyboard code 00H. If B=1 then the system HOLD processor is enabled. The HOLD key will be interpreted by DOSPLUS II. If B>1 then the system will check for the HOLD key. If pressed, the routine will pause until it is pressed again.

EXIT CONDITIONS

NZ = HOLD processor is disabled.

Z = HOLD processor is active.

To use this function to detect and pause on the HOLD key, first call the SVC with B=1. Then call it again with B>1 whenever you want to check for the HOLD key. If HOLD is pressed, the routine will pause until HOLD is pressed a second time.

==========

KBPUT (SVC 30)

Load character into type-ahead buffer

This routine will permit you to load a character into the keyboard type-ahead buffer for processing by the keyboard input routine of the system. The character is loaded following any other characters already in the buffer at the time. If the buffer is full, then an error condition results.

ENTRY CONDITIONS

B = ASCII code of character to be loaded into the type-ahead buffer.

EXIT CONDITIONS

Z = No error.
NZ = An error occurred.
A = Error code if NZ set.

==========

JPINIT (SVC 31)

Return to DOSPLUS II with Device Initialization

This is the recommended exit back to DOS Ready from programs which do their own keyboard, display and disk I/O. Control is returned to DOSPLUS II after all the DOSPLUS II devices are reinitialized. If this exit is not taken, it is possible that on exit from the user program, the DOSPLUS II devices will not be active, forcing you to reboot the system.

ENTRY CONDITIONS

A = 31

==========

SVC 32 is UNDEFINED

==========

LOCATE (SVC 33)
Locate Record

This routine will return the number of the last file record accessed.

ENTRY CONDITIONS

A = 33
DE => Address of File Control Block for current file
HL = Reserved for system use

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ
BC = Current record number if no error.
If the file was in Extended Access mode, then BC points to a three-byte location in memory where the current record number is stored.

==========

READNX (SVC 34)
Read Next Record

This routine will read the next record after the current one (the current record is defined as the last record accessed). If the file has just been opened, then READNX will read the first file record.

ENTRY CONDITIONS

A = 34
DE => File Control Block for currently open file
HL = Reserved for system use

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

DIRRD (SVC 35)
Direct Record Read

This routine will read the specified file record directly.

ENTRY CONDITIONS

A = 35
BC = Record number to be read. If BC=0 then position to beginning of file. If BC=FFFFH, then position to the end of file. If the access mode of the file is "E" (see OPEN, below) then the BC register pair points to a three-byte Extended file record pointer in memory.
DE => File control block of currently open file
HL = Reserved for system use

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

==========

JP2DOS (SVC 36)
Exit Current Program and Return to DOSPLUS II

This routine will force an immediate exit to DOS Ready. All chaining operations in progress, and the processing of multiple-command lines, will continue. Various control routines will be reset. It is the user's responsibility to close any open files before calling this SVC to exit back to DOS Ready. DOSPLUS II will not check the status of any open files.

NOTE: When the Display Error Message SVC (number 39) is called with an error code whose high bit has been set, it will branch to this routine after displaying the error message. This facility will make separate explicit calls to Display Error Message and JP2DOS unnecessary if the user wishes to return to DOS Ready on an error.

ENTRY CONDITIONS
A = 36

==========

DOSCMD (SVC 37)
Execute a DOSPLUS II Command

This routine passes a command string back to DOSPLUS II for execution. The command string may contain multiple commands but only the first will be executed. The line must be terminated with a carriage return (0DH), semicolon or ETX character (03H). Upon completion of command string processing, control is returned to DOSPLUS II (DOS Ready). The open/closed status of any existing files are not changed by this SVC; the user should make sure no files are left open.

This supervisory call differs from its TRSDOS counterpart in that the B register is not used. TRSDOS loads the length of the command string into the B register and does not make use of any terminator character; DOSPLUS II ignores the contents of B and REQUIRES a terminator character.

ENTRY CONDITIONS

A = 37
HL => Address of command string to be passed to DOSPLUS II

RETCMD (SVC 38)

Execute a DOSPLUS II Command And Return

This routine will pass a command string to DOSPLUS II. Upon completion, control is returned to the user program. File status will not be changed by this call. The command string may contain multiple commands, but only the first will be executed. It must be terminated either by a carriage return (0DH), semicolon, or ETX character (03H). Be careful that the executing DOS command does not overlay your user program (see the beginning of this section for memory areas used by DOSPLUS II). Also, in the event that the DOS command executing affects the state of any open files, you should be aware that any information contained in the affected File Control Blocks may no longer be valid.

This call does not use the B register to code the length of the command string, unlike TRSDOS.

ENTRY CONDITIONS

A = 38
HL => Address of command string to be passed to DOSPLUS II

EXIT CONDITIONS

Z = No error. Register A contains 0.
NZ = Error
A = Error code if NZ.

ERROR (SVC 39)
Display Error Message

This routine will display a full error message on the video screen corresponding to the error code in the B register.

If the high bit (bit 7) of the error code is set, then this routine will branch to the JP2DOS SVC (number 36), thereby forcing an exit back to DOS Ready after displaying the error message. Bit 7 can be set by the user by executing an "OR 80H" instruction with the code in the A register (where it is normally returned by the SVC system) before loading it into the B register and calling this routine.

ENTRY CONDITIONS

A = 39
B = Error code

On exit, if an error occurs while attempting to display an error message, then all linking and routing to the video will be terminated. This is done so that displaying the error message will not cause an endless loop.

==========

OPEN (SVC 40)
Open a File or Device for Access

This routine will open a device or a disk file for access. The user may control how a file is opened; that is, he can specify whether only a new file is to be opened (created), only an old file is to be opened (no new ones reated), or whether a file should be opened whether or not it exists previously.

ENTRY CONDITIONS

A = 40
DE => Starting address of 40-byte block of memory containing the file name or device name.
HL => Parameter list (Not needed for devices)

EXIT CONDITIONS

Z = No error.
NZ = Error
A = Error code if NZ

When opening a disk file for access, the user must allocate space for the FCB, the parameter list, file buffer, and record area (if LRL <> 256, see below), before executing the call to OPEN.

When OPEN is used to open a device, then the user must allocate space for the Device Control Block only. Although the number of bytes used for Device Control Blocks (DCBs) vary with the device, it is recommended that a standard length of 40 bytes be used whenever a call to the OPEN SVC is to be made.

File Control Block

A File Control Block (FCB) is a 40-byte area in memory which is used to hold file access information while a disk file is open. The name of the file to be opened is placed left-justified in this this area of memory prior to calling OPEN. The filespec must be terminated with either a carriage return (0DH), a semicolon or an ETX code (03H):

FILESPEC/EXT.PASSWORD:DR(DISKETTE)$

Where $ represents either a carriage return (ASCII 0DH), a semicolon or ETX (ASCII 03H) character.

The structure of the FCB while open is as follows:

FCB Address+0	DCB type	
1	Flag byte 1	
	BIT	7 = Blocked records
		6 = Random Access
		5 = Buffer = NRN
		4 = Buffer updated
		3 = File updated
		2 - 0 = Access code
2	Flag byte 2	
	BIT	7 = Fixed/variable records
		6 = Non-shrinkable file
		5 = Read/Write access
		4 = Dynamic EOF update
		3 = Extended open
		2 - 0 = Not used
3 - 4	Blocking buffer address	
5 - 6	User buffer Address	
7	Next Record Number (NRN) byte	
8	NRN Most Significant Byte	
9	NRN Next Significant Byte	
10	NRN Least Significant Byte	
11	Ending Record Number (ERN) byte	
12	ERN Most Significant Byte	
13	ERN Next Significant Byte	
14	ERN Least Significant Byte	
15	Logical Record Length (LRL)	
16	Drive Number (0 - 7)	
17	LFN/DEC	
18	Transfer Address	
20-39	Segment Descriptor List	

Do not modify the contents of an OPEN FCB, or the results may be unpredictable.

Parameter List

The parameter list is an 11-byte area in memory which contains additional information about the file. The contents of the parameter list are as follows:

Parameter List +	0 - 1	BUFADR (buffer address)
	2 - 3	RECADR (record address)
	4 - 5	EODAD (End-of-Data address)
	6	Access type
	7	Record Length
	8	Access mode
	9	Creation code
	10	End of list marker

BUFADR, RECADR and EODAD are maintained in standard LSB/MSB format.

BUFADR - Address of the user-defined data buffer. This buffer will be used by DOSPLUS II to process file accesses. The size required is 256 bytes.

RECADR - Record address. This contains the address of the record buffer area, where disk records are stored immediately after reading and before being written out to the disk in cases where the LRL is not 256. In DOSPLUS II, the RECADR area is the same length as the logical record length for Fixed-Length Files. For variable-length files with LRLs other than 256 it should be the same length as the longest record in the file, to a maximum of 256 bytes.

When a file is opened with LRL=0, that is, 256 bytes, then this address is not used, and records will be read into, and written from, the 256-byte area pointed to by BUFADR.

EODAD - End of Data Address. This is a two-byte field containing a jump vector to a routine in memory which will be executed when an "End of File" error is encountered unexpectedly during a disk read. The routine may be a user-defined error handler. If these two bytes are 0, then the OPEN SVC will return to the calling program with the error code in the A register.

ACCESS TYPE BYTE - This byte defines the type of access permitted for the file, and contains the ASCII codes of the following letters:

R (52H)	Read Only
W (57H)	Read/Write
P (50H)	Write Program file

Under DOSPLUS II, the P access type is acceptable, but not implemented. That is, it will not cause any errors, but neither will it cause the system to do anything differently. Specifying P access will give Read/Write access.

RL - Record Length. This byte specifies the record length to be used for the file, and may take values from 0 to 255 (00H to FFH). If RL = 0, then a record length of 256 bytes is assumed.

ACCESS MODE - This is a one byte field which defines the access mode for the file. It contains the ASCII code for one of the following letters:

V (56H)	Variable Record Length Access
F (46H)	Fixed Record Length Access
E (45H)	Extended File Access

A file that was created with the V access mode may later be reopened and accessed as a fixed record length file, that is, given the F access mode specifier.

The E access mode indicates that an Extended file pointer is to be used in accessing records from the file. The extended file pointer is a three byte rather than a two-byte pointer, which allows for files of up to 16 million records. This will affect the usage of the BC register pair in the DIRRD (Direct Read) and DIRWR (Direct Write) SVCs. When using these SVCs, the BC register pair normally contains the actual record number to be read or written. However, if a file is opened in Extended Access Mode, then the BC register pair will point to a location in memory where the actual 3-byte Extended Record Pointer is maintained. That is,

BC => ERP:	Most Significant Byte
	Next Significant Byte
	Least Significant Byte

Creation Code - This byte will determine whether a new file is to be created or a previous file opened. It may take the values 0, 1 2 or 3.

0	Open only a previously existing file; do not create a new file in the disk directory. Record length and end of file pointers are not reset.
1	Create a new file only; if the file previously exists, do not open it.
2	Open an existing file; if not found, create it on first available disk device and reset record length and end of file pointers.
3	Open an existing file and reset record length and end-of-file pointers. Do not create a new file. This code is unique to DOSPLUS II and is not available in TRSDOS.

==========

KILL (SVC 41)
Delete a File from Disk Directory

This routine deletes the specified file from the directory. The file may be in an Open or Closed condition when this call is issued.

ENTRY CONDITIONS
A = 41
DE => Pointer to the FCB.

EXIT CONDITIONS
Z = No error.
NZ = Error
A = Error code if NZ

==========

CLOSE (SVC 42)
Close an Open File or Device

This routine will terminate access operations to a file or device and close it. In the case of a file, any records remaining in the BUFADR area will be written to disk, and the directory entry for the file will be updated. On exit, the filespec will be placed back in the FCB. However, the password (if any) will not be present.

ENTRY CONDITIONS

A = 42
DE => Address of the FCB of file to be closed.

EXIT CONDITIONS

Z = No error.
NZ = Error
A = Error code if NZ.

==========

WRITNX (SVC 43)
Write Next Record

This routine will write the next disk record following the last record accessed, in sequential order. If the file is newly opened, then WRITNX will write the first record.

The record to be written must be placed in the record area pointed to by RECADR prior to calling WRITNX if LRL <> 0. If LRL=0 then WRITNX will write the data from the buffer area pointed to by BUFADR.

ENTRY CONDITIONS

A = 43
DE => FCB for currently open file
HL = Reserved for system use

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ

==========

DIRWR (SVC 44)
Direct Record Write

This routine will write the data in the record buffer directly to the specified recor of a currently open file. For variable length record files, you can only write to the beginning or to the end of a file; the end-of-file pointer is reset to the last record written. This restriction does not apply to fixed record length files.

If the file's logical record length (LRL) is not 256, then the data to be written must be transferred to the record buffer area pointed to by RECADR (see OPEN, above) prior to calling this routine. Otherwise the data will be written from the BUFADR area.

ENTRY CONDITIONS

A = 44
BC = Record number to be written. If BC=0 then write first record in file; if BC=FFFFH then write record at the end of file.
If Access Mode = "E" then BC points to a three-byte Extended record pointer in memory which contains the actual number of the record to be written. See OPEN, above, for details.
DE => File Control Block of currently open file
HL = Reserved for system use

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ.

==========

DATE (SVC 45)
Read/Set Real Time Clock

This routine will either set or read the system's real-time clock. Both time and date can be set or accessed with this routine. The data is returned as a 26-byte ASCII string arranged as follows:

SATAPR28197911813:20:42045

This string is composed of 10 fields, broken down as follows:

SAT APR 28 1979 118 13: 20: 42 04 5
--- --- -- ---- --- --- --- -- -- -

Field 1 = Name of day of week
Field 2 = Month
Field 3 = Day of month
Field 4 = 4-digit year
Field 5 = Numeric day of year
Field 6 = Hour
Field 7 = Minutes
Field 8 = Seconds
Field 9 = Numeric month
Field 10= Numeric day of week (0 = Monday)

Note: All date calculations are based on the Julian calendar.

ENTRY CONDITIONS

A = 45
B = Function Select
If B=0 then read time/date and store in buffer
If B=1 then set date using data from buffer pointed to by HL pair
If B=2 then set time using data from buffer pointed to by HL pair.
HL = If B=0, then HL points to a 26-byte buffer in memory to receive the data from the clock. If B=1, then HL points to a buffer containing the date. If B=2, then HL points to a buffer containing the time.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ

The user is allowed great flexibility in formatting date and time when setting the system's real time clock. Allowable separators are spaces, commas (,), dashes (-), slashes (/), periods (.), and colons (:). The following are all equivalent forms of TIME (column 1) and DATE (column 2):

TIME	DATE
03.05.00	09/01/82
03:05:00	9.1.1982
03:05	09/01/1982
3,5	9,1,82
3:5	9-1-1982
3 05	09 01 82
03-5	09,01,2 (modulo 1980)

==========

RENAME (SVC 47)
Rename a File

This routine will change the name and/or extension of a disk file. It will not change any assigned passwords.

ENTRY CONDITIONS

A = 47
DE => Pointer to buffer containing new file specification, terminated by a carriage return. A password may not be included.
HL => Pointer to buffer containing old file specification. If the file is password-protected, the proper password must be included as part of the file specification.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ.

==========

REWIND (SVC 48)
Position to the beginning of a file

This routine will reset the pointer information of an open file back to the beginning. The next READNX or WRITNX operation following this call will access the first record of the file. In order to use this call, the FCB of the affected file must be in an OPEN condition.

ENTRY CONDITIONS

A = 48
DE => FCB of file to "rewind"

EXIT CONDITIONS

Z = No error.
NZ = Error.
A = Error code if NZ.

==========

STSCAN (SVC 49)
String Scan Routine

This routine will scan through a user-defined buffer for a specified string. The string can contain ASCII codes from 0 through 255 (00H through FFH). The search will stop with the HL register pair pointing to the first character of the string in the buffer. If no match is found, the search will stop on the first carriage return encountered, or at the end of the buffer, whichever comes first.

ENTRY CONDITIONS

A = 49
B = Length of compare string
DE => Pointer to compare string
HL => Starting address of buffer to be searched.

EXIT CONDITIONS

Z = String found
NZ = String not found
HL => If Z flag set, points to the starting position of the matching string in the buffer. If NZ is set, indicating an error, then HL will be unchanged.

==========

SVC 50 is UNDEFINED

==========

WILD (SVC 51)
Wild Card Parser

This routine will compare a file specification with a comparison string containing wild-card characters. The wild card characters are:

*	Indicates a WORD mask
?	Indicates a CHARACTER mask.
!	Indicates a FILESPEC mask (includes both filespec and extension, equivalent to */* or **)

For example, */BAS will match with any file specification with the extension /BAS; ???ABC/BAS will match with any six-letter file specification whose last third, fourth and fifth letters are ABC AND has the extension /BAS. A "!" will match on ANY filespec.

ENTRY CONDITIONS

A = 51
B = Function select.

B = 0	Set wild card mask
B = 1	Compare filespec with mask
B = 2	Combine a filename with a separate extension to form a full file specification.
B = 3	Compare wildmask with "cracked" filespec.

When B=0, HL points to the starting address of the wild-card mask in memory, which must be terminated by a carriage return. Examples of wild card masks may be FOO???/BAR or */CMD. Note that wild card characters are not mandatory; an exact match on a filename such as DOTEST/CMD is perfectly valid.

When B=1, HL points to the file specification which is to be compared with the mask. The file specification must be terminated by a carriage return.

When B=2, HL points to an 11-byte buffer containing an 8-character filename and a 3 character extension. The filename and extension are both left-justified in their particular fields. For example, the file specification TYPE/S would be stored as TYPEbbbbSbb ('b's represent blanks, ASCII 32).

DE points to a 13-byte destination buffer for holding the compressed file specification contained in HL.

When B=3, then HL points to an 11-byte buffer containing an 8-character filename and 3-character extension, both left justified in their respective fields and padded with blanks. The mask will be compared against this filespec in its "cracked" form. For example, the wildmask DEL??/B?S would be compared against DELTAbbbBAS and the routine would return with a match. This makes it unnecessary to compress a filespec beforehand and can speed up a process of multiple comparisons.

EXIT CONDITIONS
When B=0, then :
NZ = Invalid mask specification
Z = Operation completed OK.
When B=1, then:
NZ = No match found, or no wildcard mask was set.
Z = Match found
When B=2, then:
DE => Points to a 13-byte buffer containing the file specification terminated by an ASCII 03H (ETX). Using the example above, on exit this buffer would contain TYPE/S followed by 03H.
When B=3, then:
NZ = No match found
Z = Match found

The user should note that to carry out a wild-card comparison, this SVC needs to be called once with B=0 to set the wild card mask, then once for each comparison. Unlike TRSDOS, the mask will be retained even if another SVC is executed, so it will be unnecessary to call WILD with B=0 once again.

==========

ERRMSG (SVC 52)
Load Error Message Into Buffer

This routine will load an 80-byte error message into a user-specified buffer area in memory. The buffer may be anywhere in free memory.

ENTRY CONDITIONS

A = 52
B = Code number of error message to be loaded
HL => 80-byte buffer for message

EXIT CONDITIONS

Z = No error.
NZ = Error
A = Error code if NZ.

==========

RAMDIR (SVC 53)
Load Disk Directory into RAM

This routine will load a directory entry (or an entire directory) into a user-defined RAM buffer. Only active files are read in (that is, KILLed files will not be read). Extended file directory entries (FXDE's) will NOT be read in by this routine.

ENTRY CONDITIONS

A = 53
B = Drive number (0 - 7)
C = Function select:
If C=0, entire directory is read into memory
If C=01H to FEH (1 to 254) then the corresponding entry in the directory will be read into memory in the format described below.
If C=0FFH (255 decimal) then free space information will be returned.
HL => Starting address of user's buffer where the directory is to be loaded. The size of the buffer will vary depending on the value of C.
If C=0, then buffer size = 34 * (number of user files) +1
If C=01H to FEH, then buffer size = 34 bytes.
If C=FFH (free space request) then buffer size = 4 bytes.
Note that the DOSPLUS II system will allow a maximum of 256 files on a pure data diskette or logical disk drive. The largest buffer size required would then be 8,705 (2201H) bytes.

EXIT CONDITIONS

Z = No error.
NZ = Error occurred.
A = Error code if NZ set.
Z = No error. HL registers point to the user buffer.

RAM Directory Format (contents of user buffer)

The directory in RAM will have the following format for each file record (when C=00H to FEH):

Buffer	+ 0	":" - Colon indicates start of each directory record.
	+ 1-15	FILENAME/EXT:D(cr). The filename, extension and a drive number from 0-7 for this record, followed by a carriage return. Note that the drive NUMBER, and not the drive name, is what is placed here. Any unused bytes following the carriage return (cr) will be padded with blanks.
	+ 16	"F" (ASCII 46H)
	+ 17	Logical Record Length. 0 implies 256 bytes.
	+ 18	Number of extents for this file.
	+ 19-20	Number of sectors allocated to the file in LSB/MSB format.

+ 21-22	Number of sectors actually used by the file in LSB/MSB format.
+ 23	EOF byte. Relative position of last byte of data in the last sector. 0 implies first position.
+ 24-25	Number of records written in LSB/MSB format
+ 26	User Attribute byte (ALWAYS 00)
+ 27	Protection level 0 through 7
+ 28-30	File creation date
+ 31-33	File creation date

When C=0, the last directory record will be followed by a "#" to signal the end of the directory. When C=01H to FEH the directory record will always end at the 34th byte without any trailing "#".

When C=0FFH, then free space information will be placed into a four-byte buffer in the following format:

Byte	0-2	number of free granules, in LSB/MSB format
	2-3	number of extents, in LSB/MSB format.

==========

SVC 54 is UNDEFINED.

==========

RS232C (SVC 55)
Initialize RS-232C Port

This routine will initialize either one of the Model II's two serial I/O ports, using the settings provided in a user-specified parameter list. This routine can also be used to turn off a serial channel.

ENTRY CONDITIONS

A = 55
B = Function select. If B=0, then turn off specified channel. If B <> 0, then turn channel on.
HL => Points to user-defined parameter list for serial I/O initialization.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ.

The parameter list is a contiguous area in memory arranged as follows:

parameter list

+ 0 = channel: "A" or "a" specifies ch. A, "B" or "b" specifies channel B.
+ 1 = Baud rate code (1 - 8):

1 - 110 baud	5 - 1200 baud
2 - 150 baud	6 - 2400 baud
3 - 300 baud	7 - 4800 baud
4 - 600 baud	8 - 9600 baud

+ 2 = word length (5 - 8 bits)
+ 3 = parity: "O" or "o" = Odd parity "E" or "e" = Even parity "N" or "n" = No parity
+ 4 = Number of stop bits (1 or 2)
+ 5 = if 00, parameter list ends here if 01, extended parameters follow

If parameter list+5 equals 01, then

+ 6-7 => Start of spool buffer for channel
+ 8-9 => End of spool buffer
+ 10 = 00 list terminator

The spool buffer length is calculated and automatically created and placed in high memory beneath any other drivers or buffers that may already be there, and the system's HIMEM pointer is lowered to protect it.

==========

SORT (SVC 56)
RAM Sort Routine

This routine will sort the contents of a block of memory. The entries to be sorted must all be the same length. The sort key may be shorter than or equal to the length of an entry.

ENTRY CONDITIONS:

A = 56
IX => Points to first entry of list to be sorted.
DE => Points to the start of last entry of list
B = Position of sort key relative to the first character of an entry
C = length of entry in the list
L = length of sort key (may not exceed value of C)
H = Sort flag. If H=0, then an ascending sort will be performed, otherwise a descending sort will be performed.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ.

==========

CLRXIT (SVC 57)
Exit to DOSPLUS II

This call will force an immediate exit to DOSPLUS II via the ABORT vector. Any chaining operations, or processing of multiple command lines, will be terminated. The user should make sure any open files are closed before calling this routine. This routine is maintained for compatibility with TRSDOS.

ENTRY CONDITIONS
A = 57

==========

FILPTR (SVC 58)
Get File Pointers

This routine will return information about any open file.

ENTRY CONDITIONS

A = 58
DE => Starting address of File Control Block. FCB must be in an open condition.

EXIT CONDITIONS

NZ = Error.
A = Error code if NZ.
Z = No error.

If no error occurred, then:

B = Drive number where file is resident (0 - 7)
C = Logical file number (position of file in the diskette's directory).

==========

SVCs 59 through 72 are UNDEFINED

==========

KBDW (SVC 73)
Keyboard Input with Wait

This routine will get one character from the keyboard type ahead buffer. Unlike KBCHAR, however, it will not return if there is no character available but will wait until a key is pressed on the keyboard or an error occurs.

If BREAK is pressed and the break processor is inactive, the routine will return with the break character (03H) in B. If the break processor is active, then control will pass to the break processor for handling of the break.

ENTRY CONDITIONS
A = 73

EXIT CONDITIONS
Z = No error
NZ = Error other than 02, "character not available."
B = keyboard code of character pressed.

==========

PRTS (SVC 74)
Get Printer Status

This routine will return the status of the printer in the A register. Only the high 4 bits are returned for the parallel driver. For the serial driver, only the status of the transmit buffer is returned. The user should monitor the condition of the Z flag to determine whether the printer is busy or not.

ENTRY CONDITIONS
A = 74

EXIT CONDITIONS
A = Printer status
NZ = Printer busy
Z = Printer not busy (can accept a new character)

Printer Status Bits (returned in the A register)
If the parallel driver is being used, then

Bit 7	1=printer busy, 0=printer not busy
6	1=paper out
5	1=printer deselected, 0=selected
4	1=no fault, 0=printer fault

If the serial driver is in use, then the condition of the Z flag will indicate the status of the spool buffer.

==========

CLAW (SVC 75)
Comm Line A Input with Wait

This routine will check Serial Port A for a valid INPUT character. It will not return until a valid character has been received.

ENTRY CONDITIONS
A = 75

EXIT CONDITIONS
Z = No error.
B = Character received from Serial Port A.
NZ = Error occurred. Register A will hold error code.

==========

CLBW (SVC 76)
Comm Line B Wait

This routine will check Serial Port B for a valid INPUT character. It will not return to the calling program until a valid input character has been received.

ENTRY CONDITIONS
A = 76

EXIT CONDITIONS
Z = No error
B = Character received from Serial Port B.
NZ = Error occurred.
A = Error code if NZ.

GET (SVC 77)

Input a Byte from a Device or File

This routine will input one byte from the specified device or open file. The device must be capable of input or both input/output for this routine to work. When inputting from a file, the logical record length may be any value, but this routine will still return a single byte.

ENTRY CONDITIONS

A = 77
DE => Points to device DCB or file FCB

EXIT CONDITIONS

Z = No error
B = Character input
NZ = Error
A = Error code if NZ.

==========

PUT (SVC 78)
Output a Single Byte to a Device or File

This routine will output a single byte to a device or open file. If the target device is set to NIL, this call will not return an error (that is, output to a NIL device is always permitted).

ENTRY CONDITIONS

A = 78
B = Byte to output
DE => points to device DCB or file FCB

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ

==========

POSN (SVC 79)
Position to a logical file Record

This routine will position to a specified logical record in a file so that the next I/O operation performed on the file (READNX or WRITNX) will access that record. The file must be in an OPEN condition before this SVC is called.

ENTRY CONDITIONS

A = 79
BC = record number to position to. If the file is in EXTENDED open mode (see OPEN, above) then BC must point to a three byte area in memory containing the record number to be positioned to.
DE => FCB of the file. The FCB must be in an OPEN condition.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ

==========

BKSP (SVC 80)
Backspace one logical record

This routine will backspace one logical record in an open file. The pointers will be adjusted so that the next I/O operation on the file will access the previous record.

ENTRY CONDITIONS
A = 80
DE => FCB of file. The FCB must be in an OPEN condition.

EXIT CONDITIONS
Z = No error
NZ = Error
A = Error code if NZ

==========

REWIND (SVC 81)
Rewind to beginning of file

This routine will reset the pointer information of an open file back to the beginning. The next file I/O operation following this call will access the first record of the file. In order to use this call, the FCB of the affected file must be in an OPEN condition.

ENTRY CONDITIONS

A	= 81
DE	=> FCB of file to rewind

EXIT CONDITIONS

Z	= No error
NZ	= Error
A	= Error code if NZ

==========

PEOF (SVC 82)
Position to End of File

This routine will set the pointers of an open file to the logical end of file, so that the next WRITNX operation (usually a write operation) will add a record to the file. If the file is newly created (that is, no previous write operations have been performed on it) then this call will set the pointers so that the next I/O operation will write the first record.

ENTRY CONDITIONS

A = 82
DE => FCB of file. The FCB must be in an OPEN condition.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ.

==========

EVAL (SVC 83)
Evaluate a command string

This routine will evaluate a command string for the presence of items in the four major fields: the SOURCE field, the DESTINATION field, the MASK field, and the PARAMETER field. It is assumed that the string to be evaluated by this call conforms to the DOSPLUS II command syntax. The string to be evaluated may reside anywhere in memory and should be terminated with one of the valid DOSPLUS II terminator characters (03H, carriage return, semi-colon or right brace).
The user should define a nine-byte EVAL block as given below before calling this routine.

NOTE: This routine calls two other routines, FSPEC and PARAM. The user should familiarize himself thoroughly with these two routines before attempting to use EVAL.

The first three fields of the command string are normally evaluated in a positional manner; that is, the source is assumed to be the first field, the destination is the second, and the mask field the third. However, the order of these fields can be changed by using the prepositions FROM, TO and USING to denote each field. EVAL will test for these prepositions and identify the fields correctly, so that a command such as COPY source destination mask can also be stated as COPY TO destination USING mask FROM source and still be correctly evaluated.

EVAL will set a bit in the first byte of the user-defined EVAL block for each field that is filled in the command line, and calls the FSPEC routine (SVC 84) to evaluate the contents of that field. If a comma or a left brace is encountered, then EVAL passes control to the PARAM routine (SVC 87) to complete the evaluation of the parameters. The parameters must be defined by the user program (see PARAM for the structure of the PARAMETER BLOCK).

If EVAL finds a wildmask in either the source or destination fields, it will move the wildmask into the mask field while retaining the associated drivespec. That is, a command such as

COPY */BAS:0 TO :10

will evaluate as

COPY :0 TO :10 USING */BAS

The FSPEC routine is called by EVAL to recover the filespec and place it into a file control block. If any parameters are present, then EVAL will pass control to the PARAM routine (SVC 87) which will complete the evaluation of the parameters in the command string.

ENTRY CONDITIONS

A = 83
IX => User-defined EVAL block
HL => Starting address of command string to be evaluated

EXIT CONDITIONS

Z = No error (Eval and parameter blocks are loaded)
IX = Unchanged.
HL => Terminating character

NZ => Error.

If NZ, then

A = error code
HL => character which caused the error

EVAL BLOCK: This is an 9-byte area in memory which will hold information about the command string that was evaluated.

Eval block+ 0	work byte for flag use
+ 1-2	Pointer to 41-byte SOURCE DCB
+ 3-4	Pointer to 41-byte DESTINATION DCB
+ 5-6	Pointer to 41-byte MASK DCB
+ 7-8	Holds starting address of the command string's parameter block.

On exit, the IX register pair points to the first byte of the EVAL block which contains information about the command string :

IX+0 :	bit 2	Set to 1 if the source field was filled
	bit 1	Set to 1 if the destination field was filled
	bit 0	Set to 1 if the mask field was filled.

On return, the calling program can easily determine which of the fields in the command string had data in them by executing an AND 7H with the EVAL block's first byte, that is, byte IX+0.

The structure of the PARAMETER BLOCK is explained under PARAM (SVC 87), below. If there are no parameters, then bytes 7 and 8 of the EVAL block must point to a memory location which contains a 00 byte. This is mandatory.

==========

FSPEC (SVC 84)
Fetch a filespec into a FCB

This routine will place a filespec into a File Control Block. The HL register points to the string containing the filespec, which may be anywhere in memory and should be properly terminated with a valid terminator character recognized by the DOSPLUS II system. DE should point to a 41 byte user-defined FCB. The first byte will be used by FSPEC as a work byte, and the filespec will be loaded into FCB+1 and following.

ENTRY CONDITIONS

A = 84
DE => 41 byte File Control Block
HL => Starting address of string containing the filespec

EXIT CONDITIONS

NZ = Error
A = Error code if NZ

If no error occurred, then:

HL => Address of terminating character of the string. If the filespec contains a wildmask with a drivespec then HL will point to the ":" of the drivespec.
DE => Unchanged.

On return, the FCB will contain the following:

FCB	+ 0	bit 7 : Set to 1 if the device field was filled. bit 6 : Set to 1 if the filespec field was filled. bit 5 : Set to 1 if the filespec contains wildcard characters bits 4-0 : Device number (may take a value from 0 to 15). If no drivespec is given, then 15 will be assumed. This is a global default which will force a search of all currently active drives in the system.
FCB	+ 1 and following	File specification, terminated with an 03H character.

The DE register pair must be incremented by 1 before any call to OPEN is made, in order to point to the proper position in the FCB for the OPEN call.

==========

RUN (SVC 85)
Load a Program from Disk and Execute

This routine will load a program from disk into memory and transfer control to the program's entry point as soon as it is loaded. The name of the program must have previously been placed in a File Control Block which may reside anywhere in free memory.

If the file terminates with an 03, then it will be loaded into memory but not executed, since the 03 indicates NO transfer address for that file ("load-only" files). In this case the RUN SVC will return to the calling program with the Z flag set.

ENTRY CONDITIONS

A = 85
DE => Starting address of FCB containing filespec of program to be executed.

EXIT CONDITIONS

NZ = Error
A = Error code if NZ.
Z = File was loaded, but ended with 03H rather than a transfer address.

==========

LOAD (SVC 86)
Load a File from Disk into Memory

This routine is similar to the RUN SVC. It will load a program file from disk into memory, however, the program will NOT be executed after loading. Control is passed back to the calling program.

ENTRY CONDITIONS

A = 86
DE => Starting address of File Control Block containing filespec of program file to be loaded.

EXIT CONDITIONS

Z = No error.
NZ = Error
A = Error code if NZ.

If no error occurred, then:

HL => Address of program file's entry point, if any.

==========

PARAM (SVC 87)
Evaluate a Parameter List

This routine will evaluate the parameter list of a command string. The command string may reside anywhere in memory and is pointed to by the HL register pair. The parameters are compared against a user-defined list in memory. Parameter words may be from 1 to 16 characters in length.

The EVAL supervisor call will continue into this routine if no error occurred, in order to complete the evaluation of a command string if one exists.

ENTRY CONDITIONS

A = 86
DE => Parameter list to compare string against
HL => Starting address of parameter string

EXIT CONDITIONS

Z = No error.
NZ = Error.
A = Error code if NZ.
HL => Character which caused the error.

If no error occurred, then:

DE = Unchanged
HL => Terminating character

PARAMETER LIST: This is a contiguous area in memory containing one or more parameter blocks and terminated by a 00 byte.

PARAMETER BLOCK: This is a user-defined string in memory which defines the parameter and the type of valid values it may take when specified in a command string. The structure of the parameter block is as follows:

Parameter block	+ 0:	bit 7: Set to 1 if the parameter can take a string expression. bit 6: Set to 1 if the parameter can take a numeric value. bit 5: Set to 1 if the parameter can take a switch (yes/no, on/off) value. bits 4 - 0: Length of parameter word minus 1
Parameter block	+ 1-2	Indirect address for the parameter data. These two bytes point to a location in memory where the parameter's value is to be loaded following evaluation. The address is specified in LSB/MSB format.
Parameter block	+ 3	The parameter string. This string and following must be in UPPER CASE, however the actual comparison with the string in the command line is done in a case-independent fashion. The length of this string is determined by bits 4-0 of the first byte of the parameter block.

If a VALID switch parameter is given in the command line without a specific value or expression, then FFFFH is assumed, defining a TRUE condition.

A switch value evaluates to FFFFH for true (YES/ON), and 00H for false (NO/OFF).

If a parameter takes a string expression, then the starting address of the string expression (NOT the quotation mark) is placed into the location pointed to by bytes 1 and 2 of the parameter block.

Parameters may take numeric values from 0 to 65535 (00H to FFFFH) only. The numeric values supplied as arguments to parameters may be in any of the acceptable bases recognized by DOSPLUS II, with the trailing base specifiers (H for hexadecimal, O or Q for octal, D for decimal, B for binary). The default base is decimal.

The PARAM routine will permit the use of only the first character of the parameter; that is, given the parameter SECTORS, it is not necessary to define a second parameter "S" if you want to be able to use the first letter of the parameter only. Defining SECTORS automatically makes both SECTORS and S valid parameters. If more than one parameter starts with the same letter, then the first-letter abbreviation will be valid for the FIRST parameter in the list which begins with that letter. That is, given two parameters SECTORS and SIDES, defined in the parameter list in that order, "S" would be valid only for SECTORS, which precedes SIDES in the list.

==========

FEXT (SVC 88)
Add a File Extension to a Filename

This routine will add a three-letter extension to a filename in an FCB if required, and may be used for adding default extensions. The filename in memory is pointed to by DE when this routine is called. The extension will only be added if none previously exists; if an extension is already present, then it will NOT be replaced.

ENTRY CONDITIONS

A = 88
DE => Starting address of filename in memory
HL => Starting address of string containing extension to the added (3 characters)

EXIT CONDITIONS

Z = The Z flag is set if the extension was NOT added (i.e., an extension already exists in the FCB)
NZ = If the Z flag is NOT set, then the extension was added to the filename in the FCB.

Register pairs HL and DE are unchanged on exit from this call.

==========

VALUE (SVC 89)
Evaluate a Numeric Value from a String

This routine will take a a numeric value in ASCII string form and convert it to a binary value using the base specifier associated with the string. The acceptable base specifiers are H for hexadecimal, O or Q for octal, D for decimal, and B for binary. If no base specifier is given then decimal is assumed.

Examples of numeric strings are:

07F8H	(hexadecimal)
6773Q	(octal)
7777O	(octal)
12387	(decimal default)
1982D	(decimal)
1010011B	(binary)

The value of any numeric string may not exceed 65535 decimal (FFFFH), since it will be converted into 16 bits. If the numeric string exceeds FFFFH, then it will be evaluated using the modulo if the defined base (decimal default). Also, numeric strings which exceed the maximum length will have the excess digits on the LEFT dropped; that is, 0F47252H will evaluate to 7252H. Overflow is NOT flagged on exit from this routine.

ENTRY CONDITIONS

A = 89
HL => Starting address of the ASCII-coded number in memory.

EXIT CONDITIONS

Z = No error.
NZ = Error. Invalid character was found in the numeric string.
A = Contains error code if NZ.
HL => Points to invalid character in string

If no error (Z set), then:

BC = Binary value of the numeric string.
HL = Points to string terminator character (carriage return, space, semicolon, right brace, etc.).

==========

SVC 90 is UNDEFINED

==========

SCREEN (SVC 91)
Screen Print

This routine will send the contents of the video display to any output device. Any character translation required (such as changing video graphics characters) must be done by filtering the destination device.

ENTRY CONDITIONS
A = 91
DE => Address of output device DCB or file FCB

EXIT CONDITIONS
Z = No error.
NZ = Error
A = Error code if NZ.

==========

NMICTL (SVC 92)
Add/remove Interrupt Tasks

This routine will insert or remove vectors into the RTC interrupt task table. The vectors point to a user-defined block in memory, called the Task Control Block or TCB. The TCB contains in its first two bytes the entry address of the user routine in memory which is to be executed under interrupt control. Other bytes may be assigned for data as required by the user routine. On entry to the user routine, the IX register will point to the TCB. The user routine can then recover any byte in the TCB by using IX plus an appropriate offset value as a pointer.

The TCB may reside anywhere in memory and need not be contiguous with the user routine, although for purposes of loading the routine from disk it may be easier to have the TCB and user routine contiguous.

Removing an already existing vector will effectively disable the routine. The RTC (real-time clock) Interrupt Task Table in DOSPLUS II has room for eight vectors (0 - 7), of which the first two (0 and 1) are available for user routines. The rest are used by the system for various background tasks such as the clock display, the printer spooler, the trace display, etc.

ENTRY CONDITIONS

A = 92
B = Insert/Remove flag. If B<> 0 then insert a vector into the interrupt task table. If B=0, then remove a vector from the table.
C = Slot number (0 to 7)
DE => Address of user routine if B<>0.

EXIT CONDITIONS

Z = No error
NZ = Error
A = Error code if NZ (parameter error)

SVC 93 is UNDEFINED

==========

VIDRAM (SVC 94)
Copy Video Memory into a RAM Buffer

This routine will copy the contents of video memory into a user defined buffer or vice versa. The user buffer in free memory must be large enough to hold a full screen of data, that is, 1920 bytes (24 lines x 80 characters).

ENTRY CONDITIONS

A = 94
B = Function select. If B=0 then copy the RAM buffer's contents into video memory. If B<>0 then copy the contents of video memory into the RAM buffer.
HL => Starting address of user's RAM buffer.

EXIT CONDITIONS

Z = No error.
NZ = Error
A = Error code if NZ.

==========

PRCTRL (SVC 95)
Printer Control Routine

This SVC permits the user to set various printer control options and check the status of printer-related functions. The options are selected by passing a option code in the B register. It may be necessary to call this routine several times, each time with a different code in B, to set up the printer control functions completely.

Printer control status is returned in the various registers on exit from the routine when option code 0 is passed to the routine.

ENTRY CONDITIONS

A = 95
B = Option code (0 - 0AH).
- 0 - Get printer status.
- 1 - Select serial printer driver (Serial Port B previously initialized).
- 2 - Select parallel printer driver
- 3 - Reset line counter to value in C.
- 4 - Reset character counter on current line to value in C.
- 5 - Begin transparent mode
- 6 - End transparent mode.
- 7 - Begin dummy mode
- 8 - End dummy mode
- 9 - Begin auto-LF mode
- A - End auto-LF mode

C = When B=3, contains value for line counter when B=4, contains value for character counter for current line.

EXIT CONDITIONS

NZ = Error
A = Error code if NZ.
Z = No error.

If B=0 on entry, then:

B = Page length
C = Maximum number of printed lines per page
D = Line length (max. Number of characters per line)
E = ASCII "P" if parallel printer driver selected; ASCII "S" if serial printer driver selected.
H = Number of characters printed since last carriage return
L = Number of lines printed since last top-of-form

Printer Control Options

Select Serial Printer Driver (B=1)

This will select a routine that will output all characters to Serial Port B. Serial Port B will have been initialized previously. It is not necessary to reinitialize Serial Port B unless you are changing any of the settings (baud rate, word length, etc.).

Select Parallel Printer Driver (B=2)

This will cause the output to go to the parallel printer port. DOSPLUS II comes preconfigured to select this port.

Set Line Count/ Set Character Count (B=3/B=4)

When B=3, the system's line counter will be reset to the value in the C register.

When B=4, the system's character counter for the current line will be reset to the value in the C register.

Begin Transparent Mode/End Transparent Mode

Normally, DOSPLUS II translates certain printer control codes rather than sending them out to the lineprinter as is. One example is a tab code (09H), which is translated to the appropriate number of spaces. Another example is a Form Feed code (0CH) which is translated into the appropriate number of carriage returns to achieve a top of form.

Setting transparent mode will override all translation by the system, and every character will be sent to the printer as is.

Set Dummy Mode/End Dummy Mode

Setting dummy mode will cause all printer output to be discarded without returning an error. This is analogous to setting the @PR device to NIL from DOSPLUS II command level.

The line and character counters will not be affected when output is discarded in this fashion.

Set Auto-LF/End Auto-LF

This option will cause each carriage return sent to the lineprinter to be followed by a line feed for those printers that need the CR-LF combination in order to linespace properly.

When this option is set, a linefeed (0AH) will always follow a carriage return (0DH), even in transparent mode.

ARCV (SVC 96)/BRCV (SVC 98)
Serial Port A/B Receive

These two routines perform character-oriented input from serial port A or B. The specified channel will have been previously initialized by the system. If you are going to use settings other than the power-up defaults, you must reinitialize the port with RS232C (SVC 55). In the DOSPLUS II system, each serial port is assigned a two character buffer by the system, but the user may establish a larger buffer through the SETCOM library command. Note that for each character, two bytes of buffer are required; one for the actual character data itself, and another for the status byte associated with that character.

When a character is input via either of these two SVCs, the oldest character in the buffer is returned in register B, and its associated status byte in register A. If the buffer is empty then the status byte returned in register A will contain the current status of the serial interface.

ENTRY CONDITIONS
A = 96 (for ARCV)
= 98 (For BRCV)

EXIT CONDITIONS
NZ = Error, or no character found
If NZ, then
IF A = 2, no character was found. Any other value is an error code.
Z = No error.
B = Character found in buffer, if any
A = Status byte associated with character in B
Bit 7: break sequence received
6: Framing error on last received character
5: Data lost due to buffer overflow
4: Parity error on last character received
3: No modem carrier present
2 - 0: Not used

==========

ATX (SVC 97)/BTX (SVC 99)
Serial Port A/B Transmit

These routines will perform character-oriented output to serial ports A or B. The serial ports will have been previously initialized by the system, and need not be reinitialized with the RS232C SVC (number 55) unless different settings are to be used.

Data will be transmitted whether or not a carrier tone is present. The Z-80 status flags and A register will return information about error conditions. The character to be transmitted must be loaded into register B before calling the appropriate SVC.

ENTRY CONDITIONS

A = 97 (for ATX)
= 99 (For BTX)
B = ASCII code of character to transmit.

EXIT CONDITIONS

Z = No error.
NZ = No character sent.
A = Status byte
Bit 7: Break sequence received
6: Framing error on last received character
5: Data lost
4: Parity error on last received character
3: No modem carrier present
2: Transmitter busy
1: Not used
0: Clear to Send (CTS) signal not present

==========

ACTRL (SVC 100)/BCTRL (SVC 101)
Serial Channel A/B Control

These two routines control the RS232C interface for channels A and B.

ENTRY CONDITIONS

A = 100 (for ACTRL)
= 101 (For BCTRL)
B = Command code:
B= 0: Get status of serial channel into register B
1: Always returns 0.
2: Turn on Request to Send (RTS)
3: Turn off Request to Send
4: Begin transmission of break sequence
5: Turn off break sequence
6: Reset input buffer count to zero
7: Reset SIO error condition

EXIT CONDITIONS

Z = No error.
NZ = Error occurred.
A = Error code if NZ

If no error occurred, then

B = Status of serial port:
Bit 7: Break sequence now being received
6: Framing error in byte being received
5: Data overflow on byte being received
4: Parity error on byte being received
3: Modem carrier not present
2: Transmitter busy
1: Not used
0: Clear to Send (CTS) not present

==========

The DOSPLUS II Resident Jump Table

DOSPLUS II has a number of system routines which may be utilized by user programs, although they are not part of the SVC Calls. These routines are accessed via a jump table on page 3 of memory (0300H). To use these routines, you should load the Z80 register pairs as indicated below, and execute a CALL to the specific address of the routine that you want to execute.

JPINIT (0300H)

This is the system return vector, which is used to reinitialize the various devices prior to exiting back to DOSPLUS II from a user program which uses its own I/O drivers. If the DOSPLUS II devices are not reinitialized, exiting from a user program may leave the devices inactive, forcing you to reboot the system.

This vector is the same as SVC 31 (JPINIT) and may be called either with a jump to 300H or loading the A register with 31 and executing a RST 8 instruction.

REGSAV$ (0303H)

This routine will save all the primary registers on the stack. The contents of the AF registers are not preserved on return from this call. The contents of the AF' registers are destroyed. A RET instruction will restore all the registers.

SAVERN$ (0306H)

Reserved for system use only. Do not use.

SAVER$ (0309H)

Reserved for system use only. Do not use.

SMULT$ (030CH)

This routine will multiply two eight-bit values and return with the result in the A register. On entry, register E must contain the multiplicand, and Register A the multiplier. The result of the operation cannot exceed FFH.

SDIVD$ (030FH)

This routine will divide the 8-bit value in the E register by the value in the A register and return the result in A. The remainder, if any, will be returned in E.

DMULT$ (0312H)

This routine will multiply a 16-bit value in the HL registers by an 8-bit value in the A register. The result is returned as three bytes. The HL register pair contains the highest and next significant bytes, in that order, and the A register contains the least significant byte.

DDIVD$ (0315H)

This routine will divide a 16-bit value in the HL registers by an 8-bit value in the A register. On return, the HL register pair will contain the 16-bit quotient, and the A register will contain the remainder, if any.

TMULT$ (0318H)

This routine will multiply a 24-bit value by an 8-bit value. The 24-bit multiplicand is contain in the B, H and L registers, with B containing the most significant byte, H the next significant, and L the least significant byte. The multiplier must be placed in the A register.

The operation returns a 32-bit result. On return, the B, H and L registers contain the most significant bytes in that order, and the A register contains the least significant byte.

TDIVD$ (031BH)

This routine will divide a 24-bit value in the B, H, and L registers by an 8-bit value in the A register. The result is a 24-bit value also returned in the B, H and L registers with the remainder, if any, in register A.

UCASE$ (031EH)

This routine converts a lower-case alpha character to uppercase. The ASCII code of the character to be converted must be in the A register, and its upper case equivalent is returned also in A.

Note that ASCII codes less than 61H (97 decimal) or greater than 7AH (122 decimal) will NOT be converted.

DBLOCK$ (0321H)

Reserved for system use only. Do not use.

DIRRED$ (0324H)

This routine will read the directory entry for a specific file into the system's file buffer. The Logical File Number (LFN) must be placed in the B register, and the C register must contain the drive number of the disk drive to be read (0 to 7). On exit, the BC register pairs are unchanged, and the HL register pair will be pointing to the system buffer where the directory entry is located.

The LFN is byte is calculated as follows: the leftmost three bits form the directory entry offset within a sector (each directory sector contains eight 32-byte directory entries) and the rightmost five bits form the sector within the directory MINUS TWO where the file entry is found. To get the correct sector number, add 2 to this value.

Thus an LFN of 00 points to the first entry position (byte offset 0) of sector 2 of the directory (0 +2), and an LFN of D7H points to the 6th directory entry (relative byte C0H) in sector 19H of the directory.

CALCULATING THE LOGICAL FILE NUMBER FOR A DIRECTORY ENTRY

LFN=0D7H

```
                    1 1 0  1 0 1 1 1   = D7H (215 decimal)
                    ------ --------
Field -->             A        B
```

FIELD A: Forms the directory entry offset within a sector, and can take the values 0-7. The starting position of the directory entry in the sector may be found by ANDing the LFN byte with E0H.

FIELD B: Forms the SECTOR NUMBER - 2 within the directory where the file is located (5 Bits). The sector number can be derived by loading the LFN into the A register and performing the following instructions:

```
AND 1FH
ADD A,2
```

The resulting value in the A register will be the sector number where the directory entry is located.

In the example above, if D7H is ANDed with E0H, the result will point to the file entry that begins at relative byte C0H. If D7H is ANDed with 1FH the result is 17H. Adding 2 to this value yields 19H, which is the directory sector where the entry is located.

DIRWRT$ (0327H)

This routine peforms the converse of DIRRED$. It will write a directory entry for a specific file. The directory information must be contained in the system's buffer. The LFN is placed in the B register, and the drive number to be written (0 through 7) in the C register.

LOCDCB$ (032AH)

This routine will return the location of the Device Control Block of any device. The device number must be placed in the A register. On return the IX register will contain the starting address of the specified device's DCB. The valid device numbers are from 0 to 7 (Disk devices are not included).

LOCDCT$ (032DH)

This routine will return the location of the Drive Control Table for any active disk drive. The drive number (0 through 7) is placed in the C register prior to calling this routine. On return, the IY register will contain the starting address of that drive's DCT.

FETMEM$ (0330H)

This routine will create a protected block in high memory for use by your programs. The number of bytes needed for the block is placed in the BC register pair prior to calling this routine. On return, the Z flag will indicate the status of the operation. If NZ, then an error occurred (insufficient memory), and A contains the error code.

If the Z flag is set, then the operation was successful. The contents of the BC register pair will be unchanged, and HL points to the first free memory address in the block. This value will be HIMEM+1 (HIMEM is adjusted accordingly).

RECMEM$ (0333H)

This routine will reclaim blocks of protected high memory. Only the lowest protected block can be reclaimed. Place the start address the block to be reclaimed in the HL registers (if FETMEM$ was called, this value will already be correct), and the number of bytes to reclaim in the BC registers, then call RECMEM$. On return, HIMEM will be reset to the new value. If HL equals the contents of HIMEM+1 then the operation was successful.

VDON$ (0336H)

This routine will enable the video memory, which is bank switched with the high 4K of memory. This routine should be used to switch video memory rather than any other user routine, or changing the DCB values. This routine will also relocate the stack. Do not change the stack pointer while video memory is enabled!

The contents of the AF registers are destroyed by this call.

VDOFF$ (0339H)

This routine will deselect video memory and restore normal memory in the top 4K. It will also restore the stack to its normal location. The contents of AF are destroyed.

BRKVEC (033CH)

This is the system break vector. It is used to set the break processor to exit to DOS. Taking this exit will ensure that the break processor status is not disturbed on exit.

==========

The Resident Disk I/O System

DOSPLUS II has an I/O package in its resident module which permits direct access to the disk drives independently of the file-oriented I/O routines. The disk routines are accessed by a RST 10H instruction, with the routine number in the A register, and the other registers loaded as appropriate. The Resident Disk I/O system supports all DOSPLUS II supported devices.

Register usage is as follows (some calls may not require that ALL of these registers be loaded):

HL => Starting address of user's buffer
E = LSB of sector number to access
D = Cylinder number to access
C = Drive number to access (0 - 7)
B = MSB of sector number
A = I/O routine number

Note that the B and E registers are used to code the sector number which is to be accessed.

On return, the state of the Z flag will signal whether or not the operation was completed successfully. If NZ is set, then an error occurred during the operation, and the A register contains the error number. If Z is set, then the operation was successful and HL will point to the user's data buffer.

The available routines are as follows:

A = 0	Check if drive ready. On return, if the Carry flag is set, then the disk is write-protected. Registers required: C Register A returns drive status.
A = 1	Home drive head to cylinder 0 and initialize drive. Registers required: C
A = 2	Seek specified cylinder/sector address. Registers required: B, C, D, E
A = 3	Read specified sector with Seek operation. Registers required: B, C, D, E, HL. Register A returns status.
A = 4	Verify specified sector with Seek operation. Registers required: B, C, D, E, HL.
A = 5	Write specified sector with Seek operation. Registers required: B, C, D, E, HL. Register A returns status.
A = 6	Read Directory sector with Seek. Registers required: B, C, E, HL.
A = 7	Write directory sector with Seek. Registers required: B, C, E, HL
A = 8	Format specified cylinder with Seek. Registers required: C, D, HL.
A = 9	Write directory sector using data address marks as flagged in drive's DCT. Registers required: B,C, E, HL.

Routines 3 through 9 will execute a seek to the specified cylinder/sector location before proceeding with the I/O operation.

Routines 6 and 7 will go automatically to the directory cylinder; that is, the cylinder number specified in the DCT when the call is made will be ignored. The location of the directory cylinder will be taken from the disk's boot sector. Routine 9 will write a directory sector after checking the appropriate DCT for the type of data address marks (DAMs) to use. If the DCT indicates that "read-protected" data address marks are to be used, then the directory sector will be written out with those DAMs.

"Read-protected" DAMs do not actually prevent a directory sector from being read; rather they serve to differentiate the directory cylinder and sectors from all other tracks on the disk for easy identification by the system.

These I/O routines do not make use of the disk directory to locate any particular file. It is the user's responsibility to keep track of the locations being accessed. He should not assume that merely reading or writing a sector to the disk will cause its location to be coded somewhere in the operating system.

It is important to note that when using these calls, the DCT (Drive Control Table) for a particular drive must NOT be assumed to hold current information. A drive's DCT is NOT updated until a directory sector is read. Reading any other sector will not force a DCT update. For example, using routine 0 to check a drive's ready status does not update the DCT.

To ensure that the DCT is correct for that particular diskette, set bit 0 of byte DCT+5 (using the DCT for the particular drive being operated on) to 1 and execute a read of a directory sector, prior to performing any disk I/O operations.

System Error Messages

These are the error codes returned by the system in the A register when the NZ flag is set on return from a Supervisor Call. The corresponding error messages are given for each error code. These messages are the same ones returned by the ERROR library command.

Dec Hex	Error
000 (00H)	No error
001 (01H)	Bad function code on SVC call
002 (02H)	Character not available
003 (03H)	Parameter error
004 (04H)	CRC error on disk I/O operation
005 (05H)	Disk sector not found
006 (06H)	Unknown error
007 (07H)	Illegal disk change
008 (08H)	Disk drive not ready
009 (09H)	Invalid data provided
010 (0AH)	Unknown error
011 (0BH)	File already exists
012 (0CH)	No drive available to open file
013 (0DH)	Write attempt to Read-Only file
014 (0EH)	Write fault on disk I/O
015 (0FH)	Write-protected disk
016 (10H)	DCB has been modified
017 (11H)	Directory read error
018 (12H)	Directory write error
019 (13H)	Invalid filespec
020 (14H)	GAT read error
021 (15H)	GAT write error
022 (16H)	Hit read error
023 (17H)	HIT write error
024 (18H)	File not found
025 (19H)	File access denied
026 (1AH)	Directory space full
027 (1BH)	Disk space full
028 (1CH)	End of file encountered
029 (1DH)	Record number out of range
030 (1EH)	Directory space full -- cannot extend file
031 (1FH)	Program not found
032 (20H)	Invalid drivespec
033 (21H)	Unknow error
034 (22H)	Load file format error
035 (23H)	Memory fault
036 (24H)	Unknown error
037 (25H)	Open attempt to an Open file
038 (26H)	I/O attempt to an Unopen file
039 (27H)	Illegal I/O attempt

Dec Hex	Error
040 (28H)	Seek error
041 (29H)	Lost data on disk I/O
042 (2AH)	Printer not ready
043 (2BH)	Printer out of paper
044 (2CH)	Printer fault
045 (2DH)	Printer not available
046 (2EH)	Invalid I/O attempt to VLR type file
047 (2FH)	Required command parameter not found
048 (30H)	invalid command parameter
049 (31H)	Hardware fault on disk I/O
050 (32H)	Unknown error
051 (33H)	CRC error on header read
052 (34H)	Seek error on read
053 (35H)	Lost data on read
054 (36H)	CRC error on read
055 (37H)	Data record not found on read
056 (38H)	Attempt to read System data record
057 (39H)	Attempt to read deleted data record
058 (3AH)	Device not available
059 (3BH)	CRC error on header write
060 (3CH)	Seek error on write
061 (3DH)	Lost data on write
062 (3EH)	CRC error on write
063 (3FH)	Data record not found on write
064 (40H)	Write fault on disk drive
065 (41H)	Disk write protected
066 (42H)	No function exists
067 (43H)	Drive not in system
068 (44H)	Invalid devicespec
069 (45H)	No device space available
070 (46H)	Device not available
071 (47H)	Device in use
072 (48H)	Protected system device
073 (49H)	Modem carrier lost
074 (4AH)	Transmitter not available
075 (4BH)	Framing error
076 (4CH)	Serial data lost
077 (4DH)	Serial parity error
078 (4EH)	Clear to Send not detected
079 (4FH)	Illegal Logical File Number
080 (50H)	Invalid input channel
081 (51H)	Invalid output channel
082 (52H)	Insufficient memory
083 (53H)	Device already exists
084 (54H)	Data set member not found
085 (55H)	Terminated
086 (56H)	Program terminated
087 (57H)	Invalid data set member

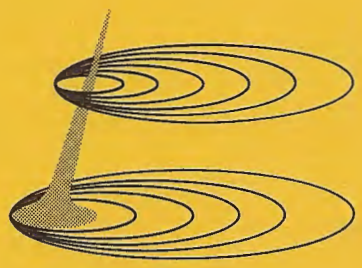
Dec Hex	Error
088 (58H)	Function not available
089 (59H)	Illegal Input/Output channel
090 (5AH)	GAT table corrupt
091 (5BH)	HIT table corrupt
092 (5CH)	No extent terminator on file
093 (5DH)	Directory links to record not linking back to it
094 (5EH)	Track assigned beyond disk boundary
095 (5FH)	Forward directory link to inactive entry
096 (60H)	Forward link to non-extension entry
097 (61H)	Cannot establish disk type
098 (62H)	Illegal data range
099 (63H)	End of File sector beyond allocated sectors
100 (64H)	Extension assigned before end of extent
101 (65H)	Extension record not assigned to any file
102 (66H)	Multiple files assigned to granule
103 (67H)	Directory record has an invalid HIT byte
104 (68H)	Directory record has a Zero HIT byte
105 (69H)	Extended directory record has an invalid HIT byte
106 (6AH)	Extended directory record has a Zero HIT byte

Some of these error codes are used by the utilities that are supplied with the DOSPLUS II system. In particular, error codes 84 through 106 are used by utilities such as DIRFIX (see Utilities manual).

TRS-80™
Model II

EDITOR ASSEMBLER

EDAS 4.0



by Galactic Software Ltd.

****
****
******** GALACTIC SOFTWARE EDITOR ASSEMBLER 4.0 *******
******** COPYRIGHT 1980, BY GALACTIC SOFTWARE *******
****
****

TABLE OF CONTENTS

Section 1

Section 2

First Edition 1980

GALACTIC SOFTWARE LTD. MODEL II EDITOR ASSEMBLER 4.0
by Roy Soltoff, MISOSYS and Bill Schroeder, GALACTIC

GALACTIC's MODEL II EDITOR ASSEMBLER includes the following features:

1> Depression of <ENTER> without a command immediately provides a summary listing of the Editor Assembler's available commands.

2> "Load" and "Write" text buffers from/to the disk, as well as assembled object code filed to disk as a directly executable program file.

3> "Move" block allows the user to move lines of text from one location to another location in the text file.

4> "Find" will locate a designated string within the text. It will stop at each location and may be manually continued to the next occurence of the string.

5> "Global" replace allows the user to change a sequence of characters (STRING1) to a different sequence of characters (STRING2) throughout the entire text buffer.

6> "System" allows the user to perform any TRSDOS library command from the editor and return to the editor. You may do a DIR or maybe a KILL or LIST and even enter the DEBUGGER and never leave the environment of the Editor Assembler. Use this command to set FORMS, perform disk INIT, or CREATE files

7> "Edit" provides straightforward editing of designated text lines. The Editor maintains command syntax identical to the Model II BASIC editor while all editing is done in reverse video providing excellent user interface. Line insert, replace, and renumber round out the Editor's complement of commands.

8> "Assemble" with numerous switches is provided to allow for many different types of assembled output. PLUS the unique assemble to memory system, so the user may effectivly debug the object code before leaving the Editor Assembler environment.

9> "Jump" allows you to execute your program, that has been assembled to ram, and then return to the Editor Assembler.

10> "Usage" allows quick reference to the present usage of available ram by printing the number of bytes remaining in your text buffer, how many are in use, plus the address of the first free byte after the text. This last address is useful when using the ASSEMBLE TO MEMORY feature.

11> "Hardcopy" will print all or part of the text buffer.

12> "Type" will print the text buffer without line numbers.

13> The <F1> key is employed as a functional <CLEAR> key.

14> The symbol table is sorted in ascending alphanumeric order and output 5-across in 80 column format.

15> The <UP ARROW> and <DOWN ARROW> keys provide instant scroll up or down, one line at a time, or repeated, with the repeat key.

16> The <F2> key is employed to provide instant advance of a entire text page (23 lines). This "PAGE" function may also be used with the repeat key.

17> The <HOLD> key is employed as a functional Pause key.

18> All Editor Assembler commands may be entered in either upper case or lower case providing ease of operation as a text editor.

19> Great amounts of time and effort were expended to give the user of this Editor Assembler the absolute best in ease of operation and functional efficiency. Optimize assembly programing time, with the EDITOR ASSEMBLER designed with the programmer in mind.

W A R R A N T Y

ALL GALACTIC SOFTWARE IS WARRANTED FOR ONE YEAR FROM THE DATE OF PURCHASE TO BE FREE FROM CODING DEFECTS. SHOULD A PROBLEM BE FOUND BY THE USER, GALACTIC WILL CORRECT IT AT NO CHARGE, DURING THE WARRANTY PERIOD. THIS SERVICE SHALL ONLY BE PROVIDED TO CUSTOMERS WHO REGISTER WITH GALACTIC, WITHIN 30 DAYS OF PURCHASE, BY RETURNING THE REGISTRATION CARD PROVIDED IN THE REAR OF THE MANUAL. GALACTIC'S LIABILITY SHALL NOT, IN ANY WAY, EXTEND BEYOND CORRECTION OF THE PROGRAM ITSELF. THE USER IS TOTALLY RESPONSIBLE FOR ALL DATA ENTRUSTED TO THE PROGRAM AND HIS HARDWARE.

INTRODUCTION

The Galactic Software Editor Assembler is a RAM-resident text editor and assembler for the Model II microcomputer system. The Editor Assembler was designed to provide the maximum in user interface and ease of use while providing capabilities powerful enough for the expert Z80 assembly language programmer.

The text editing features of the Editor Assembler facilitate the manipulation of alphanumeric text files. The most common use of the editing capability is in the creation and maintenance of assembly language source programs.

The assembler portion of the Editor Assembler facilitates the translation of symbolic language source code programs into machine executable code. This object code may then be executed directly from TRSDOS (tm) as a program file. Previous knowledge of machine language and the hexadecimal number system is assumed throughout this manual.

The Assemble command (A) supports the assembler language specifications set forth in the ZILOG "Z80-ASSEMBLY LANGUAGE PROGRAM MANUAL, 3.0 D.S., REL.2.1, FEB 1977, with the following exceptions.

Macros are not supported.

Operand expressions may only contain the "+", "-", "&" (logical AND), and "<" (shift) operators, and are evaluated on a strictly left to right basis. Parentheses are not allowed!

Conditional assembly commands, where a programmer may control which portions of the source code are assembled, are not supported.

Constants may only be decimal (D), hexadecimal (H), or octal (O).

The only Assembler commands supported are *LIST OFF and *LIST ON.

A label can contain only alphanumeric characters. (Use of the "-" and "?" is not supported). A label can be up to 6 characters long. The first character must be alphabetic. The other characters must be alphanumeric.

NOTATION CONVENTIONS

()	Parentheses enclose optional information. They are never input in Editor Assembler commands.
...	The ellipses represents repetition of a previous item.
line	Any decimal number from 1 to 65529.
.	A period may be used in place of any line number. It represents a pointer to the current line of source code being assembled, printed, or edited.
T	The character <T> may be used in place of any line number. It represents the top of the text buffer.
B	The character <B> may be used in place of any line number. It represents the bottom of the text buffer.
inc	A number representing an increment between successive line numbers.

All Editor Assembler commands may be entered in lower case as well as upper case to facilitate its use as a general purpose text editor. Assembler source code must be entered in upper case only. It is suggested that <CAPS LOCK> be used to enter source code.

A file called "Z80CODE/SOR" has been written to the diskette containing the Galactic Software Editor Assembler. This file is a Z-80 code source file containing the entire Z-80 code instruction set which can be loaded into the Editor Assembler. If assembled, it will produce Z-80 object code in numeric order. The generated listing will be similar to the NUMERIC LIST OF INSTRUCTION SET located at the rear of your Editor Assembler manual.

GETTING STARTED WITH EDAS 4.0

It is strongly recommended that before using your new Editor Assembler, you should make a BACKUP copy to use in a working environment and retain the EDAS diskette as your MASTER copy. The BACKUP utility procedures are found in your "TRS-80 Model II Owner's Manual" in the section entitled "UTILITY PROGRAMS". After creating a BACKUP copy of the EDAS diskette, store the MASTER diskette in a safe place. Use only your "working" copy for production.

EDAS 4.0 is a directly executable program file. It is accessed simply by entering:

EDAS40

in response to the TRSDOS query

```
TRSDOS READY
.................................................
```

EDAS 4.0 will load, execute, and display the following:

```
GALACTIC SOFTWARE LTD.  MODEL II EDITOR ASSEMBLER 4.0
 By Roy Soltoff, MISOSYS and Bill Schroeder, Galactic
      Copyright 1980 by Galactic Software ltd.
 >
```

The right carat ">" which appears in reverse video, is the prompting character displayed by EDAS whenever it is ready to accept a command. If you would like a memory jogger as to what commands are acceptable to EDAS, just depress <ENTER>. The entire command repertoire will be instantly displayed.

WELCOME TO THE WORLD OF SOPHISTICATED BUSINESS SOFTWARE

ASSEMBLY LANGUAGE

Syntax

The basic format of an assembly language statement is:

(LABEL) OPCODE (OPERAND(S)) (COMMENT)

LABELS

A label is a symbolic name of a line of code. Labels are always optional. A label is a string of characters no greater than 6 characters. The first character must be a letter (A-Z). A label may not contain the dollar sign ($) character. The dollar sign ($) is reserved for the value of the reference counter of the current instruction.

The following labels are reserved for referring to registers only and may not be used for other purposes:

A, B, C, D, E, H, L, I, R,
IX, IY, SP, PC, AF, BC, DE, and HL.

The following 8 labels are reserved for branching conditions and may not be used for other purposes (these conditions apply to status flags):

FLAG	CONDITION SET	CONDITION NOT SET
Carry	C	NC
Zero	Z	NZ
Sign	M (minus)	P (plus)
Parity	PE (even)	PO (odd)

OPCODES

The OPCODES for the Galactic Software Model II Editor Assembler correspond to those in the Z-80-ASSEMBLY LANGUAGE PROGRAMMING MANUAL, 3.0 D.S., REL 2.1, FEB 1977.

OPERANDS

Operands are always one or two values separated by commas. Some instructions require no operands at all.

A value in parentheses "()" specifies indirect addressing when used with registers, or "contents of" otherwise.

Constants may end in any of the following letters:

H - hexadecimal

D - decimal

O - octal

A constant not followed by one of these letters is assumed to be decimal. A constant must begin with a digit. Thus "FFH" is not permitted, while "0FFH" is valid.

Expressions using the "+", "-", "&", and "<" operators are described in the section, Expressions.

COMMENTS

All comments must begin with a semicolon (;). If a source line starts with a semicolon in column 1 of the line, the entire line is a comment.

EXPRESSIONS

A value of an operand may be an expression consisting of "+", "-", "&", or "<" symbols. These operations are executed in strictly left to right order. No parentheses are allowed. All four operators are binary. Both "+" and "-" have unary uses also.

Addition (+)

The plus will add two constants and/or symbolic values. When used as a unary operator, it simply echoes the value.

Examples:

```
001E    CON30   EQU   30

0010    CON16   EQU   10H

0003    CON3    EQU   3

002E    A2      EQU   CON30 + CON16
```

Subtraction (-)

The minus operator will subtract two constants and/or symbolic values. Unary minus produces a 2's complement.

Examples:

```
000E    A2      EQU   CON30 - CON16

FFF2    A4      EQU   -A2
```

Logical AND (&)

The logical AND operator logically adds two constants and/or symbolic values.

Examples:

```
3C00    A1      EQU   3C00H & 0FFH

0000    A2      EQU   0 & 15

0000    A3      EQU   0AAAAH & 5555H
```

Shift (<)

The shift operator can be used to shift a value left or right. The form is:

```
VALUE     <     AMOUNT
```

If AMOUNT is positive, VALUE is shifted left. If AMOUNT is negative, VALUE is shifted right. The magnitude of the shift is determined from the numeric value of AMOUNT

Examples:

```
C000    A1      EQU   3C00H < 4

03C0    A2      EQU   3C00H < -4

BBFF    A3      EQU   3CBBH < 8 + 255

03C0    A3      EQU   15 + 3C00H < -4
```

Z-80 STATUS INDICATORS (FLAGS)

The flag registers (F and F') supply information to the user regarding the status of the Z-80 at any given time. The bit positions for each flag are shown below:

7	6	5	4	3	2	1	0
S	Z	X	H	X	P/V	N	C

where:

C = Carry flag
N = Add/Subtract flag
P/V = Parity/Overflow flag
H = Half-carry flag
Z = Zero flag
S = Sign flag
X = Not used

Each of the two Z-80 flag registers contain 6 bits of status information which are set or reset by CPU operations. Four of these bits are testable (C, P/V, Z, and S) for use with conditional jump, call, or return instructions. Two flags (H, N) are not testable and are used for BCD arithmetic. Two flag register bits (3, 5) are not used by the Z-80.

CARRY FLAG (C)

The carry flag is set or reset depending on the operation being performed. For "ADD" instructions that generate a carry and "SUBTRACT" instructions that generate a borrow, the carry flag will be set. The carry flag is reset by an "ADD" that does not generate a carry and a "SUBTRACT" that generates no borrow. This saved carry facilitates software routines for extended precision arithmetic. Also, the "DAA" instruction will set the carry flag if the conditions for making the decimal adjustment are met.

For instructions RLA, RRA, RLS, and RRS, the carry bit is used as a link between the LSB and MSB for any register or memory location. During instructions RLCA, RLC s and SLA s, the carry contains the last value shifted out of Bit 7 of any register or memory location. During instructions RRCA, RRC s, SRA s, and SRL s, the carry contains the last value shifted out of Bit 0 of any register or memory location.

For the logical instructions AND s, OR s, and XOR s, the carry flag will be reset.

The carry flag can also be set (SCF) or complemented (CCF).

ADD/SUBTRACT FLAG (N)

This flag is used by the decimal adjust accumulator instruction (DAA) to distinguish between "ADD" and "SUBTRACT" instructions. For all "ADD" instructions, N will be set to a "zero". For all "SUBTRACT" instuctions, N will be set to a "one".

PARITY/OVERFLOW FLAG (P/O)

This flag is set to a particular state depending on the operation being performed.

For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (-128). This overflow condition can be determined by examining the sign bits of the operands.

For addition, operands with different signs will never cause overflow. When adding operands with like signs and the result has a different sign, the overflow flag is set. For example:

```
  +120  =  0111 1000     ADDEND
  +105  =  0110 1001     AUGEND
 ---------------------------------
  +225  =  1110 0001    (-95) SUM
```

The two numbers added together has resulted in a number that exceeds +127 and the two positive operands has resulted in a negative number (-95) which is incorrect. The overflow flag is therefore set.

For subtraction, overflow can occur for operands of unlike signs. Operands of like sign will never cause overflow. For example:

```
   +127  =  0111 1111     MINUEND
(-)-64   =  1100 0000     SUBTRAHEND
 ---------------------------------
   +191  =  1011 1111     DIFFERENCE
```

The minuend sign has changed from a positive to a negative giving an incorrect difference. The overflow flag is therefore set.

Another method for predicting an overflow is to observe the carry into and out of the sign bit. If there is a carry in and no carry out, or if there is no carry in and a carry out, then overflow has occurred.

This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of "one" bits in a byte are counted. If the total is odd, "ODD" parity (P=0) is flagged. If the total is even, "EVEN" parity is flagged (P=1).

During search instructions (CPI, CPIR, CPD, and CPDR) and block transfer instructions (LDI, LDIR, LDD, and LDDR), the P/V flag monitors the state of the byte count register (BC), When decrementing the byte counter results in a zero value, the flag is reset to zero, otherwise the flag is a one.

During "LD A,I" and "LD A,R" instructions, the P/V flag will be set with the contents of the interrupt enable flip-flop (IFF2) for storage or testing.

When inputting a byte from an I/O device "IN r,(C)", the flag will be adjusted to indicate the parity of the data.

THE HALF CARRY FLAG (H)

The half carry flag (H) will be set or reset depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by the decimal adjust accumulator instruction (DAA) to correct the result of a packed BCD add or subtract operation. The H flag will be set (1) or reset (0) according to the following table:

H	ADD	SUBTRACT
1	There is a carry from Bit 3 to Bit 4	There is no borrow from Bit 4
0	There is no carry from Bit 3 to Bit 4	There is a borrow from Bit 4

THE ZERO FLAG (Z)

The Zero flag (Z) is set or reset if the result generated by the execution of a certain instruction is a zero.

For 8-bit arithmetic and logical operations, the Z flag will be set to a "one" if the resulting byte in the Accumulator is zero.

For compare (search) instructions, the Z flag will be set to a "one" if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Z flag will contain the state of the indicated bit.

When inputting or outputting a byte between a memory location and an I/O device (INI, IND, OUTI, or OUTD), if the result of B-1 is zero, the Z flag is set, otherwise it is reset. Also for byte inputs from I/O devices using "IN r,(C)", the Z flag is set to indicate a zero byte input.

THE SIGN FLAG (S)

The Sign flag (S) stores the state of the most significant bit of the accumulator (Bit 7). When the Z-80 performs arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a "zero" in bit 7. A negative number is identified by a "one". The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 to 127. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is from -1 to -128.

When inputting a byte from an I/O device to a register, "IN r,(C)", the S flag will indicate either positive (S=0) or negative (S=1) data.

PSEUDO-OPS

There are nine pseudo-ops (Assembler directives) which the assembler will recognize. These assembler directives, although written much like processor instructions, are commands to the assembler instead of the processor. They direct the assembler to perform specific tasks during the assembly process but have no meaning to the Z-80 processor. These assembler pseudo-ops are:

ORG nn — Sets address reference counter to the value nn.

EQU nn — Sets the value of a label to nn in the program: can occur only once for any label.

DEFL nn — Sets the value of a label to nn and can be repeated in the program with different values for the same label.

END — Signifies the end of the source program so that any following statements are ignored. If no END statement is found, a warning is produced. The END statement can specify a transfer address (i.e. END LABEL or END 6000H). The transfer address is used by the TRSDOS program execution to transfer control to the address specified in the END statement.

DEFB n — Defines the contents of a byte at the current reference counter to be "n".

DEFB 's' — Defines the content of one byte of memory to be the ASCII representation of character "s".

DEFW nn — Defines the contents of a 2-byte word to be "nn". The least significant byte is located at the current reference counter while the most significant byte is located at the reference counter plus one.

DEFS nn — Reserves "nn" bytes of memory starting at the current value of the reference counter.

DEFM 's' — Defines the contents of n bytes of memory to be the ASCII representation of string "s", where n is the length of "s" and must be in the range 0-63.

ASSEMBLER COMMANDS

The Galactic Software Editor Assembler supports only two assembler commands. Each command must start in column one of a source line, and must start with an asterisk (*). The assembler commands are:

*LIST OFF — Causes the assembler listing to be suspended, starting with the next line.

*LIST ON — Causes assembler listing to resume, starting with this line.

COMMANDS:

The GALACTIC Model II Editor Assembler can perform the following commands. These commands may be typed after the prompt symbol ">" which is displayed in reverse video for clarity. The prompt symbol appearance indicates the "command mode" of the Editor Assembler. The following list contains all command mode instructions recognized by the Editor Assembler with a brief description of each.

A Assemble source currently in the text buffer.

C Display the FILESPEC of the last source text file accessed either by Load or Write.

D Delete specified line(s).

E Edit a specified line of text.

F Find a specified string of characters.

G Globally change a string of characters (STRING1) to another string of characters (STRING2) throughout the text buffer.

H Provide hard copy output (line printer) of a specified range of text buffer lines.

I Insert source text line(s) at a specified line with a specified line number increment.

J Jump to a specified address.

L Load a source text file from disk.

M Move a block of text from one location to another.

N Renumber source text lines in the text buffer.

P Print a specified range of source text code currently in the text buffer.

Q Quit the Editor Assembler and return to TRSDOS.

R Replace lines currently in the text buffer.

S System command to execute any TRSDOS command from within the environment of the Editor Assembler.

T Type source text lines without line numbers to a line printer.

U Display the memory utilization - bytes used by the text, bytes available, and the first free address.

W Write the current text buffer to disk.

F1 Clear the CRT screen.

F2 Page forward the display 23 lines.

↑ Scroll up one source text line.

↓ Scroll down one source text line.

HOLD Performs a functional pause of any operation.

GALACTIC EDITOR ASSEMBLER COMMAND DETAILS

1.> ASSEMBLE (A)

Syntax: A (SWITCH(/SWITCH)...)

SWITCH may be any of the following five options:

NL No assembler listing is written to the screen.

NO No assembled object code is generated to disk or memory.

NS No symbol table is printed either to the screen or the line printer (if enabled).

LP Send assembler listing, error messages, and symbol table (if enabled) to a line printer.

WE Cause the assembly to wait when an error occurs. Depressing any key will continue assembly until another error is found. If you want to continue the assembly without stopping for additional errors, enter the character <C>

If you want to assemble an object program to either disk or memory, do not enter the switch parameter, "NO". The prompt

Object code to disk or memory (D,M)?

will be displayed. A response of "M" will assemble the object code to memory. You will not be permitted to overwrite any region below the end of the text buffer nor

will you be permitted to overwrite the symbol table stored in high memory. The error message,

Attempt to overwrite protected region - job aborted

will be displayed if your assembled program will violate these restrictions. Upon successful completion of the assembly to memory, the message,

Memory region loaded
XXXXX is the transfer address

will be displayed.

A response of 'D' assumes a disk object code file. The 'D' response will issue the query,

Enter filespec

Respond with the filespec that you want to use to save the assembled program file. The Editor Assembler will open the file if existing and output the message,

Replaced

or create the file if non-existant and output the message,

New file

Assembly will start and the program file will be written to disk.

2.> DISPLAY CURRENT SOURCE FILESPEC (C)

Syntax: C

This command will display the filespec used for the most recent Load or Write command. If neither Load nor Write were utilized, or the text buffer region was cleared, the message,

Filespec unknown

is displayed.

3.> DELETE (D)

Syntax: D (line1 (,line2))

This command is used to delete the line or lines specified from the source text buffer. The character <T> is used to indicate the top of the text buffer and the character <B> is

used to indicate the bottom of the text buffer.

Examples:

D 100,500	Delete lines 100 through 500 (inclusive) from the text buffer.
D T,B	Delete the entire text buffer.
D	Delete the current source text line. A period (.) may also be used to indicate the current line.
D 105	Delete the single line 105.

4.> EDIT (E)

Syntax: E (line)

This command permits the user to edit or modify any source text line. While in the edit mode, the line being editted is displayed in reverse video. The syntax and function of all edit subcommands are identical to those implemented in the DISK BASIC editor.

Edit Subcommands:

A	Abort and restart the line edit.
nC	Change n characters.
nD	Delete n characters.
E	End editting and enter the changes.
H	Delete the remainder of the line and insert the following string. The "H" command should not be used to delete an entire line of text. There M U S T always be at least one character on a line, or future use of that line will cause problems.
I	Insert string.
nKx	Kill all characters up to the nth occurrence of x.
L	Print the rest of the line and go back to the starting position of the line.
Q	Quit and ignore all editting.
nSx	Search for the nth occurrence of x.

BACKSPACE Move edit pointer back one space.

ESCAPE Escape from any edit mode subcommand.

ENTER Enter the line in its presently editted form and exit the edit mode.

5.> FIND (F)

Syntax: F (string)

The edit buffer is searched starting at the current line+1 for the first occurrence of "string". If no string is specified, the search is the same as that of the last Find command in which a string was specified (provided a Global command was not previously specified). If the search string is found, the line containing it is displayed and period (.) is updated to the displayed line. If the string is not found, the message,

String not found

is displayed and period (.) remains unchanged. A "PT" command can be used to position the line pointer to the top of the text buffer prior to use of the Find command.

6.> GLOBAL (G)

Syntax: G /string1/string2/

A string of characters can be changed throughout the text buffer by one easy command. The GLOBAL CHANGE command will change the appearances of STRING1 to the sequence STRING2. No changes will be performed on the first line of the text buffer. Also, only the first appearance of STRING1 in each line that STRING1 appears will be altered.

The first non-blank character becomes the string delimiter (the slash character is shown above; any character is permitted). Null strings are not permitted (i.e. the string must contain at least one character).

It is not necessary that STRING2 be the same length as STRING1. It can be of lesser, equal, or greater length; however, no string can exceed 16 characters in length. If a change would result in a line exceeding the maximum line length (128), the change will not be performed on that line and the message,

FIELD OVERFLOW

will be issued. The search for STRING1 continues for the remaining lines.

A line which contains STRING1 will be displayed as it exists both before and after the change. The <HOLD> key may be used to pause the output. Use of the <BREAK> key will stop further changing.

Example:

```
G /MODIFY/ALTER/
```

7.> HARDCOPY (H)

Syntax: H (line1 (,line2))

This command will print a line or a group of lines to a line printer. If a properly paged display is desired, it is suggested that you set the forms control by issuing the Editor Assembler's "System" command as in:

```
S FORMS (P=xx,L=xx,)
```

Examples:

H T,B	Print the entire text buffer.
H 100,500	Print lines 100 through 500 inclusive.
H.	Print the single line pointed to by period (.).
H	Print the 23 lines starting with the current line.

8.> INSERT (I)

Syntax: I line# (,inc)

The Insert command is used to insert or add text lines in the buffer. All lines of source text are entered with the use of the Insert command. After using the Insert command to specify were you wish to place new lines, the EDITOR will generate the designated line number and allow the inserting of that numbered text line. After entering the first text line the editor will generate the next line number higher, as specified by your increment selection. Incremental line numbers will continue to be generated as long as there is room between lines or room left in the text buffer.

The <BREAK> key will allow you to leave the insert mode at any time.
If a desired increment is not specified the last specified increment is assumed. Period (.) may be used for "line#" to indicate the current line.

9.> LOAD (L)

Syntax: L (filespec)

The Load command will read the file denoted by the FILESPEC into the text buffer. The text file will be concatenated to any text already in the text buffer. FILESPEC is explained in your TRSDOS (tm) user manual under the "TRSDOS" section entitled "file specification". It is composed of a FILENAME, optional EXTension, optional PASSWORD, optional DRIVE reference, and optional diskette name as in

FILENAME/EXT.PASSWORD:D(DISKETTE NAME)

(ex. YOURPROG/ASM:1). If you do not enter the FILESPEC, Editor Assembler will use the filespec entered for the last Load or Write command provided there is text already in the text buffer. If the text buffer is empty and you do not enter a filespec with the Load command, Editor Assembler will prompt you for the filespec.

10.> MOVE (M)

Syntax: M line1, line2, line3

This command is used to move a block of lines from one location in the text buffer to another. A large quantity of text lines can be moved to a different position in one easy operation. In the command syntax, "line1" and "line2" are the beginning and ending line numbers of the text block to be moved. "Line1" and "line2" are permitted to reference the same line number if only one line is to be moved. "Line3" is the line number of the line that the text block will follow after the move. The line number references must be offset by commas (,). If any of the entered line numbers are non-existant, the message,

No such line

will be issued.

"Line3" is not permitted to equal "line1" or "line2". "Line3" is not permitted to be a line interior to the range "line1" through "line2". The message,

Command parameter(s) incorrect

will be issued if your input violates any of these conditions.

The text to be moved is stored temporarily in the spare text region. If this region is not large enough to store the block, the message,

Text buffer full

will be issued. Try moving the block in segments.

Upon completion of the move, all lines in the text buffer will be renumbered starting from ten (10) and using the line increment currently in effect. Renumbering is absolutely essential to perform proper operation of Editor Assembler commands.

Example:

You desire to move the block of text starting at line 500 and ending at line 900 to follow line 1510. Issue the command,

M 500,900,1510.

11.> RENUMBER (N)

Syntax: N (line(,inc))

The "N" command is used to renumber the lines in the text buffer. The first line in the buffer is assigned the number specified as "line". If "line" is not specified, it defaults to 00100. The remaining lines in the buffer are renumbered according to the increment (inc) or the previous increment in a RENUMBER, REPLACE, or INSERT command if the increment was not specified. Period (.) points to the same line as it did before the NUMBER command was used, but the actual number of this line may be changed.

Examples:

N	Renumbers from 100 with the previous increment.
N5	Renumbers from 5 with the previous increment.
N10,5	Renumbers from 10 in steps of 5.

12.> PRINT (P)

Syntax: P (line1(,line2))

The PRINT command will display a line or a group of lines on the monitor screen. Period (.) is updated to point to the last line printed.

Examples:

P T,B	Displays all lines in the text buffer.
P 100,500	Displays lines 100 through 500 inclusive.
P .	Displays the current line only.
P	Displays 23 lines starting with the current line. The PRINT command operates in a screen scroll mode. If you want to "page" the screen, use the "F2" command.

13.> QUIT (Q)

Syntax: Q

The QUIT command is used to exit the Editor Assembler and perform a proper return to TRSDOS. By using command "Q", the <BREAK> key will be restored to TRSDOS.

14.> REPLACE (R)

Syntax: R (line<inc))

The REPLACE command only replaces one line and enters INSERT mode. If "line" exists, it is deleted then inserted. If line doesn't exist, it is inserted as with the INSERT command. If "inc" is not specified, the last increment specified by an INSERT, REPLACE, or RENUMBER command is used. Period (.) is always updated to the current line.

Examples:

R	Replace the current line.
R 100,10	Start replacing lines beginning at line 100 and incrementing with 10.
R 100	Start replacing at line 100 using the last specified increment.

15.> SYSTEM (S)

Syntax: S ANY-TRSDOS-COMMAND (PARAMETERS)

The SYSTEM command is used to interface with TRSDOS while in the environment of the Editor Assembler. Any TRSDOS command can be accessed. It is recommended that you not attempt to access the TRSDOS "COPY" nor "BACKUP" commands due to the possibility of overwriting the Editor Assembler. IT IS

IMPORTANT TO NEVER DEPRESS THE <BREAK> KEY DURING A SYSTEM TRSDOS COMMAND. To break any TRSDOS command, use the <ESCAPE> key.

Examples:

S DIR	List the diskette directory.
S FORMS (P=51,L=42)	Set printer parameters.
S LIST filespec	List the contents of a file.
S PURGE :d	Delete files from drive "d".

16.> TYPE (T)

Syntax: T (line1(,line2))

The TYPE command prints a line or group of lines onto the Line Printer. Period (.) is updated to point to the last line printed. This command is much like the HARD COPY command, only no line numbers are printed. Only the source text is printed.

17.> MEMORY USAGE (U)

Syntax: U

This command will display the number of bytes of text buffer in use, the number of bytes spare and the first address available for assembly to memory.

This command is useful to ascertain requirements for storing the text buffer to disk. Note that a disk file, which is written in ASCII, will contain an additional four (4) bytes per text line. The 4 bytes arise from the difference in storage formats of text in memory versus text in an ASCII file.

It also is useful when assembling to memory. Since the Assembler will not permit you to overwrite it or the text buffer, you will have to "ORG" your program in the free text buffer area. The first available address is output by this command.

An example of output is:

```
12288 bytes in use
27934 bytes spare
37292 (91AC) is the first free address
```

18.> WRITE (W)

Syntax: W (filespec)

This command will write the text buffer to the file denoted by FILESPEC. If no FILESPEC is entered, the filespec referenced by the previous Load or Write command will be used unless the text buffer is empty. If a FILESPEC is unavailable for use, you will be prompted for it.

If the file denoted by FILESPEC is non-existant, a file will be created and the message,

New File

will be issued.

If the file denoted by FILESPEC is an existing file, it will be replaced by the write operation and the message,

Replaced

will be issued. YOU WILL NOT BE GIVEN AN OPPORTUNITY TO CANCEL A WRITE REQUEST ON AN EXISTING FILE. Know what you are doing.

19.> SCROLL UP (↑)

The SCROLL UP command displays the line preceding the current line and updates period (.) to point to the line displayed. If the current line is the first line in the text buffer, it is displayed and period (.) remains unchanged. SCROLL UP is an immediate command and must be the first character of a command line in order to be interpreted.

20.> SCROLL DOWN (↓)

The SCROLL DOWN command displays the line following the current line and updates period (.) to point to the line displayed. If the current line is the last line in the text buffer, the last line is displayed and period (.) remains unchanged. SCROLL DOWN is an immediate command and must be the first character of a command line to be interpreted.

21.> CLEAR SCREEN (F1)

The <F1> key is used to perform a functional clear screen (similar to "S CLS").

22.> PAGE FORWARD (F2)

The <F2> key is used to advance the display by 23 lines. This command is similar to the PRINT command except that the monitor screen is cleared before displaying the 23 lines.

23.> PAUSE (HOLD)

The <HOLD> key is used to pause the computer during a display during any assembly or Editor Assembler printing. When a pause is sensed, depression of any key except <HOLD>, <SHIFT>, or <CTRL> will continue the operation paused.

Error Messages

The Galactic Software Model II Editor Assembler recognizes three types of errors. These are:

1.> Command errors - The error message is displayed and control is returned to command mode.

2.> TRSDOS errors - The error message (or error number) is displayed and control is returned to command mode.

3.> Assembler errors - These three types of errors may occur while executing an Assemble command.

   a. Terminal - Assembly is terminated and control is returned to command mode.

   b. Fatal - Processing of the line containing the error is immediately stopped and no object code is generated for that line. Assembly proceeds with the next line.

   c. Warning - The error message is displayed and assembly of the line containing the warning continues. The resulting object code may not be what the programmer intended.

Following is a list of all errors and an explanation of each.

COMMAND ERRORS

1.> Buffer full

There is no more room in the text buffer for adding text.

2.> Command parameter(s) incorrect

Any command line not entered according to the syntax appropriate for that command will generate this error message.

3.> Illegal command

The first character of the command line entered does not specify a valid Editor Assembler command.

4.> Invalid source file

A Load filespec command was issued where the file identified by filespec is not a valid Editor Assembler source file.

5.> Line number too large

Renumbering with the specified starting line number and increment would cause line(s) to be assigned numbers greater than 65529. The renumbering is not performed. This message would also be displayed if you attempted to INSERT a line with a line number exceeding 65529.

6.> No room between lines

The next line number to be generated by INSERT or REPLACE would be greater than or equal to the line number of the next line of text in the edit buffer. The increment must be decreased or the lines in the buffer renumbered.

7. No such line

A line specified by a command does not exist. The command is not performed.

8. No text in buffer

A command requiring text in the buffer was issued when the text buffer was empty. The commands Load, Insert, Quit, System, Jump, <F1>, and Display current filespec can be executed when the text buffer is empty. All other commands require at least one line of text to be in the buffer.

9.> String not found

The string being searched for by the Find command could not be found between the current line and the end of the text buffer. This message will also be displayed at the completion of a Global command.

TRSDOS ERRORS

1.> Disk drive not ready

This message will be displayed after a Load, Write, or Assemble to disk command is executed if either the specified drive is not ready (no diskette, diskette in backwards, drive door not closed, etc.) or the specified drive does not exist.

2. Disk is write protected

A Write command was issued with a filespec designating a drive loaded with a diskette that is protected from a write operation.

3.> Unusable file specification

A Load, Write, or Assemble to disk command was executed with a filespec that did not conform to TRSDOS specifications. It is also possible that the drive specified was not in the range 0-3.

4.> Filespec not in directory

The filespec entered for execution of a Load, Write, or Assemble to disk command could not be located in the drive directory.

5.> Access denied (password incorrect or missing)

An attempt was made to access a TRSDOS file. Either the password entered was incorrect or no password was entered for a password protected file.

6.> Too many files in the directory

The directory space is full on the designated diskette.

7.> No disk space available

A Write or Assemble to disk command was executed which resulted in a file using the available disk space prior to completion. The operation terminates and the file is closed.

NOTE: Under TRSDOS 1.2, a TRSDOS system error causes unpredictable behavior of the system when a diskette becomes full. It is strongly recommended that you pay close attention to the amount of available space on a diskette by issuing the System commands DIR or FREE. As a diskette's available free space diminishes, you may want to avoid creating any new files on it and continue your operation with a diskette that has sufficient free space. File storage requirements for the text buffer may be ascertained using the Editor Assembler's USAGE command.

8.> Hardware failure during I/O

This message is displayed when a disk operation is unsuccessful and TRSDOS returns error code 41 or 49.

9.> Printer is not ready for use

Any output sent to the line printer when the printer is unavailable will generate this error. The printer may be turned off, out of paper, in trouble, or not plugged into the system.

10.> *TRSDOS error code # <xxxxx>*

Any other TRSDOS error not specifically identified above will be displayed in this form. If you want the full TRSDOS explanation, issue the command:

S ERROR xxxxx

TERMINAL ERRORS

1. Attempt to overwrite protected region (job aborted)

During an assembly to memory, a block of code was assembled that would load into a memory region other than the spare text buffer area. Your program will not be permitted to load to an address below the end of the text buffer or above the symbol table. Use the Usage command to locate the first available memory address.

2.> Symbol table overflow

There is not enough memory for the assembler's symbol table. You have three options:

a. Remove comment lines and/or comments following Z-80 code operands. This may free up enough space to perform the assembly.

b. TRSDOS locks out space above X'F300' for user use. This space is utilized by the DEBUG program and Serial port drivers. If your operation will not use the either serial port and DEBUG is to remain OFF, then this space can be recaptured. Do the following:

   1. Save your current text buffer.

   2. Return to TRSDOS via the Quit command.

   3. Enter the TRSDOS command, "DEBUG ON"

   4. Load the Editor Assembler program using the TRSDOS "LOAD" command.

   5. Enter the TRSDOS command, DEBUG

   6. Using the DEBUG command "R", change register pair "DE" to X'FFFF'.

   7. Using DEBUG's Jump command, jump to X'3403'. You will enter the Editor Assembler with its top-of-memory now set to X'FFFF'.

8. Enter the Editor Assembler command, S DEBUG OFF

9. Load your previously saved text buffer and attempt to assemble it.

c. Split your source program into two or more programs that can be assembled separately.

FATAL ERRORS

1.> Bad label

The character string found in the label field of the source statement does not match the criteria specified under ASSEMBLY LANGUAGE - LABELS.

2.> Expression error

The operand field contains an ill-formed expression.

3.> Illegal addressing mode

The operand field does not specify an addressing mode which is legal with the specified opcode.

4.> Illegal opcode

The character string found in the opcode field of the source statement is not a recognized instruction mnemonic or assembler pseudo-op.

5.> Missing information

Information vital to the correct assembly of the source line was not provided. The opcode is missing or the operands are not completely specified.

WARNINGS

1.> Branch out of range

The destination of a relative jump instruction (JR or DJNZ) is not within the proper range for that instruction. The instruction is assembled as a branch to itself by forcing the offset to hex X'FE'.

2.> Field overflow

A number or expression result specified in the operand field is too large for the specified instruction operand. The result is truncated to the largest allowable number of bits.

3.> Multiply defined symbol

The operand field contains a reference to the symbol which has been multiply defined. The first definition of the symbol is used to assemble the line.

4.> Multiple definition

The source line is attempting to illegally redefine a symbol. The original definition of the symbol is retained. Symbols may only be redefined by the DEFL pseudo-op and only if they were originally defined by DEFL.

5.> No end statement

The program END statement is missing

6.> Undefined symbol

The operand field contains a reference to a symbol which has not been defined. A value of zero is used for the undefined symbol.

Object file format

The disk file object code format consists of the following structure:

1.> A file header string consisting of:

a. The first byte in the file is a hex byte X'05' which indicates the header field of an object file.

b. The second byte is the header length byte and indicates the length of the header following.

c. The length byte is followed by the FILENAME and EXTENSION that was specified when the file was last written to.

d. The filename field is immediately followed by the entire DATE string as recovered by the TRSDOS SVC DATE - Function Code 45. By LISTing the first sector of the file, you can determine when the file was last written by examining this header. A TRSDOS RENAME command will not change the filename stored in the header.

2.> Multiple blocks of object code depending on the length of your assembled program are placed next. The object code blocks have the following code format:

a. A beginning byte of X'01' which indicates the start-of-block

b. A 1-byte length indicating the length of the code block following, including the block load address (block length of 256 will show X'02'). The Editor Assembler writes 128-byte blocks (length = X'82').

c. The block length byte is followed by the 2-byte block load address which is the address that will be loaded with the first byte of the block.

d. Finally the block immediately follows for as many bytes as two less than the block length.

3. Steps 2a, 2b, 2c, and 2d are repeated for as many blocks as are in the file. An X'02' is then written to indicate the end of the program code and the start of the entry point or

transfer address. An X'02' is written to indicate the length of the entry point address. This is then followed by the 2-byte entry point or transfer address generated from the label or constant in the operand field of the assembler source END statement.

Source file format

The source code file format is as follows:

1.> A header record as described under "Object file format" is also used for source files with the exception that the first byte is a hex X'53' to identify the file as source.

2.> Text lines are written in ASCII each composed of a 5-character line number (bit 7 is not set), a space, the text line, ending with an <ENTER> (X'0D').

3.> The file end is indicated by an end-of-file mark of X'1A' which would be in the first character position of a text line.

4.> Model I source text files follow a different format (header start byte of X'D3', followed by a 6-character filename with text line numbers having bit 7 set). In spite of this difference, source files generated on a Model I machine using the MISOSYS DISK*MODified EDTASM and uploaded to a Model II machine can be loaded into this Editor Assembler.

LINKAGE TO DEBUGGING

In order to facilitate the debugging of user generated programs, a number of features have been built into this Editor Assembler. It provides the option of assembling source code directly to memory. It provides a command to transfer control to a user-supplied address (via the JUMP command). It provides for the access of DEBUG through the System command. Other subtle enhancements have been implemented.

A re-entry address to the Editor Assembler has been provided. If at any time during the debugging phase, you want to return to the Editor Assembler without reinitializing it (which would have deleted the entire text buffer), and are under the control of DEBUG, issue a DEBUG Jump command to X'3400'. A return to the Editor Assembler will be performed and it will take over the supervision of the <BREAK> key without reinitializing the pointers to the text buffer.

When you exit from the Editor Assembler by means of the Jump command, address X'3400' is pushed onto the stack just prior to executing the jump. If your program maintains stack integrity, an easy return to the Editor Assembler is achieved by means of a "RET" instruction. An example of this procedure is as follows:

```
BEGIN    LD    (SPSAV),SP    ;SAVE THE POINTER
           .
           .
           .
         USER PROGRAM
           .
           .
           .
EXIT     LD     SP,(SPSAV)   ;RESTORE STACK
         RET                 ;& RETURN TO EDAS
```

Z-80 INDEX TO INSTRUCTION SET

Execution time (E.T.) for each instruction is given in microseconds for an assumed 4 MHZ clock. Total machine cycles (M) are indicated with total clock periods (T states). Also indicated are the number of T states for each M cycle. For example:

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T. 1.75

indicates that the instruction consists of two machine cycles. The first cycle contains four clock periods (T states). The second cycle contains three clock periods for a total of seven clock periods or T states. The instruction will execute in 1.75 microseconds.

Register format is shown for each instruction with the most significant bit to the left and the least significant bit to the right.

OPERAND NOTATION

The following notation is used in the assembly language:

1. "r" specifies any one of the following registers:

   A, B, C, D, E, H, & L

2. "(HL)" specifies the contents of memory at the location addressed by the contents of the register pair HL.

3. "n" specifies a one-byte expression in the range 0 to 255. "nn" specifies a two-byte expression in the range 0 to 65535.

4. "d" specifies a one byte expression in the range -128 to +127.

5. "(nn)" specifies the contents of memory at the location addressed by the two-byte expression "nn".

6. "b" specifies an expression in the range 0 to 7.

7. "e" specifies a one-byte expression in the range -126 to 129.

8. "cc" specifies the state of the Flags for conditional JR and JP instructions.

9. "qq" specifies any one of the following register pairs:

   BC, DE, HL, & AF

10. "ss" specifies any one of the following register pairs:

    BC, DE, HL, & SP

11. "pp" specifies any one of the following register pairs:

    BC, DE, IX, & SP

12. "rr" specifies any one of the following register pairs:

    BC, DE, IY, & SP

13. "'" specifies any one of the following:

    r, n, (HL), (IX+d), & (IY+d)

14. "dd" specifies any one of the following register pairs:

    BC, DE, HL, & SP

15. "m" specifies any of the following:

    r, (HL), (IX+d), & (IY+d)