TRS-80™ Model III

Disk System Owner's Manual

Mini-Disk Operation
TRSDOS™ Disk Operating System
Disk BASIC Programming Language

Radio Shack®
The biggest name in little computers™

CUSTOM MANUFACTURED IN THE USA BY RADIO SHACK A DIVISION OF TANDY CORPORATION

The FCC Wants You to Know . . .

This equipment generates and uses radio frequency energy. If not installed and used properly, that is, in strict accordance with the manufacturer's instructions, it may cause interference to radio and television reception.

It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation.

If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, you should consult the dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet prepared by the Federal Communications Commission helpful: *How to Identify and Resolve Radio-TV Interference Problems.*

This booklet is available from the US Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

Warning

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.

CHANGE OF ADDRESS

NOTE: If you move, please fill out this card and return it so that you may continue to receive information regarding this program.

Purchase Date ________	13050181
	Version/Date
	Cat. No. 26-1063

NEW ADDRESS:	OLD ADDRESS:
Name ________	Name ________
Company ________	Company ________
Address ________	Address ________
City ________	City ________
State ________ Zip ________	State ________ Zip ________

INSTRUCTIONS FOR USE

1. Register one software package per card only.
2. Complete the Software Registration portion of this form and mail it immediately. The Catalog No. may be found by examining the upper-right corner of your diskette.
3. For convenience a change of address card has been included. Copy all information from the Registration Card onto it prior to sending the Registration Card.

Model III Users

XFERSYS UTILITY ON TRSDOS 1.3

The Model III diskette in this package contains a NEW version of TRSDOS which is not compatible with OLD versions of TRSDOS.

OLD TRSDOS diskettes to be used under the NEW TRSDOS MUST be XFERSYSed before use. Once XFERSYSed, an OLD TRSDOS diskette becomes a NEW TRSDOS diskette and should not be used with OLD TRSDOS again. If you started with an OLD system or data diskette, the XFERSYSed diskette will be a NEW system or data diskette respectively.

OLD diskettes used under NEW TRSDOS without XFERSYSing, may cause extraneous information to be read at the end of files, giving a false End-of-File (EOF) indication. Some programs will not function properly under these conditions.

NEW diskettes used under OLD TRSDOS, may not access all data and/or NEW programs may not run correctly.

If you need to use the XFERSYS utility, see the TRSDOS section of your TRS-80 Model III Disk System Owner's Manual.

Note: When changing from one TRSDOS to the other, you MUST press the RESET button each time the diskette in Drive 0 is changed. You may also XFERSYS onto a NEW data disk. If this is done, all system files of the system diskette will be moved onto the data diskette.

RADIO SHACK APPLICATION PROGRAMS WHICH WERE DELIVERED ON AN OLD TRSDOS DISKETTE SHOULD NOT BE XFERSYSed.

OLD:	TRSDOS 1.1 and 1.2.
NEW:	TRSDOS 1.3
file:	A collection of information stored as one named unit in the directory.
program:	A file which causes the computer to perform a function.
data:	Information contained in a file which is used by a program.
system diskette:	A diskette containing TRSDOS. When this diskette is placed in Drive 0 and the RESET button is pressed, TRSDOS will begin to run.
data diskette:	A diskette which does not contain TRSDOS. If this diskette is placed in Drive 0 and RESET is pressed, the screen will clear and "Not a SYSTEM Disk" will be displayed.
XFERSYS:	A program contained on the TRSDOS 1.3 diskette.

Tips On Loading Disk BASIC

There are several ways, other than those described in the manual, to start up Disk BASIC.

BASIC *program* -F:*files* -M: *address*

program is a TRSDOS file specification for a Disk BASIC program. After start-up, Disk BASIC will run the program. This is optional.

-F:_files_ tells Disk BASIC the maximum number of files that may be open at one time. _files_ is a number from 0 to 15. This is optional; if omitted, 3 is used. If Variable length files are needed, you must include the suffix v after _files;_ otherwise, _files_ will be Fixed length.

-M:_address_ tells Disk BASIC not to use memory above the specified address. This is optional; if omitted, Disk BASIC uses all memory up to Top.

If all options are omitted, Disk BASIC will prompt with `HOW MANY FILES?` and `MEMORY?`

The options allow you to specify any or all of the following:

- A program to run after Disk BASIC is started.
- The maximum number of data files that may be Open at one time. The larger the number of files, the less area available for storing and executing your programs. (Note: Each Fixed length file takes up 360 bytes and each Variable length file takes up 616 bytes of memory.)
- The highest address to be used by Disk BASIC during program execution. Omit this unless you are going to call machine-language subroutines.

Examples

Under TRSDOS READY, if you type:

`BASIC` (ENTER), Disk BASIC will enter the command mode once you answer the files and memory prompts.

`BASIC -F:1` (ENTER), Disk BASIC will Open one fixed length file and protect no memory.

`BASIC -M:32000` (ENTER), Disk BASIC will Open three fixed length files and will use memory no higher than 32000.

`BASIC PAYROLL -F:3V` (ENTER), Disk BASIC will start-up, then load and run the BASIC program called PAYROLL; three variable length files can be Opened, and BASIC can use all available memory.

Note: If you PATCH Disk BASIC to prompt you for cassette speed when you enter Disk BASIC, you will still be prompted with `CASS?` no matter which option you use.

Important Note for Model III TRSDOS 1.3 Users!

The 1.3 version of TRSDOS will return with `ERROR 31 (PROGRAM NOT FOUND)` when you attempt to use the ROUTE command.

Thank-You!

Radio Shack®

T/C A Division of Tandy Corporation

8759109

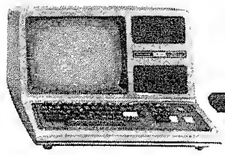
TRS-80™ Model III

Disk System Owner's Manual

Radio Shack®

A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

10 9 8 7 6 5 4 3 2 1



To Our Customers

Congratulations on your purchase of the Model III Disk System. We think it's a valuable tool which will save you work as well as give you hours of enjoyment (or maybe both at once). You'll have all the power of the non-disk Model III, plus the following features:

- Your Computer can now be controlled by TRSDOS™, the powerful TRS-80 Disk Operating System. TRSDOS is included on a diskette with the Disk System.
- Using TRSDOS, you can run a wide variety of programs, such as the Disk BASIC **interpreter** included on the TRSDOS diskette.
- Each **"system" diskette** has approximately 126,720 bytes of storage available for your own programs and data; each **"data" diskette** has 178,944 bytes available.
- You can load and save data at the approximate rate of 250,000 bits per second.
- Your system can continue to grow in power and convenience. When Radio Shack issues improvements and enhancements to the system programs, you can "install" them simply by obtaining a new release of the TRSDOS diskette.

Model III Manuals

Publications related to the use of the Model III Disk System:

1. *Model III Disk System Owner's Manual* (this manual). We'll call it the "Disk Manual" for short.
2. *Model III Disk System Quick Reference Card.*
3. *Model III Operation and BASIC Language Reference Manual*, the "Model III Manual" for short.
4. *Model III BASIC Quick Reference Card.*

For Disk Operation:

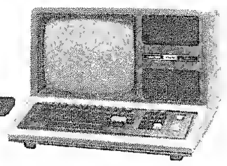
This Disk Manual supplements the Model III Manual. Use the *Disk Manual* as the primary source of information; we'll tell you when to refer to the non-disk Model III Manual.

For Non-Disk Operation:

To use the Computer as a *non-disk* system, all you need is the Model III Manual.

For Programming Information:

The Model III Manual contains most of the programming information, except that which pertains to disk input/output. In this manual, we will assume that you



are familiar with the BASIC programming definitions and details given in the Model III Manual.

About This Manual

The Model III Disk System is intended for use by novices as well as experienced computer operators and programmers. In designing and writing this Disk Manual, we've tried to define and satisfy the needs of both groups:

- Novices who might prefer a sequential presentation which emphasizes procedures and explains the purpose of various features.
- Experienced users who might prefer a more analytical presentation which makes it easy to find specific information.

In this manual, you'll find information that should satisfy your needs, whichever group you might belong to.

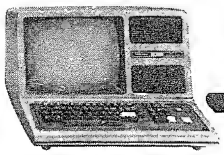
The "Sample Sessions" are especially geared for novices, while the Technical Information chapters are for the more experienced users.

Keep in mind, however, that it isn't necessary to read the entire manual to operate the Disk System. If you are only interested in Disk BASIC, for example, read the Operation section of this book and then turn directly to the Disk BASIC section. You can then go back to the TRSDOS section when you need to.

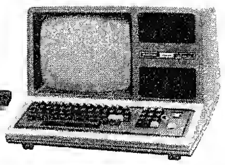
Special Terms

Even in the non-technical sections of this manual, we've had to use numerous special terms. Rather than scattering and repeating definitions throughout the book, we have used the following convention which we hope you'll find helpful.

Special terms which are fully defined in another part of the manual are printed in **boldface**. Look up the word or phrase in the Index; this will tell you where the word is fully defined.

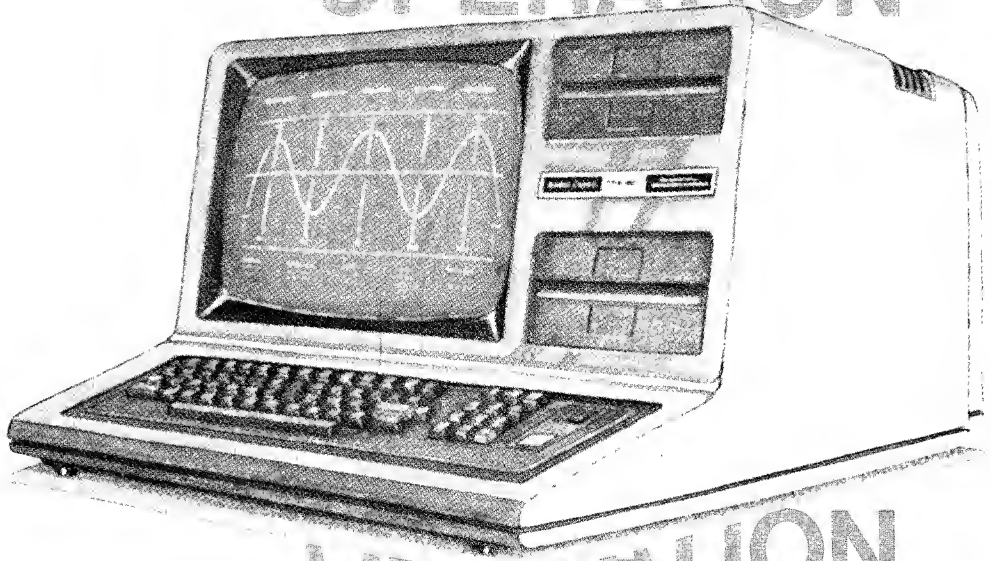


Contents

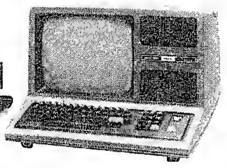


Disk BASIC

OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION
OPERATION



ATION
OPERATION
OPERATION
OPERATION



Installation

First set up the Computer according to the instructions in the Model III Manual.

If you have a one- or two-drive system, installation is now complete. The built-in drives should be ready for use.

If you have a three- or four-drive system, you need to connect the external drives.

External Disk Drives

The two external drives are *not* interchangeable. They have different Radio Shack Catalog Numbers and a few internal differences.

	System Name	**Catalog Number**
First External Drive Purchased (Includes Cable)	''Drive 2/3''	26-1164
Second External Drive Purchased	''Drive 2''	26-1161

The 26-1164 drive may be used as Drive 2 or 3, depending on the number of drives in the system. In a three-drive system, it is always Drive 2 (the last drive). In a four-drive system, it is always Drive 3 (again, the last drive).

The 26-1161 drive may only be used in a four-drive system, in which it must be Drive 2.

1. Locate the flat ''ribbon'' cable that was included with the 26-1164 drive. Notice that it has a single plug on one end, and two plugs clustered at the other end. See **Figure 1** for plug labels.
2. Connect the solitary ''Computer'' plug to the Disk Expansion Jack on the bottom rear of the Computer. See **Figure 2**.
3. Now refer to **Figure 3**. Connect the external drive(s) to the other end of the cable, as follows:

3-A. If you have one external drive (26-1164):
Connect it to the ''Drive 2'' plug near the middle of the ribbon cable.

3-B. If you have two external drives (26-1164 and 26-1161):
Connect the 26-1164 to the ''Drive 3'' plug on the end of the cable.
Connect the 26-1161 to the ''Drive 2'' plug near the middle of the cable.

4. Plug the external drive(s) into an appropriate source of AC power. Power requirements are specified on the unit and in the specifications given in this manual.

You are now ready to start the Disk System.

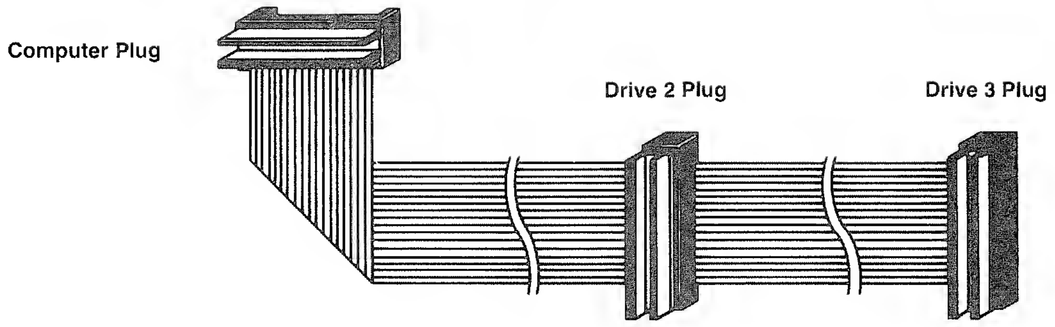


Figure 1. External Disk Cable with Plugs Labeled.

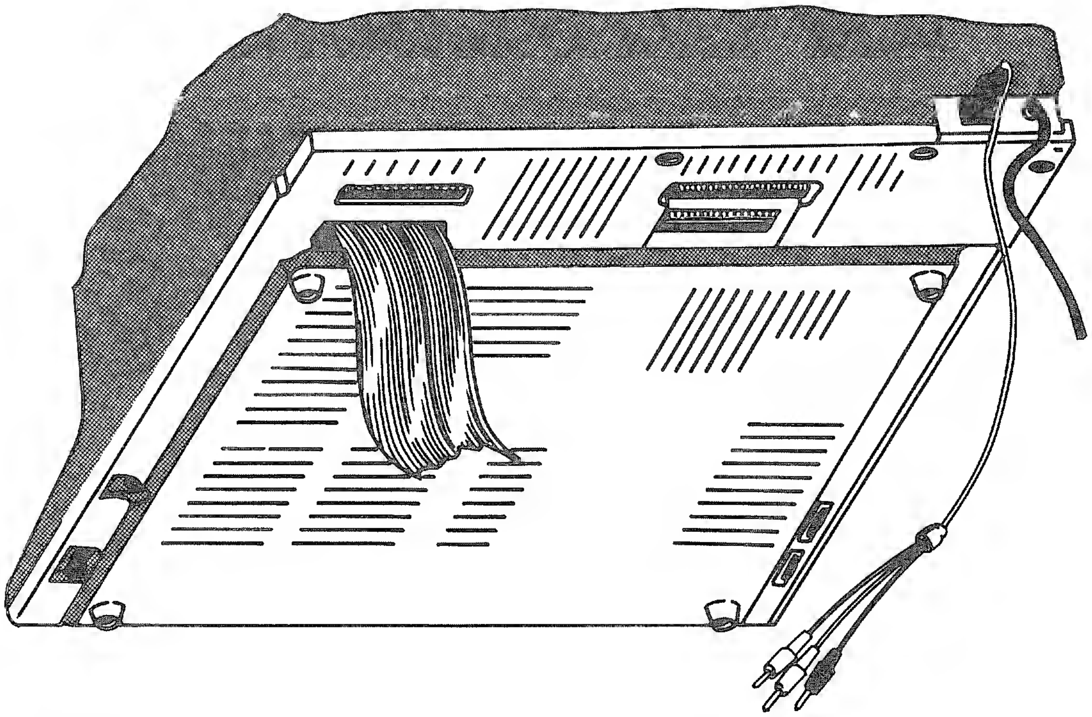
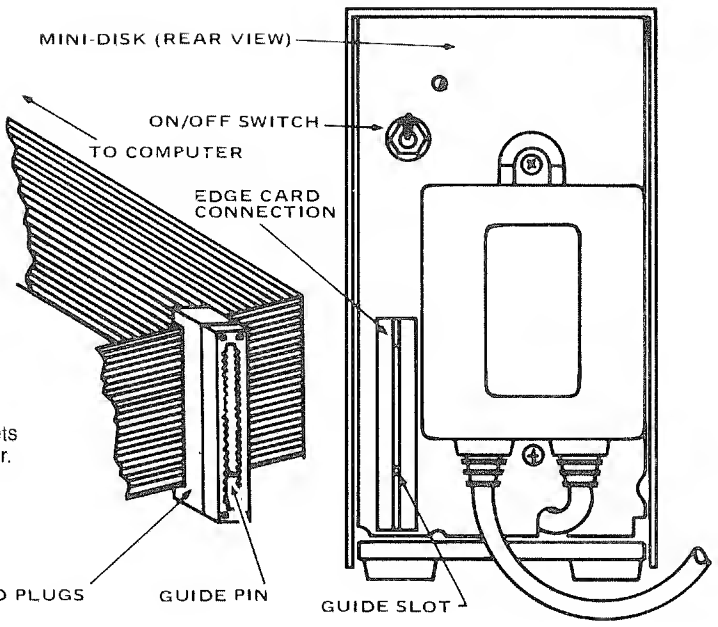


Figure 2. Connection of the External Disk Cable to the Model III.



Attach the plug so the cable exits toward the *rear* of the Computer.

Figure 3. Connection of external disk drives.

Operation

First, take a few minutes to become familiar with the various elements of your Disk System. Refer to **Figures 4** and **5**. This is very important. If you try to use the Computer without having a little background information, you could damage a diskette.

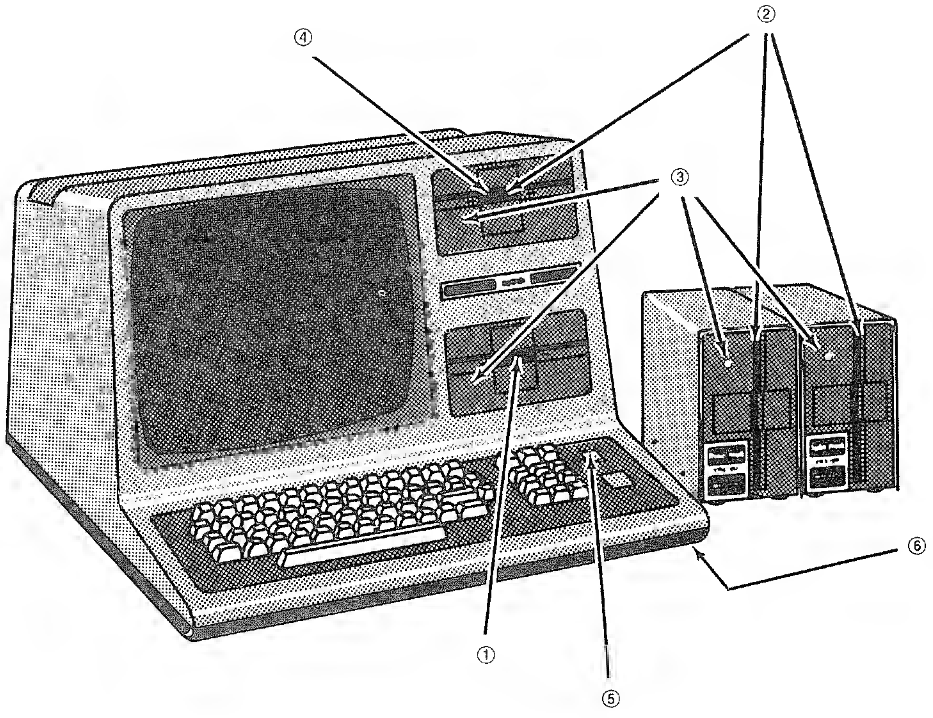


Figure 4. The Model III Disk System with External Drives (optional/extra).

① **Drive 0.** The TRSDOS "system diskette" goes in this drive.

② **Drives 1, 2, and 3.** These drives may contain "data diskettes." Data diskettes are described briefly in this chapter.

③ **Drive Select LED.** When a drive is being accessed, its LED lights up.

④ **Drive Door.** To insert or remove a diskette, open this door. Never remove a diskette while the LED is lit, or while the diskette contains open files.

⑤ **Reset Button.** When you press this button, the Computer will attempt to load the operating system software from Drive 0. The TRSDOS diskette should be in Drive 0 when you press this button.

⑥ **Power Switch.** All drives should be *empty* when you turn the Computer on or off. Otherwise, the information on the diskettes could be destroyed.

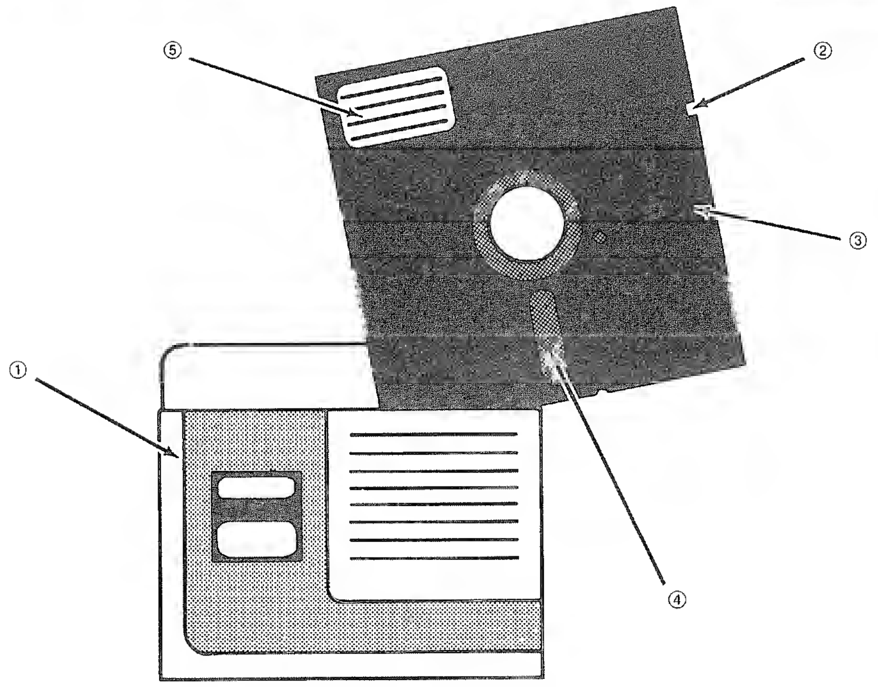


Figure 5. A Diskette. (Catalog Number 26-305, 26-405, or 26-406)

① **Storage Envelope.** While a diskette is not in use, keep it here.

② **Write Protect Notch.** When this is covered, the disk-drives cannot write (change information) on the diskette. *Do not pinch* the tab into the notch when you apply it. If the tab becomes indented, the disk drive may not sense that the disk is write-protected. Leave the notch uncovered if you want to save or change information on the diskette.

③ **Jacket.** The diskette is permanently sealed inside this protective jacket. Do not attempt to remove it.

④ **Read/Write Window.** The disk drive accesses the diskette surface through this window. Don't touch the diskette surface.

⑤ **Label.** To write on this label, use only a felt-tipped pen. Any other writing implement might damage the diskette.

Diskettes

In general, handle diskettes carefully, using the same precautions you use with tape cassettes and high-fidelity records. A small indentation, dust particle, or scratch can render all or part of a diskette unreadable—*permanently.*

- Keep the diskette in its storage envelope whenever it is not in one of the drives.
- Do not place a diskette in the drive while you are turning the system on or off.
- Keep diskettes away from magnetic fields (transformers, AC motors, magnets, TVs, radios, etc.). Strong magnetic fields will erase data stored on a diskette.

- Handle diskettes by the jacket only. Do not touch any of the exposed surfaces. *Don't try to wipe or clean the diskette surface;* it scratches easily.
- Keep diskettes out of direct sunlight and away from heat.
- Avoid contamination of diskettes with cigarette ashes, dust or other particles.
- Do not write directly on the diskette jacket with a hard point device such as a ball point pen or lead pencil; use a felt tip pen only.
- Store diskettes in a vertical file folder on a shelf where they are protected from pressure to their sides (just as phono records are stored).
- In very dusty environments, you may need to provide filtered air to the computer room.

Tips on Labeling Diskettes

Each diskette has a permanent label on its jacket. This label is for "vital statistics" that will never change. For example, to help keep track of diskettes, it's a good idea to assign a unique number to each diskette. Write such a number on the permanent label. You might also put your name on the diskette, and record the date when the diskette was first put into use. Remember, use only a felt tip pen for marking.

This "permanent" label is not a good place to record the contents of the diskette since that will change, and you don't want to be erasing or scratching out information from this label.

System Start-Up

1. Turn all peripherals on.
2. Turn the Computer on. Wait until all disk drive motors stop.
3. Locate the TRSDOS diskette that was supplied with the Disk System. Insert it into Drive 0, with the label side facing up and the read/write window pointing into the drive slot. See **Figure 6**.
4. When the diskette is fully inserted, close the drive door.
5. Press RESET. The Computer should now load TRSDOS and begin the start-up dialog described in the next section.

If nothing happens on the Display, or if the message: `DISKETTE?` or `NOT A SYSTEM DISK` is displayed, check the following:

- Are you using a TRSDOS "system" diskette?
- Is the diskette properly inserted into Drive 0?
- If external drives are present, are they properly connected and turned on?

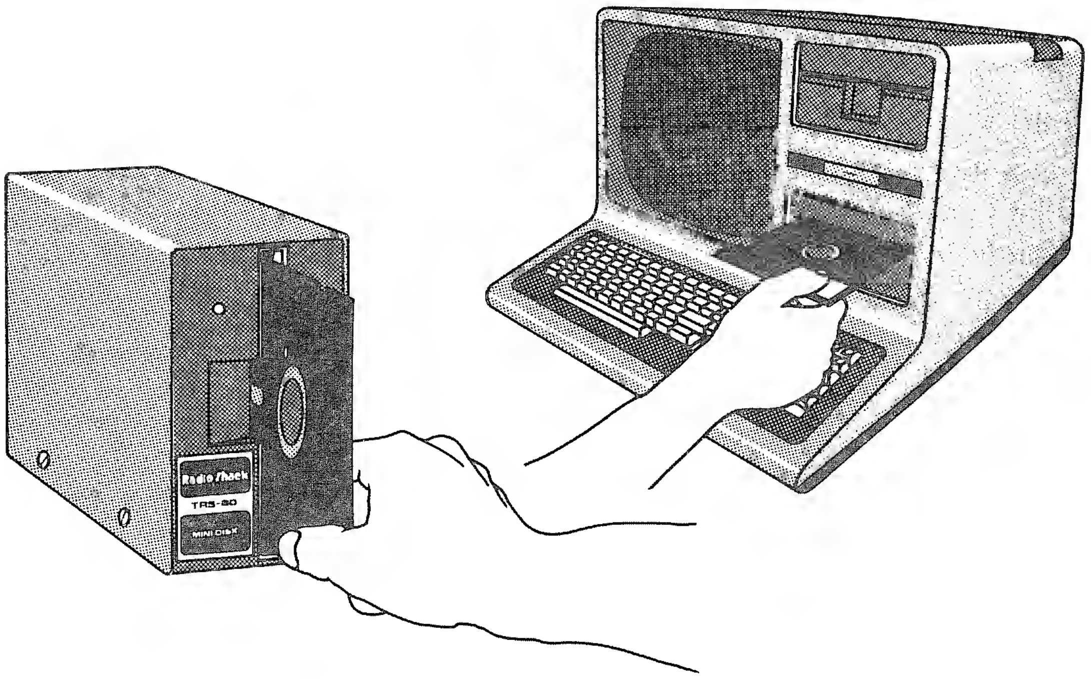


Figure 6. Inserting a Diskette.

If you can't find the problem, refer to the **Troubleshooting and Maintenance** chapter for further suggestions.

TRSDOS Start-Up Dialog

Whenever you reset the Model III Disk System, it loads TRSDOS and begins the start-up dialog.

1. The TRSDOS version number and date of creation will be displayed, followed by the amount of RAM (32K or 48K) and the number of drives in the system.
2. TRSDOS will prompt you to enter the date in the form MM/DD/YY. For example, 07/04/80 for July 4, 1980. Type in the correct date and press ENTER. TRSDOS will not continue until you type in the date correctly.
3. TRSDOS will prompt you to enter the time in 24-hour form HH:MM:SS. For example, 14:45:00 for 2:45 p.m. Type in the correct time and press ENTER. If you don't wish to set the time, simply press ENTER at the beginning of the line. TRSDOS will set the time to 00:00:00.
4. TRSDOS will now display the message, `TRSDOS READY`

Whenever this is displayed, you are in the TRSDOS READY mode, and you may type in a TRSDOS command.

Important Disk Operations

In this section we will describe three very important operations:

1. Duplicating the TRSDOS diskette (BACKUP)
2. Initializing a data diskette (FORMAT)
3. Converting files from Model I to Model III TRSDOS (CONVERT).

All new customers should complete the TRSDOS BACKUP procedure now; multi-drive customers should also complete the FORMAT operation for a few diskettes. Detailed information is provided later in this manual; here we will simply outline the procedures.

Making a BACKUP (Duplicate) of TRSDOS

Your first operation should be to duplicate the TRSDOS diskette you received from Radio Shack. The TRSDOS diskette contains a utility program called BACKUP to accomplish this.

1. Locate the TRSDOS diskette and a new, blank diskette. The TRSDOS diskette will be referred to as the "source," while the blank one will be called the "destination," during BACKUP.
2. Start TRSDOS as explained in the previous section. `TRSDOS READY` should be displayed.
3. Type: `BACKUP` (ENTER)
4. TRSDOS will now load and start BACKUP. It will ask you:

   `SOURCE DRIVE NUMBER?`

   Specify the drive which contains the original TRSDOS diskette by typing:

   `0` (ENTER)
5. Next TRSDOS will ask: `DESTINATION DRIVE NUMBER?`

   Now specify the drive which will be used for making the duplicate TRSDOS. If you have two or more drives in your system, type: `1`(ENTER)
6. TRSDOS will ask: `SOURCE DISK MASTER PASSWORD?`

   Type: `PASSWORD` (ENTER)

   (PASSWORD is the password of the supplied diskette.)
7. Now the duplication process will begin.

   If the destination diskette is not formatted, BACKUP will format it before continuing. (Before any diskette can be used, it must be initialized or "formatted"—the data regions defined and labeled, and a table of contents or "directory" created.)

If you are using a single-drive system, TRSDOS will prompt you to swap source and destination diskettes several times during the formatting/backup process.

After a single-drive BACKUP, TRSDOS will display the message:

```
INSERT SYSTEM DISKETTE (ENTER)
```

Be sure you have a TRSDOS diskette in Drive 0, then press (ENTER).

The duplication process is now complete. We suggest you save the original TRSDOS and use the duplicate as your working copy. If anything happens to the working copy, you can make another one from the original.

Making a Data Diskette (FORMAT)

This section applies to multi-drive systems only.

Drive 0 must always contain a TRSDOS diskette, so the Computer can have access to the system programs stored there. Much of the storage capacity of this diskette is taken up by the system programs.

However, the other drives in the system may contain "data" diskettes which have no system programs. All of the storage capacity of such diskettes is available for your programs and data.

The FORMAT utility program takes a diskette and initializes or "formats" it. If the diskette was previously formatted, all prior information can be lost. The resultant diskette contains no system files and may only be used in Drive 1, 2 or 3.

1. In the TRSDOS READY mode, type: `FORMAT` (ENTER)
2. TRSDOS will start the formatter program and ask you a series of questions:

   ```
   FORMAT WHICH DRIVE?
   ```

   Insert a blank diskette into Drive 1. Type: `1` (ENTER)

   ```
   DISKETTE NAME?
   ```

   This name will serve as an internal label for the diskette. Type in any appropriate name of one to eight letters and numbers, starting with a letter. Press (ENTER) at the end of the name.

   ```
   MASTER PASSWORD?
   ```

   The password may be from one to eight letters and numbers, starting with a letter. Press (ENTER) at the end of the password.

   Use of the password allows BACKUP, PROT, and PURGE access to all non-system files. Unless special protection is needed, we suggest you use the password PASSWORD. Whatever password you select, don't forget it!

If the diskette contains data, TRSDOS will warn you:

```
DISKETTE CONTAINS DATA, USE DISK OR NOT?
```

The warning is needed since FORMAT erases all previous information from the diskette. Type `N` (ENTER) to cancel FORMAT; type `Y` (ENTER) or `U` (ENTER) to continue it.

3. TRSDOS will now format and verify the diskette. The data diskette will then be ready for use in Drive 1, 2, or 3.

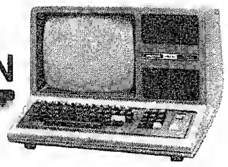
Model I/III Conversion (CONVERT)

In general, Model I TRSDOS diskettes cannot be used in a Model III Disk System. However, Model III TRSDOS includes a special program, CONVERT, to read a Model I TRSDOS diskette and copy its non-system files onto a Model III TRSDOS diskette.

In two-drive systems, the files must be copied onto a Model III system diskette; in three- or four-drive systems, the files may be copied onto a data diskette.

Here are abbreviated instructions for using this program. For further details, see CONVERT.

1. Using a Model I Disk System, remove all passwords from the diskette to be converted. You can do this with the PROT command, described in the *Model I TRSDOS/Disk BASIC Owner's Manual.*
2. Start Model III TRSDOS.
3. Place the Model I diskette in Drive 1, 2 or 3. (In two-drive systems, use Drive 1; in three- or four-drive systems, Drive 2.)
4. In three- or four-drive systems, place a Model III data diskette in Drive 1.
5. Type: `CONVERT` (ENTER)
4. The conversion program will start by asking for the source drive number. Type in the number of the drive containing the Model I diskette, then press (ENTER).
5. Next, the conversion program will ask for the destination drive number. Type in the number of the drive containing the Model III diskette, then press (ENTER). (In two-drive systems, use Drive 0; in three- or four-drive systems, Drive 1.)
6. Now all the non-system files will be converted and copied onto the destination diskette. As each file is copied, its name will be displayed.
7. When the process is completed, you may remove the Model I diskette. It is unchanged by the CONVERT program. The destination diskette contains the converted files.
8. To restore password protection to the converted files, you may use the PROT or ATTRIB command.



Disk BASIC

Quick Instructions for Using Disk BASIC

In this section we'll "walk you" through the following procedures:

1. Starting Disk BASIC
2. Running a simple program
3. Saving a program in a disk file
4. Loading a program from a disk file

For programming information, see the **Disk BASIC** section of this manual. Here we are showing procedures only.

Starting Disk BASIC

Under TRSDOS READY, type: `BASIC` (ENTER)

The Computer will load and start BASIC. First, it will ask two questions. Press (ENTER) in response to each of them.

```
HOW MANY FILES? (ENTER)
MEMORY SIZE? (ENTER)
```

A heading will be displayed, followed by:

```
READY
>
```

You may now begin using Disk BASIC.

Saving a Program

You should have a program in memory, and be in BASIC's READY mode. Type:

```
SAVE "PROGRAM" (ENTER)
```

BASIC should now save the program in a disk file we arbitrarily named "PROGRAM." Any other suitable **file name** would do.

Loading a Program

For this sample session, we will load the program just saved.

First type: `NEW` (ENTER) to erase it from memory. (This is to prove that it *can* be retrieved from the disk file.)

Now type: `LOAD "PROGRAM"` (ENTER) and BASIC will load the specified program.

You may now list it and run it.

For further information on using Disk BASIC, see Section 3 of this manual.

Setting the Cassette Baud Rate under Disk BASIC

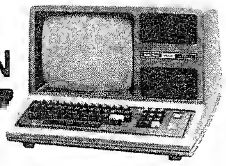
TRSDOS sets the cassette baud rate to High. If you would like to change this, use the following TRSDOS command:

```
PATCH BASIC/CMD (ADD=5202,FIND=00,CHG=FF) (ENTER)
```

Consequently, you will be prompted with: `CASS?` whenever you start Disk BASIC.

You may then type either `H` (High) or `L` (Low) to choose the rate you need.

To change the system diskette back to its original state (i.e., no `CASS?`), simply use the TRSDOS PATCH command again but reverse the FIND and CHG values.



Troubleshooting and Maintenance

If you have problems operating your Model III Disk System, please check the following symptoms and cures, and check the corresponding table in your Model III Manual.

If you can't solve the problem, take the unit to your local Radio Shack. We'll have it fixed and returned to you as soon as possible.

Symptom	Cure
Disk drive motors run continuously when the Computer is turned on.	Check external drive connection sequence. Drive 26-1164 must always be the last external drive.
Computer will not load TRSDOS.	1. Make sure you have inserted the TRSDOS diskette properly in Drive 0. 2. Make sure all peripherals are properly connected.
Error Messages	Look up the message in the TRSDOS or BASIC **Error Message** Section. The "cure" should be listed.
Frequent disk I/O errors	1. Diskette is partially erased. Backup the diskette, then re-format it. 2. Diskette is worn out. Use backup copy, if available, to make a new working copy. 3. Disk drives need cleaning or alignment by Radio Shack service technicians.

Maintenance

For reliable operation, the disk drives must be kept clean and properly aligned. These procedures should be done by Radio Shack service technicians, according to the following schedule:

Degree of Use	**Maintenance Interval**
Commercial data processing environment	Every month for medium use.
Occasional home use	Every 8-10 months; more often if needed.

For further instructions, see the **Troubleshooting and Maintenance** section in your Model III Manual.

Notation and Abbreviations

For the sake of clarity and brevity, we've used some special notation and type styles in this book.

CAPITALS and punctuation

indicate material which must be entered exactly as it appears. (The only punctuation symbols not entered are ellipses, explained below.) For example, in the line:

```
DUMP LISTER (START=7000,END=7100,TRA=7004)
```

every letter and character should be typed as indicated.

lowercase italics

represent words, letters, characters or values you supply from a set of acceptable values for a particular command. For example, the line:

LIST *filename*

indicates that you can supply any valid **file specification** after LIST.

. . .

Ellipsis indicates that the preceding items can be repeated. For example:

ATTRIB *filename (option, . . .)*

indicates that several options may be repeated inside the parentheses.

ϸ

This special symbol is used occasionally to indicate a blank-space character (ASCII code 32 decimal, 20 hexadecimal).

```
PRINT "ϸHϸIϸ!"
```

X'*nnnn*'

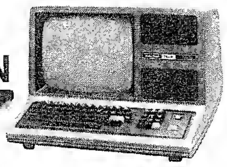
Indicates that *nnnn* is a hexadecimal number. All other numbers in the text of this book are in decimal form, unless otherwise noted.

X'7000'

indicates the hexadecimal value 7000 (decimal 28672).

`COMPUTER TYPE`

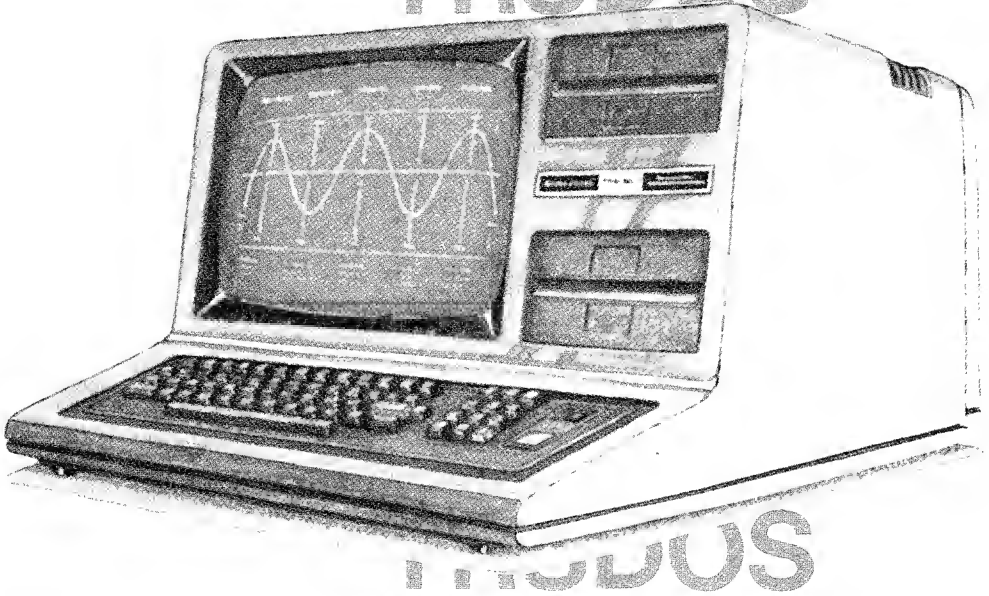
Any words, letters, or numbers that are displayed on the screen will be in computer type (dot-matrix). Uppercase letters are used; however, your screen at times may display lowercase letters instead.



Specifications

Diskettes	5¼″ mini-diskettes Radio Shack Catalog Number 26-305, 26-405 (package of three), or 26-406 (package of 10)
Diskette Organization (Formatted Diskette)	Single-sided Double-density 40 Tracks 18 Sectors/Track 256 Bytes/Sector
Operating Temperature	55 to 80 Degrees Fahrenheit 13 to 27 Degrees Celsius
Power Requirements (External Drives)	120 VAC, 60 Hz, 28 VA (240 VAC, 50 Hz, Australian; 220 VAC, 50 Hz, European)

TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS
TRSDOS



TRSDOS
TRSDOS
TRSDOS

Description of TRSDOS

What Is TRSDOS?

TRSDOS (pronounced "TRISS-DOSS") stands for "TRS-80 Disk Operating System." It fulfills three roles:

1. Master Program
2. Command Interpreter
3. Program Manager

As the master program, TRSDOS enables the microprocessor and its various components to interact efficiently. The components include:

- Random Access Memory (RAM). TRSDOS reserves space for its own needs and allocates space for user programs.
- Disk Drives. TRSDOS interfaces with the disk hardware and provides a file system for storing system and user data on diskettes.
- Input/output devices. These include the keyboard, video display, printer, and RS-232-C equipment.

TRSDOS is also a command interpreter. Whenever it displays `TRSDOS READY`, you may enter commands that control how the system works. These are known as "library" commands.

In its role as program manager, TRSDOS will load and run system or user programs. During this time, the system or user program is in control of the Computer.

Figure 7 illustrates the relationships between these three roles.

Where Does BASIC Fit In?

Referring to **Figure 7**, you'll see that Disk BASIC falls under the "language package" category.

Disk BASIC consists of some general enhancements to Model III BASIC, plus the disk input/output capability. It uses Model III BASIC (stored in ROM) whenever possible. For instance, the Model III BASIC ROM includes all of the mathematical functions.

If you're used to the non-disk system, there's one difference you should understand from the beginning: In the non-disk system, BASIC is in control when you start-up. In the disk system, however, TRSDOS is in control when you start-up. You have to tell TRSDOS to load and run BASIC. Only *then* can you begin running a program written in BASIC.

How TRSDOS Uses RAM

TRSDOS is stored on the system diskette included with your Disk System. Each time the Computer is turned on or reset, the TRSDOS master program is loaded into RAM so it can take charge.

TRSDOS occupies approximately 40,000 bytes of space on the diskette; however, only a portion of that is in RAM at once. This is possible because TRSDOS is divided into several independent ''modules.''

The ''resident'' module is in memory at all times. It consists of **input/output drivers**, tables, the **command interpreter**, and other essential routines.

Additional modules are loaded as needed, and replaced (or ''overlaid'') by other modules when they are no longer needed.

Note: After you enter a **library** or **utility command**, you will usually hear TRSDOS accessing the system disk. It is loading an overlay module which contains the code necessary to complete the command.

The Memory Map in **Figure 8** illustrates how TRSDOS utilizes the available memory space.

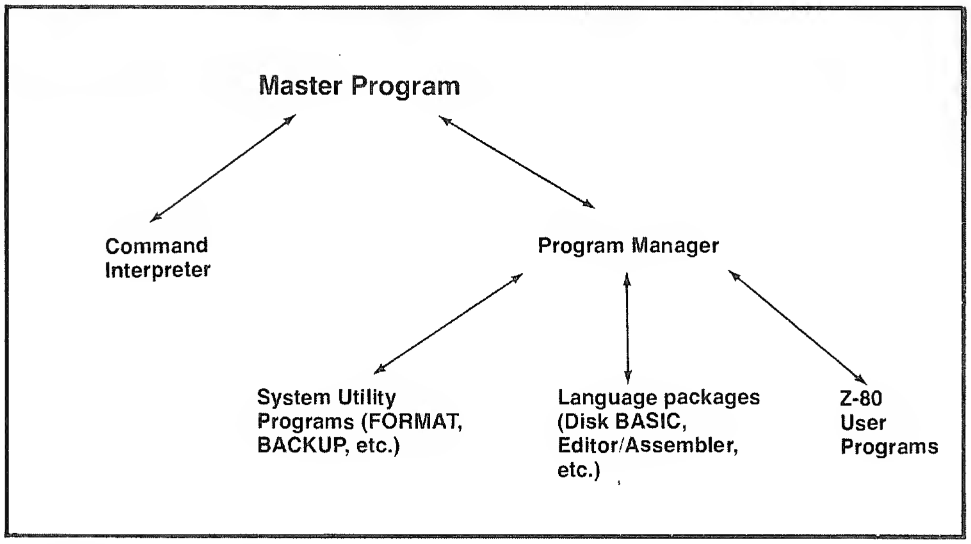

Master Program
Command Interpreter
Program Manager
System Utility Programs (FORMAT, BACKUP, etc.)
Language packages (Disk BASIC, Editor/Assembler, etc.)
Z-80 User Programs

Figure 7. TRSDOS Roles.

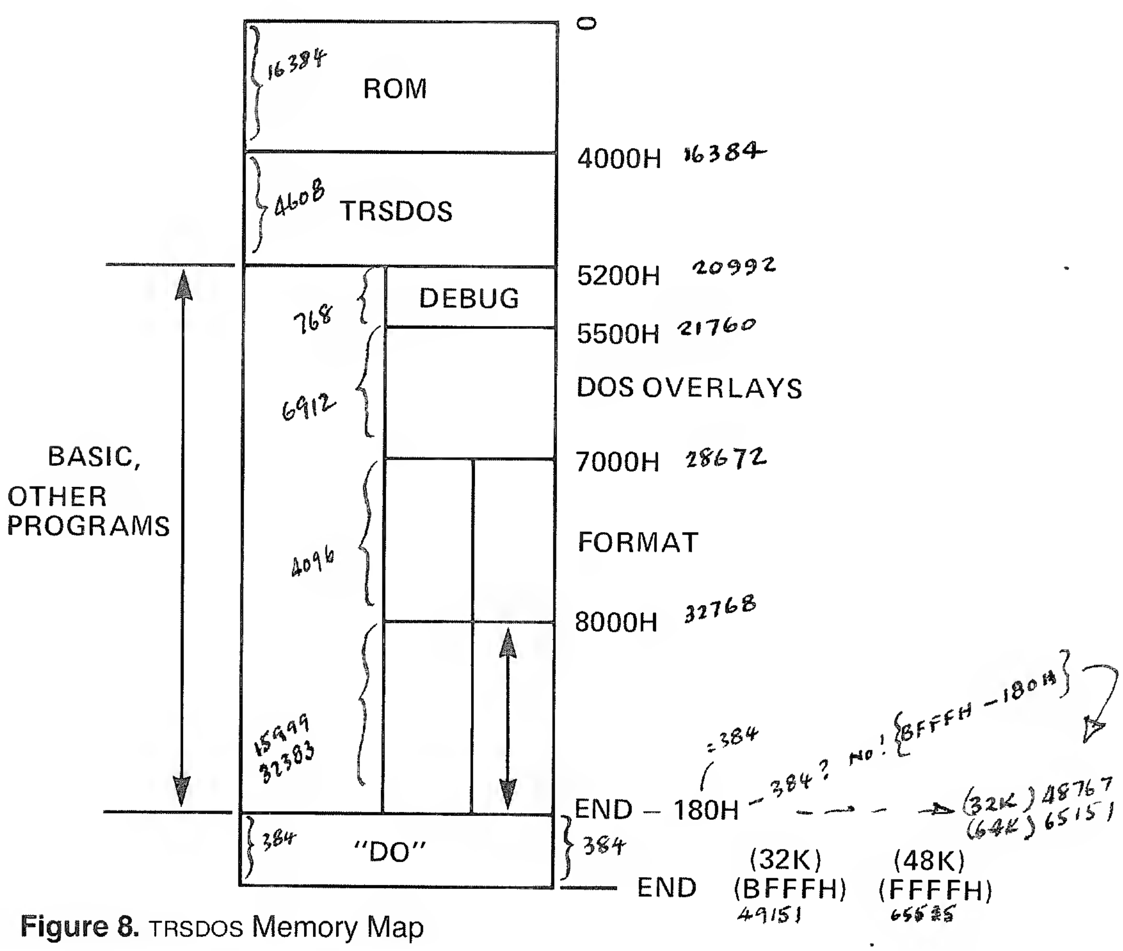


Figure 8. TRSDOS Memory Map

Using TRSDOS

Entering a Command

Whenever the TRSDOS prompt,

```
TRSDOS READY
```

is displayed, you can type in a command, which can be no more than 63 characters in length. You must press (ENTER) to end the line. TRSDOS will then "accept" the command.

For example, type: `CLS` (ENTER) TRSDOS will clear the Display and the TRSDOS READY prompt will reappear.

In general, your commands will require more than one word. For example, to kill (delete) a file named MYNAME, you have to specify the command and the filename.

```
KILL MYNAME (ENTER)
```

tells TRSDOS to find the file named MYNAME, eliminate it from the diskette, and release the space previously occupied by that file.

Whenever you type in a line, TRSDOS follows this procedure:

1. First it checks to see if what you've typed is the name of a TRSDOS command. If it is, TRSDOS executes it immediately.
2. If what you typed is not a TRSDOS command, then TRSDOS will check to see if it's the name of a program file on one of the drives.
3. When searching for a file, TRSDOS looks first through Drive 0, then Drive 1, etc., unless you include an particular **drive specification** with the file name — or specify the MASTER command (see **Library Commands**).

   If TRSDOS finds a specified user file, it will load and execute the file if it is a program file. Otherwise, you'll get an error message.

Command Syntax

Command syntax is a command's general form (like the grammar or structure of an English sentence). The syntax tells how to use keywords (such as DIR, LIST, CREATE, etc.) together with the necessary parameters and punctuation.

If you need help remembering the syntax form of a specific command while you're operating TRSDOS, type in:

`HELP` *command* (ENTER)

command should be the specific Library Command you're having trouble with. TRSDOS will give you the syntax format as well as a brief definition of the command.

Commands (Syntax Forms)

No-file commands

command (options) comment

> *options* is a list one or more parameters that may be needed by the command. Some commands have no options. The parentheses around options must be included unless the option itself is omitted.
>
> *comment* is an optional field used to document the purpose of the command. Comments are useful inside automatic command input files (see BUILD and DO).

One-file commands

command filename (options) comment

> *filename* is a standard TRSDOS file specification.
>
> *options*—See definition above.
>
> *comment*—See definition above.

Two-file commands

command filename delimiter filename (options) comment

filename is a standard TRSDOS file specification.

delimiter is a blank space.

options—See definition above.

comment—See definition above.

File Specification

The only way to store information on a diskette is to put it in a disk file. Afterwards, that information can be retrieved via the file name you gave the file when you created or renamed it.

A file specification has the general form:

filename/ext.password:d

filename consists of a letter followed by up to seven optional letters or numbers.

/ext is an optional name-extension; 'ext' is a sequence of up to three letters or numbers, starting with a letter.

.password is an optional password; 'password' is a sequence of up to eight letters or numbers, starting with a letter.

:d is an optional disk-drive specification; 'd' is one of the digits 0,1,2,3.

Note: There can be no blank spaces inside a file specification. TRSDOS terminates the file specification at the first blank space.

For example: `FILEA/TXT.MANAGER:3` references the file named FILEA/TXT with the password MANAGER, on Drive 3.

The name, extension, and drive-specification all contribute to the uniqueness of a file specification. The password does not. It simply controls access to the file.

File Names

A file name consists of a name and an optional name extension. For the name, you can choose any letter, followed by up to seven additional numbers or letters. To use a name extension, start with a diagonal slash / and add no more than three numbers or letters; however, the first character must be a letter.

For example:

MODEL3/TXT	INVENTORY	DATA11/BAS
NAMES/A12	AUGUST/A15	WAREHOUS
TEST	TEST1	TEST1/A

are all valid and distinct file names.

Although name extensions are optional, they are useful for identifying what type of data is in the file. For example, you might want to use the following set of extensions:

/BAS	for BASIC program
/TXT	for ASCII text
/CMD	for machine-language command file
/DAT	for data

One advantage of using extensions is that you can tell by just looking at the directory what kind of information a file contains.

Another advantage is that TRSDOS can recognize certain extensions. For example, if a file has the extension /CMD, the TRSDOS will load and attempt to execute that file when you type: *filename* (ENTER) omitting the extension /CMD.

Drive Specification

If you give TRSDOS a command such as: `KILL TEST/A` TRSDOS will search for the file TEST/A first in Drive 0, then in Drive 1, 2, and finally in Drive 3 until it finds the file.

Anytime you omit a drive-specification, TRSDOS will follow this sequence, unless you use the MASTER command.

It is possible to tell TRSDOS exactly which drive you want to use by specifying the drive. A drive specification consists of a colon followed by one of the digits 0,1,2, or 3, corresponding to one of the four drives you might be using.

For example: `KILL TEST/A:3` tells TRSDOS delete the file TEST/A on Drive 3 only.

Anytime TRSDOS has to open a file (e.g., to list it for you), it will follow the same sequence. When TRSDOS has to write a file, it will skip over any **write-protected** diskettes.

Password

You can protect a file from unauthorized access and use by assigning passwords to the file. That way, a person cannot gain access to a file without using the appropriate password.

It's important to realize that every file has a password, even if you didn't specify it when the file was created. In such instances, the password becomes eight blank spaces. In this case, the file becomes unprotected—anyone can gain total access simply by referring to the filename.

TRSDOS allows you to assign two passwords to a file:

- An "Update word," which grants total access to the information
- An "Access word," which grants limited access to the information (see ATTRIB)

When you create a file, the Update and Access words are both set equal to the password you specify. You can change them later with the ATTRIB command.

A password consists of a period followed by one to eight letters or numbers. If you do not assign a password to a file, TRSDOS uses a default password of eight blanks.

For example, suppose you have a file named SECRET/BAS. and the file has MYNAME as an update and access word. Then the command: `KILL SECRETS/BAS` will not cause the file to be killed. You must include the password MYNAME in the file specification.

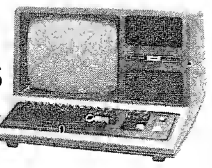
Suppose a file is named DOMAIN/BAS and has blanks for the password. Then the command: `KILL DOMAIN/BAS.GUESS` will not be obeyed, since GUESS is not the password.

A Few Important Definitions

System vs. Data Diskettes

A **system diskette** is one which contains the TRSDOS disk operating system software. Subject to space limitations, it may also contain your own files. A system diskette must always be in Drive 0 while the Computer is in use.

A **data diskette**, on the other hand, does not contain the operating system software, and therefore cannot be used in Drive 0. It may be used in Drive 1, 2 or 3. Such a diskette has a maximum of space available for storing your own programs and data.



System, Program, and Data Files

System files include the TRSDOS operating system software, the BASIC language interpreter, the FORMAT, BACKUP and CONVERT utilities, and other software which is released by Radio Shack. These files appear in the Directory with an "S" attribute. (See DIR)

Program files are stored in a special format which allows them to be loaded and executed directly from the TRSDOS READY level. For example, the BASIC interpreter is a program file.

Data files include all files that are not in the correct format to allow loading and executing from TRSDOS READY. For example, a program written in BASIC will be stored as a data file. It can be loaded and executed from BASIC, but not from TRSDOS READY.

Master Passwords

Each diskette is initially assigned a master password during FORMAT or BACKUP. (Your master password for TRSDOS is PASSWORD.) The master password allows you to use BACKUP, PURGE, and PROT on a diskette. Using a diskette's master password, you may change it (see PROT).

TRSDOS Library Commands

APPEND
Append files

APPEND *source-file destination-file*

source-file is the specification for the file which is to be copied onto the end of the other file.

destination-file is the specification for the file which is to receive the appendage (addition).

Note: Both *source-* and *destination-files* must be in ASCII format (data files or BASIC programs saved with the A option).

APPEND copies the contents of the source-file onto the end of the destination-file. The source-file is unaffected, while the destination-file is extended to include the source-file.

Note: The logical record lengths must match. See DIR for more information on logical record lengths.

Examples

```
APPEND WORDFILE/C WORDFILE/D
```

A copy of WORDFILE/C is appended to WORDFILE/D.

```
APPEND REGION1/DAT TOTAL/DAT.GUESS
```

A copy of REGION1/DAT is appended to TOTAL/DAT, which is protected with the password GUESS.

Sample Uses

Suppose you have two data files, PAYROLL/A and PAYROLL/B.

```
      PAYROLL/A
Atkins, W.R. .......
Baker, J.B. ........
Chambers, C.P. .....
Dodson, M.W. .......
KicKamon, T.Y. .....
```

```
      PAYROLL/B
Lewis, G.E. ........
Miller, L.O. .......
Peterson, B. .......
Rodriguez, F. ......
```

You can combine the two files with the command:

```
APPEND PAYROLL/B PAYROLL/A
```

PAYROLL/A will now look like this:

```
      PAYROLL/A
Atkins, W.R. .......
Baker, J.B. ........
Chambers, C.P. .....
Dodson, M.W. .......
KicKamon, T.Y. .....
Lewis, G.E. ........
Miller, L.O. .......
Peterson, B. .......
Rodriguez, F. ......
```

PAYROLL/B will be unaffected. To see the APPENDed file, type `LIST PAYROLL/A (ASCII)`.

Note: Do not load a program under BASIC after an APPEND.

ATTRIB
Change a File's Password

ATTRIB *file (visibility,*ACC = *name,*UPD = *name,*PROT = *level)*

file **is the file specification.**

visibility **must be I or N. Tells TRSDOS whether the file is Invisible (I) or Non-invisible (N) (see DIR). If omitted, *visibility* is unchanged.**

ACC = *name* tells TRSDOS the access word. If omitted, the access word is unchanged. If ACC = , is used, the access word is set to blanks.

UPD = *name* tells TRSDOS the update word. If omitted, the update word is unchanged. If UPD = , is used, the update word is set to blanks.

PROT = *level* tells TRSDOS the protection level for access. If omitted, *level* is unchanged.

Level	Degree of access granted by access word
FULL	Full access, no protection.
KILL	Kill, rename, read, execute, and write (gives total access, i.e., the least-protected).
NAME	Rename, read, execute, and write.
WRITE	Read, execute, and write.
READ	Read and execute.
EXEC	Execute only.

Note: Each level allows access to itself plus all lower levels.

ATTRIB lets you change the passwords to an existing file or makes the file invisible or non-invisible. Passwords are initially assigned when the file is created. At that time, the update and access words are set to the same value (either the password you specified or a blank password).

Examples

```
ATTRIB DATAFILE (I,ACC=JULY14,UPD=MOUSE,PROT=READ)
```

Makes the file invisible, sets the access password to JULY14 and the update password to MOUSE. Use of the access word will allow only reading and executing the file.

```
ATTRIB PAYROLL/BAS.SECRET (N,ACC=,)
```

Sets the access word to blanks. The file is made non-invisible and the protection level assigned to the update word is left unchanged.

```
ATTRIB OLD/DAT.APPLES (UPD=,)
```

Sets the update word to blanks.

```
ATTRIB PAYROLL/BAS.PW (PROT=EXEC)
```

Leaves the access and update words unchanged, but changes the level of access.

Sample Uses

Suppose you have a data file, PAYROLL, and you want an employee to use the file in preparing paychecks. You want the employee to be able to read the file but not to change it. Then use a command like:

```
ATTRIB PAYROLL (I,ACC=PAYDAY,UPD=AVOCADO,PROT=READ)
```

Now tell the clerk to use the password PAYDAY (which allows read only); while only you know the password, AVOCADO, which grants total access to the file.

Protecting BASIC Programs

You may give a BASIC program execute-only protection using the ATTRIB command. For example, suppose the program is named TEST (no password). Under TRSDOS READY, execute this command:

```
ATTRIB TEST (ACC=,UPD=VALLEY,PROT=EXEC)
```

Now TEST has a blank access password, an update password of VALLEY, and an access level of execute only. Without using the update password, there is now only one way to execute the program:

1. Start BASIC.
2. Type: `RUN "TEST"`

   (This is the only way to access the program. If the operator attempts to LOAD it instead, BASIC will erase the program from memory before returning with READY.)

After the RUN "TEST" command, BASIC will load and execute the program. If the operator presses (BREAK) or if the program ends normally, BASIC will erase the program before returning with the READY message. This makes it impossible to obtain a listing of the program—unless the update password is used.

Of course, if you use the update password, you may gain full access to the program.

AUTO
Automatic Command after System Start-up

AUTO *command-line*

command-line **gives TRSDOS a command or the name of an executable program file created by BUILD.**

if *command-line* is given, the command will be executed on reset/ power-up.

if *command-line* is omitted, the previous AUTO command is erased from the diskette.

This command lets you provide a command to be executed whenever TRSDOS is started (power-up or reset). You can use it to get a desired program running without any operator action required, except for typing in the date and time.

When you enter an AUTO command, TRSDOS writes command-line into its start-up procedure. TRSDOS does not check for valid commands; if the command line contains an error, it will be detected the next time the System is started up.

Examples

```
AUTO DIR (SYS)
```

Tells TRSDOS to execute the command DIR (SYS) after the start-up procedure. Each time the System is reset or powered up, it will automatically execute that command after you enter the date and time.

```
AUTO BASIC
```

Tells TRSDOS to load and execute BASIC each time the System is started up.

```
AUTO FORMS (WIDTH=80)
```

Tells TRSDOS to reset the printer width parameter each time the System is started up.

```
AUTO PAYROLL/CMD
```

Tells TRSDOS to load and execute PAYROLL/CMD (must be a machine-language program) after each System start-up.

```
AUTO DO STARTER
```

Tells TRSDOS to take automatic key-ins from the file named STARTER after each System start-up. See BUILD and DO.

To Erase an AUTO Command

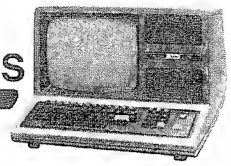
Type: `AUTO` (ENTER)

This tells TRSDOS to erase any automatic command. The command will be deleted the next time you power-up or RESET the System.

The acknowledgement: `AUTO = ""` is displayed after an erasure.

To Override an AUTO Command

You can bypass any automatic command by holding down (ENTER) while pressing RESET. You must continue holding down (ENTER) until you are prompted for the date during the initialization process.



BUILD
Create an Automatic Command Input File

BUILD *file*

file is a file specification which cannot include an extension.

This command lets you create an automatic command input file which can be executed via the DO command. The file must contain data that would normally be typed in from the keyboard to the TRSDOS READY mode.

BUILD files are intended for passing command lines to TRSDOS just as if they'd been typed in at the TRSDOS READY level. **Note:** CLEAR *cannot* be used in a DO file.

When you enter the BUILD command, BUILD creates the file and immediately prompts you to begin inserting lines. Each time you complete a line, press (ENTER). (While typing in a line, you can use the usual cursor control keys for erasures and corrections.)

To end the BUILD file, simply press (BREAK) at the beginning of a line.

First type: `BUILD` *filename*

You will then be prompted to type in the command text. You then type in up to 63 characters, then press (ENTER). You may enter as many lines as necessary. Press (BREAK) to quit and return to TRSDOS READY.

A Sample BUILD-File

Here's a hypothetical BUILD-file that initializes the serial interface and the printer driver:

```
SETCOM (BAUD=1200,WAIT)

FORMS (WIDTH=80)

PAUSE SERIAL INTERFACE & PRINTER INITIALIZED
```

CLEAR
Clear User Memory

CLEAR (START = *aaaa*,END = *bbbb*,MEM = *cccc*)

START = *aaaa* tells TRSDOS where to start clearing user memory. *aaaa* is a four-digit hexadecimal number from 6000 to the end of user memory. If this option is omitted, 6000 is used. If this option is used, END = *bbbb* must also be used.

END = *bbbb* tells TRSDOS to clear user memory to a specified end. *bbbb* is a four-digit hexadecimal number no less than the START number and no greater than the top of memory. If this option is used, START = *aaaa* must also be used.

MEM = *cccc* sets the memory protect address. *cccc* is a four-digit hexadecimal number from 0000 to FFFF. If this option is omitted, the memory protect address is reset to end of user RAM.

If all options are omitted, all available RAM is cleared, memory-protect is reset to end of memory, the Display is cleared, all I/O drivers are reset (see Memory Requirements of TRSDOS).

This command gets you off to a fresh start.

Depending on the options you select, this command will:

- Zero user memory (load binary zero into each memory address above 6000)
- Clear the Display
- Un-protect all memory

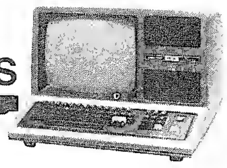
See **Memory Requirements of TRSDOS** for more information on the memory-protect address. **Note:** CLEAR *cannot* be used in a DO file.

Example

```
CLEAR (START=9000,END=0A000)
```

Note: Hexadecimal numbers which begin with a letter must be prefaced by zero (see above example).

```
CLEAR (MEM=7000)
```



CLOCK
Turn On Clock Display

CLOCK (*switch*)

switch gives TRSDOS one of two options, ON or OFF.

If option is omitted, TRSDOS uses ON.

This command controls the real-time clock display in the upper right corner of the Video Display. When it is on, the 24-hour time will be displayed and updated once each second, regardless of what program is executing.

Clock display is OFF at TRSDOS start-up.

Note: Except during cassette and disk I/O, the real-time clock is always running, *regardless* of whether the clock display is on.

Examples

```
CLOCK
```

Turns on the clock display.

```
CLOCK (OFF)
```

Turns the clock display off.

See TIME and DATE.

CLS
Clear the Screen

CLS

This command clears the Display and puts it in the 64 character/line mode.

Example

```
CLS
```

COPY
Copy a File or Files

Three forms:

A) COPY ***source-file destination-file***

source-file **is a file specification for the file to be copied.**

destination-file **is a file specification for the name and drive of the duplicate file.**

B) COPY ***source-file :d***

source-file **is defined above.**

:*d* tells TRSDOS to copy the file onto drive *d*, using the same file name.

C) COPY ***/ext :d***

/ext **is a "wild-card" file specification in which the file name is omitted and the extension is given. TRSDOS will copy all files which have a matching extension, regardless of the file name.**

:*d* is defined above.

This command copies source-file into the new file defined by destination-file. This allows you to copy a file from one disk to another, using a single drive if necessary. (In the latter case, you must include drive specifications in *both* file specifications.) For single-drive systems (Drive 0), both diskettes must contain TRSDOS. (i.e., Data diskettes aren't allowed in Drive 0.)

Examples

```
COPY OLDFILE/BAS NEWFILE/BAS
```

Copies OLDFILE/BAS into a new file named NEWFILE/BAS. TRSDOS will search through all drives for OLDFILE/BAS, and will copy it onto the first disk which is not write-protected.

```
COPY NAMEFILE/TXT :1
```

This command specifies a file named NAMEFILE/TXT to another disk.

```
COPY FILE/EXT:0 :1
```

This command copies FILE/EXT from Drive 0 to Drive 1.

```
COPY /BAS:0 :1
```

tells TRSDOS to copy all Drive 0 files which have the extension /BAS. The files will be copied onto Drive 1, using their present file names and extensions.

Sample Use

Whenever a file is updated, use COPY to make a backup file on another disk. You can also use COPY to restructure a file for faster access. Be sure the destination disk is already less segmented than the source disk; otherwise the new file could be more segmented than the old one. (See FREE for information on file segmentation.)

To rename a file on the same disk, use RENAME, not COPY.

CREATE
Create a Pre-allocated File

CREATE *filename* (LRL = *aaa*,REC = *bbb*)

> ***filename* is the file specification.**
>
> **LRL = *aaa* is the logical record length. *aaa* is a decimal number between zero and 255. If omitted, 256 is assumed.**
>
> **REC = *bbb* is the number of records to allow for. *bbb* is the number of records desired. If omitted, no records are allocated.**

This command lets you create a file and pre-allocate (set aside) space for its future contents. This is different from the default (normal) TRSDOS procedure in which space is allocated to a file dynamically, i.e., as necessary when data is written into the file.

If you open the file for sequential writes, TRSDOS will de-allocate (recover) any unused granules when the file is closed. If you open the file for random access, TRSDOS will not de-allocate space when the file is closed.

You may want to use CREATE to prepare a file which will contain a known amount of data. This will usually speed up file write operations. File reading will also be faster, since pre-allocated files are less segmented or dispersed on the disk — requiring less motion of the read/write mechanism to locate the records.

Examples

```
CREATE DATAFILE/BAS (REC=300, LRL=0)
```

Creates a file named DATAFILE/BAS, and allocates space for 300 256-byte records.

```
CREATE NAMES/TXT.IRIS (LRL=64,REC=50)
```

Creates a file named NAMES/TXT protected by the password IRIS. The file will be large enough to contain 50 records, each 64 bytes long.

```
CREATE PAYROLL/BAS
```

Creates a file named PAYROLL/BAS but allocates no space to it.

Sample Use

Suppose you are going to store personnel information on no more than 250 employees, and each data record will look like this:

Name (Up to 25 letters)
Social Security Number (11 characters)
Job Description (Up to 92 characters)

Then your records would need to be 25 + 11 + 92 = 128 bytes long.

You could create an appropriate file with this command:

```
CREATE PERSONNL/TXT (REC=250,LRL=128)
```

Once created, this pre-allocated file would allow faster writing than would a dynamically allocated file, since TRSDOS would not have to stop writing periodically to allocate more space (unless you exceed the pre-allocated amount).

DATE
Reset or Get Today's Date

DATE *mm/dd/yy*

***mm/dd/yy* is the specification for the month (*mm*), day (*dd*) and year (*yy*).**

Each must be a two-digit decimal number between the following ranges:

mm	**01-12**
dd	**00-31**
yy	**00-99**

The specifications are an option; however, if one specification is used, they all must be used.

If *mm/dd/yy* is omitted, TRSDOS displays the current date.

If *mm/dd/yy* is given, TRSDOS resets the date.

This command lets you reset the date or display the date.

You initially set the date when TRSDOS is started up. After that, TRSDOS updates the date automatically, using its built-in calendar. You can enter any two-digit year after 1900.

When you request the date, TRSDOS displays it in the format: `07/25/80` for July 25, 1980.

Examples

```
DATE
```

Displays the current date.

```
DATE 07/18/80
```

Resets the date to July 18, 1980.

DEBUG
Start Debug Monitor

DEBUG

This command starts the debug monitor, which allows you to enter, test, and debug machine-language programs.

Its features include:

- Full- or half-screen displays of memory contents
- Commands for modifications to RAM and register contents
- Single-step execution of programs
- Breakpoint interruption of program execution
- Transfer of control (Jump)
- "Editing" of disk-files

DEBUG uses the memory area from X'4E00' to X'54FF' (see **TRSDOS Memory Map**). DEBUG can only be used on programs in the user area X'5500' to TOP.

Examples

```
DEBUG
```

Turns DEBUG ON. Press (Q) to quit debugging and return to TRSDOS.

```
Q
```

Turns DEBUG OFF.

Command Description

Debug commands are usually entered by pressing a single key. In most cases, you do not have to press (ENTER) after the command has been typed in. Either a prompt will immediately be displayed or DEBUG will execute the operation without further instruction.

In some cases, you will have to enter a specific hexadecimal value or address (see R and J commands, for instance). Instead of pressing (ENTER) after the address is typed in, you will have to press (SPACEBAR).

Once you have entered the DEBUG program, you may use any of the following special commands:

D (Display Memory Contents)

Press (D) to display the contents of memory. TRSDOS will respond with the prompt: `D ADDRESS =` You should type in the hexadecimal address of the memory location you wish to see.

The display will be either half- or full-screen, depending on the format you are currently using (see below).

X (Half-Screen Display)

Press (X) to put the Display in the half-screen format. A 128-byte block of memory will be displayed starting with the next lowest address which is a factor of 16.

Figure 9 shows a typical half-screen format.

S (Full-Screen Display)

Press (S) to display the contents of a 256-byte block of memory starting with the next lowest address which is a factor of 256.

Note: The last 16 bytes on the Display will be overlaid by any command line typed in after the full-screen display is updated.

M (Modify RAM)

Press (M) to change to the disk utility display format (see the F command). TRSDOS will respond with the prompt: `M ADDRESS =` You should type in the four-digit hexadecimal address of the memory location you wish to modify, followed by a blank space (anything other than a space will abort the command).

The display will change to the memory edit format. The cursor will appear as a blinking character at the specified location.

To exit the modify mode, press (ENTER) to keep all changes made.

R (Change Register Contents)

Type:

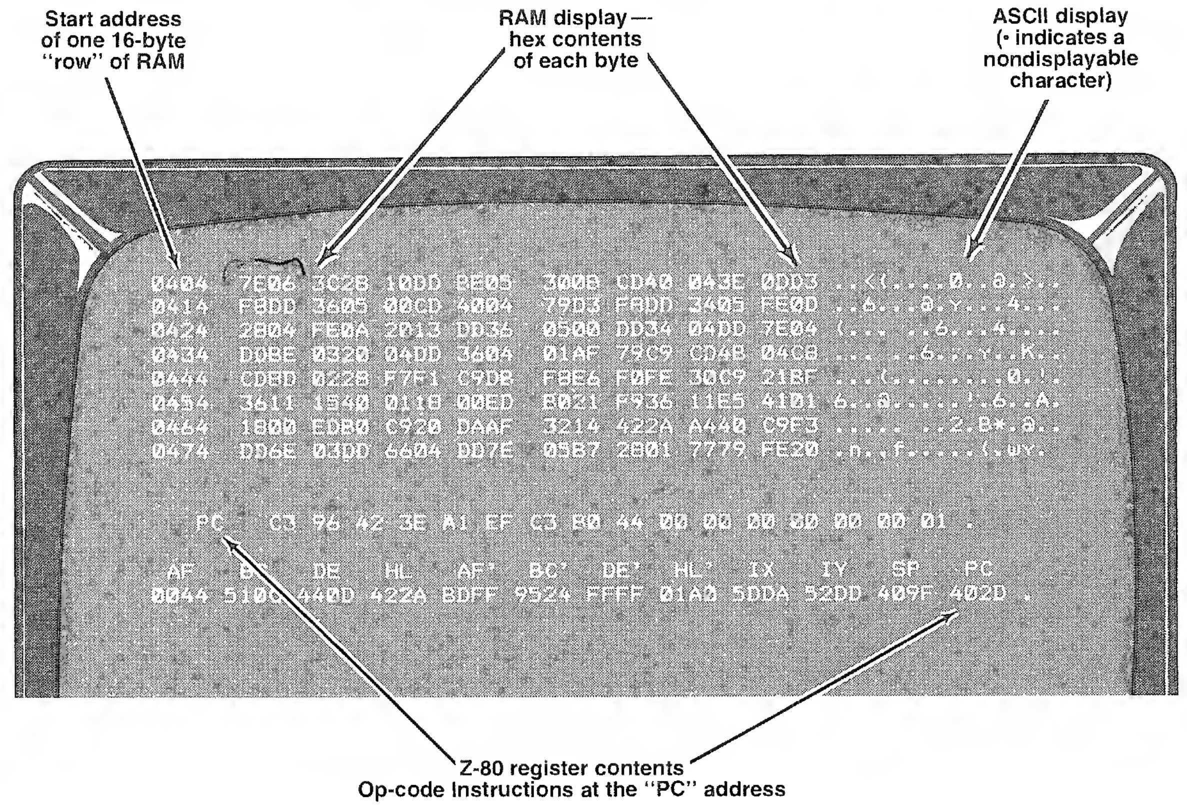
R*aa,bbbb* (SPACEBAR)

_aa_ is the name of one of the register pairs AF, BC, DE, HC, or PC.

bbbb is the four-digit hexadecimal value which will be loaded into _aa_. If fewer than four digits are typed in before pressing (SPACEBAR), leading zeros are assumed.

I (Instruction Single-Step)

Pressing (I) will allow the Computer to execute a single Z-80 instruction. The display will then be updated.


Start address of one 16-byte "row" of RAM
RAM display— hex contents of each byte
ASCII display (• indicates a nondisplayable character)
Z-80 register contents
Op-code Instructions at the "PC" address

```
0404  7E06 3C28 10DD BE05  300B CD40 043E 0DD3 ..<(....0..@.>..
0414  FBDD 3605 00CD 4004  79D3 FBDD 3405 FE0D ..6...@.y...4...
0424  2804 FE0A 2013 DD36  0500 DD34 04DD 7E04 (.... ...6...4....
0434  DDBE 0320 04DD 3604  01AF 79C9 CD4B 04C8 .... ...6...y..K..
0444  CDBD 0228 F7F1 C9DB  FBE6 F0FE 30C9 21BF ...(........0.!.
0454  3611 1540 0118 00ED  B021 F936 11E5 4101 6..@.....!.6..A.
0464  1800 EDB0 C920 DAAF  3214 422A A440 C9F3 ..... ..2.B*.@..
0474  DD6E 03DD 6604 DD7E  05B7 2801 7779 FE20 .n..f.....(.wy.

   PC   C3 96 42 3E A1 EF C3 B0 44 00 00 00 00 00 00 01 .

 AF   BC   DE   HL   AF'  BC'  DE'  HL'  IX   IY   SP   PC
0044 5100 440D 422A BDFF 9524 FFFF 01A0 5DDA 52DD 409F 402D .
```

Figure 9. Half-Screen Format.

The instruction in the memory contents referenced by the program counter is executed. The program counter is increased by the appropriate value, and the control is returned to DEBUG.

DEBUG will not, however, step through a call or jump into a ROM address.

C (Call Single-Step)

If you wish to complete an entire call/return sequence, press Ⓒ. The call is then executed and control is returned to DEBUG when the subroutine returns. Otherwise, this instruction acts just like the I command.

You will not be able to step through a call or jump into a ROM address.

U (Update)

Pressing Ⓤ causes the Display to be updated repeatedly. Press any key to exit from this mode.

; (Increment Display Address)

If the Display is half-screen, the first location shown is incremented by 16 when you press ;. If the full-screen format is displayed, the starting address will be incremented by 256.

– (Decrement Display Address)

If the Display is half-screen, the first location is decremented by 16 when you press –. If the full-screen format is displayed, the starting address will be decremented by 256.

J (Jump)

Press J to transfer control to a machine-language program, setting optional breakpoints.

Debug will respond with the prompt: `J ADDRESS? =`

You may type in a jump address and a breakpoint address. The command is terminated when you press ENTER. Type in the addresses in one of three formats:

```
J ADDRESS? = aaaa,bbbb ENTER
J ADDRESS? = aaaa ENTER
J ADDRESS? = ,bbbb ENTER
```

aaaa **is a four-digit hexadecimal address specifying the jump destination. If omitted, the address in the PC register is used.**

bbbb **is a four-digit hexadecimal address specifying a breakpoint. Before the Computer executes an instruction at this address, it will return control to DEBUG. If this address is omitted, control will not return to DEBUG.**

Notes: Breakpoints must be set at the *beginning* of Z-80 instructions. You may not set breakpoints in ROM addresses. The breakpointed address will contain an X'F7' until the breakpoint is encountered. Then the original contents will be restored and DEBUG will take control again.

Q (Quit)

Pressing Q turns off DEBUG and returns control to TRSDOS.

F (File Patch Utility)

This command lets you load and modify the contents of a diskette file.

When you press (F), DEBUG will respond with the prompt: `FILESPEC?`. Enter the name of the file to be patched.

DEBUG will set up a full-screen display showing the first 256 bytes in the file. You can "page" through the file using the (;) and (−) keys.

Figure 10 gives a typical display.

In this file-display mode, both hexadecimal and ASCII are given for each byte. If a code has no displayable character, a period is shown in the ASCII area.

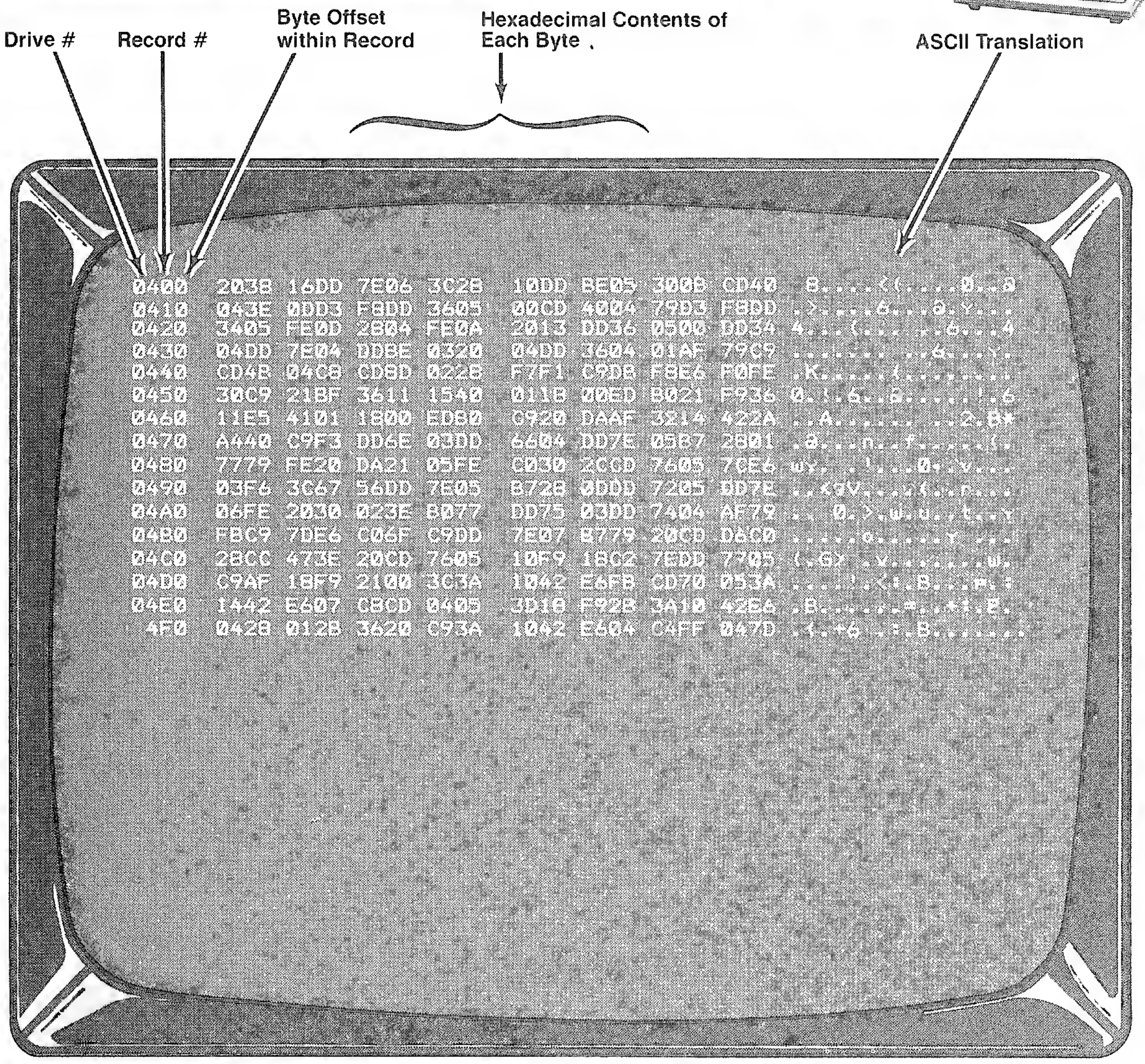
The display control commands are like those for the normal file-display mode:

(;)	Next page
(−)	Previous page

To change the file contents, press (M). This puts you in a modify-memory mode like the one previously described. Use the arrow keys to position the cursor (a blinking character), then type in the correct contents as a hexadecimal value. When you are through changing a page on the display, press (ENTER). The diskette file will be updated and you will be returned to the file-display mode.

To cancel changes made, do not press (ENTER), press (BREAK). This will put you back in the file-display mode without updating the diskette file. You may press (;) then (−) to restore the page display to its actual contents.

To quit patching a file, press (BREAK) while in the file display mode. DEBUG will prompt you for a new file specification. Press (BREAK) again and you will be returned to TRSDOS READY.



```
0400  2038 16DD 7E06 3C28  10DD BE05 300B CD40   8....<(....0..@
0410  043E 0DD3 F8DD 3605  00CD 4004 79D3 F8DD  .>....6...@.y...
0420  3405 FE0D 2804 FE0A  2013 DD36 0500 DD34  4...(.....6...4
0430  04DD 7E04 DDBE 0320  04DD 3604 01AF 79C9  ..........6...y.
0440  CD4B 04C8 CD8D 022B  F7F1 C9D8 F8E6 F0FE  .K.....{........
0450  30C9 21BF 3611 1540  011B 00ED B021 F936  0.!.6..@.....!.6
0460  11E5 4101 1800 ED80  C920 DAAF 3214 422A  ..A.........2.B*
0470  A440 C9F3 DD6E 03DD  6604 DD7E 05B7 2801  .@...n..f.....(.
0480  7779 FE20 DA21 05FE  C030 2CCD 7605 7CE6  wy...!...0..v...
0490  03F6 3C67 56DD 7E05  B728 0DDD 7205 DD7E  ..<gV....((..n..
04A0  06FE 2030 023E B077  DD75 03DD 7404 AF79  .. 0.>.w.u..t..y
04B0  FBC9 7DE6 C06F C9DD  7E07 B779 20CD D6C0  ......o.......y..
04C0  28CC 473E 20CD 7605  10F9 18C2 7EDD 7705  (.G> .v.......w.
04D0  C9AF 18F9 2100 3C3A  1042 E6FB CD70 053A  .....<!.B...m.:
04E0  1442 E607 C8CD 0405  3D10 F928 3A10 42E6  .B......=..+!.B.
 4F0  0428 012B 3620 C93A  1042 E604 C4FF 047D  .(.+6 .:.B......
```

Figure 10. Full-Screen Format

DIR
List the Diskette Directory

DIR *:d* (INV,SYS,PRT)

:d **is the desired drive directory. If omitted, Drive 0 is assumed.**

INV lists the invisible user files. If omitted, non-invisible user files are listed.

SYS lists system and user files. If omitted, only non-invisible user files are listed.

PRT lists the directory to the Printer. If omitted, the directory will be listed on the Video Display only.

If no option is given, TRSDOS lists non-invisible user files in Drive 0.

This command gives you information about a disk and the files it contains.

To pause the listing, press (@). To continue, press (ENTER). To terminate the listing, press (BREAK).

Examples

```
DIR
```

Displays the directory of non-invisible user files in Drive 0.

```
DIR :1 (PRT)
```

Lists the directory of the user files in Drive 1 to the Printer.

Sample Directory Listing

(See **Figure 11.**)
Definition of column headings

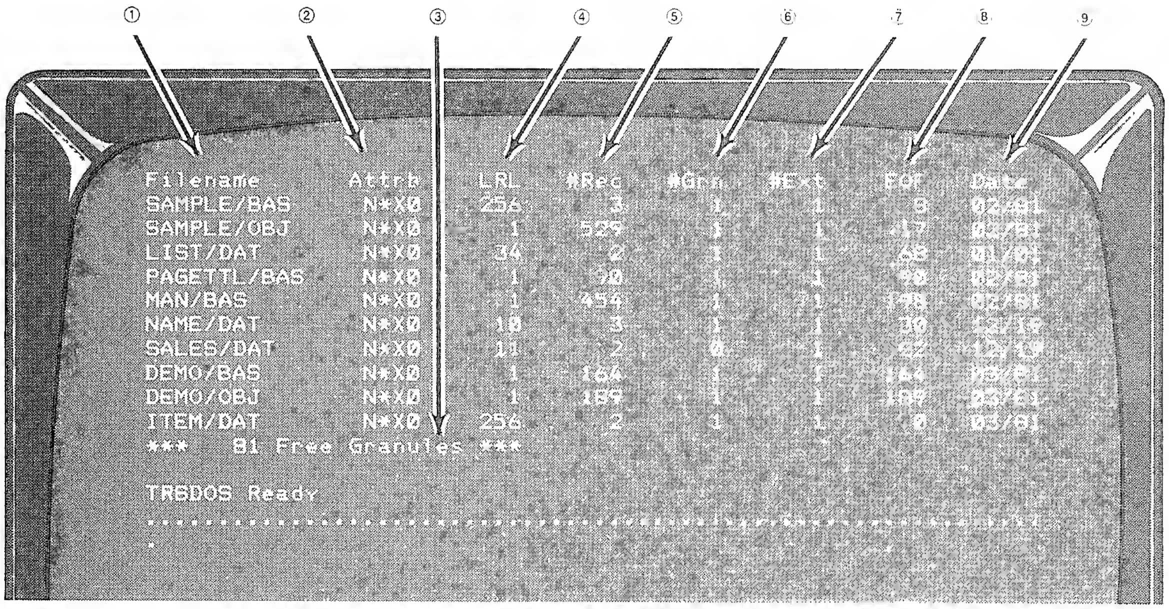
① File Name — The name and extension assigned to a file when it was created. The password (if any) is not shown.

② Attributes — A four-character field.

The first character is either I (Invisible) or N (Non-invisible).

The second character is S (System) or * (User) file.

The third character gives the password protection status:



```
Filename      Attrb   LRL  #Rec  #Grn  #Ext  Eof  Date
SAMPLE/BAS    N*XØ    256     3     1     1    9  02/81
SAMPLE/OBJ    N*XØ      1   529     1     1   17  02/81
LIST/DAT      N*XØ     34     2     1     1   68  01/81
PAGETTL/BAS   N*XØ      1    90     1     1   90  02/81
MAN/BAS       N*XØ      1   454     1     1  198  02/81
NAME/DAT      N*XØ     10     3     1     1   30  12/79
SALES/DAT     N*XØ     11     2     Ø     1   42  12/79
DEMO/BAS      N*XØ      1   164     1     1  164  03/81
DEMO/OBJ      N*XØ      1   189     1     1  189  03/81
ITEM/DAT      N*XØ    256     2     1     1    Ø  03/81
***   81 Free Granules ***

TRSDOS Ready
```

Figure 11. Directory Listing.

X	The file is unprotected (no password).
A	The file has an access word but no update word.
U	The file has an update word but no access word.
B	The file has both update and access words.

The fourth character specifies the level of access assigned to the access word:

0	Total access
1	Kill file and everything listed below.
2	Rename file and everything listed below.
3	This designation is not used.
4	Write and everything listed below.
5	Read and everything listed below.
6	Execute only.
7	No access.

③ Number of Free Granules — How many free granules remain on the diskette.

④ Logical Record Length — Assigned when the file was created.

⑤ Number of Records — How many logical records have been written.

⑥ Number of Granules — How many granules have been used in that particular file.

⑦ Number of Extents — How many segments (contiguous blocks of up to 32 granules) of disk space are allocated to the file.

⑧ End of File (EOF) — Shows the last byte number of the file.

⑨ Creation Date — When the file was created.

DO
Begin Auto Command Input from a BUILD-File

DO *command-line*

command-line **is the name of file created with BUILD. No extension should be specified. The file will automatically be given the extension /BLD.**

This command reads and executes the lines stored in a special-format file created with the BUILD command. The System executes the commands just as if they had been typed in from the Keyboard.

Command lines in a BUILD file may include library commands or file specifications for user programs.

When DO reaches the end of the automatic command input file, it returns control to TRSDOS.

The DEBUG and CLEAR command *cannot* be included in a BUILD file.

In addition to executing TRSDOS library commands, you can load and execute user programs from a DO-file. You will probably want to make your program name be the last line in the DO-file.

Examples

```
DO STARTER
```

TRSDOS will begin automatic command input from STARTER, after the operator answers the Date and Time prompts.

```
AUTO DO STARTER
```

Whenever you start TRSDOS, it will begin automatic command input from STARTER.

Sample Uses

Suppose you want to set up the following TRSDOS functions automatically on start-up:

```
FORMS (WIDTH=80)
CLOCK (ON)
```

Then use BUILD to create such a file. If you called it BEGIN, then use the command: `AUTO DO BEGIN` to perform the commands each time TRSDOS starts up.

DUAL
Duplicate Output to Video and Printer

DUAL (*switch*)

switch **is one of two options, ON or OFF. If omitted, TRSDOS uses ON.**

This command duplicates all video output to the Printer, and vice versa. It takes effect immediately.

Notes:

1. Video and printer output may be different because of intrinsic differences between output devices and output software.
2. Using the DUAL command will slow down the video output process.
3. The DUAL command cannot be used during ROUTE and vice versa.
4. The printer should be ready when you execute the command.

Sample Use

For a printed copy of all system/operator dialog, type: `DUAL` **ENTER**

To turn off the DUAL process, type: `DUAL (OFF)` **ENTER**

DUMP
Store a Program Into a Disk File

DUMP *file* (START = *aaaa*,END = *bbbb*,TRA = *cccc*,RELO = *dddd*)

file is the file specification

START = *aaaa* is the start address of memory block. *aaaa* must be a four-digit hexadecimal number greater than or equal to X'7000.'

END = *bbbb* is the end address of the memory block. *bbbb* must be a four-digit hexadecimal number.

TRA = *cccc* is the transfer address where execution starts when the program is loaded. *cccc* must be a four-digit hexadecimal number. If this option is omitted, the command will default to TRSDOS re-entry.

RELO = *dddd* is the start address for relocating or loading the program back into memory. *dddd* must be a four-digit hexadecimal number. If this option is omitted, no relocation will take place.

Note: Addresses must be hexadecimal form, without the X' ' notation. You must add the prefix "0" to any hex number which begins with a letter.

This command copies a machine-language program from memory into a program file. You can then load and execute the program at any time by entering the file name in the TRSDOS READY mode.

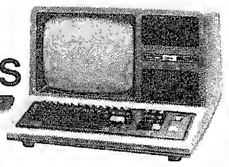
Examples

```
DUMP LISTER (START=7000,END=7100,TRA=7004)
```

Creates a program file named LISTER/CMD containing the program in memory locations X'7000' to X'7100'. When loaded, LISTER/CMD will occupy the same addresses, and TRSDOS will protect memory beginning at X'7000'. The program is executable for the TRSDOS READY mode.

```
DUMP PROG2 (START=7000,END=7F00,TRA=8010,RELO=8000)
```

Creates a program file named PROG2/CMD containing the program in addresses X'7000' to X'7F00'. When loaded, PROG2/CMD will reside from X'8000' to X'8F00'. Execution will start at X'8010'.



ERROR
Display Error Message

ERROR *number*

number is a decimal number for a TRSDOS error code.

This command displays a descriptive error message. For example, after receiving the message, `* * ERROR 47 * *` you may respond with the command: `ERROR 47` (ENTER) and TRSDOS will display the full error message.

For a complete list of error codes and messages, see the **Technical Information** section of this manual.

FORMS
Set Printer Parameters

FORMS (WIDTH = *w*, LINES = *l*)

WIDTH = *w* is the maximum number of characters per output line. If a line reaches this length, TRSDOS will insert a carriage return to force a new line. If this option is omitted, the current maximum width will be used. To disable the maximum line width feature, use WIDTH = 255. TRSDOS will not force new lines.

LINES = *l* is the number of lines per page. TRSDOS does not use this value. However, BASIC will use it in computing the necessary page displacement for execution or if LPRINT CHR$(12) is executed. If LINES = *l* is omitted, the current value is used.

This command lets you modify the printer forms control features of TRSDOS. The default values are:

Maximum line width: 132
Lines/page: 60

FORMS also sets the line count to 0.

Examples

If you are using 8½″-wide forms, you will probably want to set WIDTH = 80:

```
FORMS (WIDTH=80)
```

If you are using 14″-long forms, you may want to set LINES = 78.

```
FORMS (LINES=78)
```

This change will allow the BASIC statement, LPRINT CHR$(12), to advance a page by the correct number of lines.

Notes:

1. The WIDTH you specify is stored in RAM location 16427. The LINES you specify is stored in RAM location 16424.
2. The Printer *must be ready* when you execute this command.

FREE
Display Disk Allocation Map

FREE :*d* (PRT)

:*d* is the drive specification. If omitted, Drive 0 is used.

(PRT) tells TRSDOS to send the map to the Printer.
If omitted, TRSDOS sends the map to the Video Display only.

This command gives you a map of granule allocation on a diskette. (A granule, 1280 bytes, is the unit of space allocation.) It also shows the location of the directory and any flawed sectors.

When a diskette has been used extensively (file updates, files killed, extended, etc.), files often become segmented (dispersed or fragmented). This slows the access time, since the disk read/write mechanism must move back and forth across the diskette to read and write to a file.

FREE helps you determine just how segmented your disk files are. If you decide you'd like to re-organize a particular file to allow faster access, you can then COPY it onto a relatively ''clean'' diskette.

Examples

`FREE`

Displays a free space map of the diskette in Drive 0.

`FREE (PRT)`

Lists the free space for Drive 0 to the Printer.

`FREE :1 (PRT)`

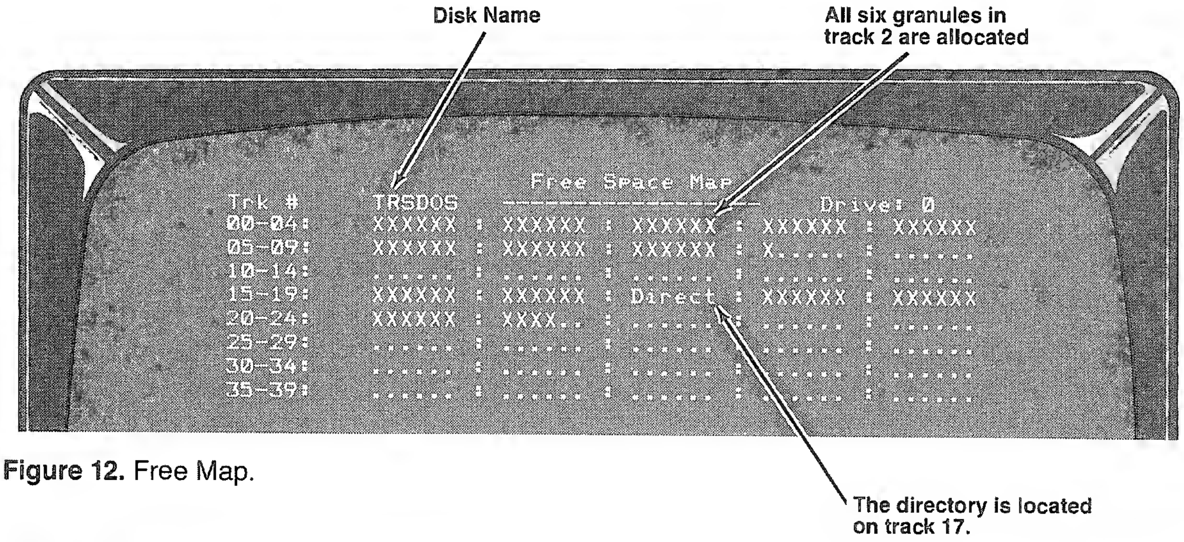
Lists the Drive 1 map to the Printer.

A Typical FREE Display

Four special symbols are used in the FREE map.

∘	Unused Granule
Direct	Directory Information
X	Allocated Granule
Flawed	Granule Contains a Flawed Sector (Unusable)

A typical free map display is shown in **Figure 12**.


Disk Name
All six granules in track 2 are allocated
The directory is located on track 17.

```
                        Free Space Map
Trk #      TRSDOS   ---------------------        Drive: 0
00-04:     XXXXXX : XXXXXX : XXXXXX : XXXXXX : XXXXXX
05-09:     XXXXXX : XXXXXX : XXXXXX : X..... : ......
10-14:     ...... : ...... : ...... : ...... : ......
15-19:     XXXXXX : XXXXXX : Direct : XXXXXX : XXXXXX
20-24:     XXXXXX : XXXX.. : ...... : ...... : ......
25-29:     ...... : ...... : ...... : ...... : ......
30-34:     ...... : ...... : ...... : ...... : ......
35-39:     ...... : ...... : ...... : ...... : ......
```

Figure 12. Free Map.

HELP
Explanation of TRSDOS Command

HELP *command*

command **is the specific TRSDOS command or subject on which you need help. If omitted or if an invalid subject is given, TRSDOS will list all available subjects.**

Example

If you type in the following: `HELP BACKUP` (ENTER) TRSDOS will respond with the syntax format, a definition of the command, and an explanation of the abbreviation.

`HELP SYNTAX` tells TRSDOS to explain the HELP descriptions.

KILL
Delete a File or Group of Files

Two syntaxes:

A) **KILL** ***file***
file **is a file specification**

B) **KILL** ***/ext:d***
/ext **is a file extension that** ***must*** **contain three characters.**
:d **is a drive specification. It** ***must*** **be provided.**

This command deletes one file or a group of files, depending on which form is used. Form A deletes the specified file. If no drive specification is given, TRSDOS deletes the file from the first diskette that contains it.

Form B deletes all files with a specified extension, regardless of the file name of each file. If no drive specification is given, the files will be deleted from the first drive that contains a matching file specification.

Examples

```
KILL TESTPROG/BAS
```

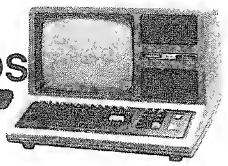
Deletes the named file from the first drive that contains it.

```
KILL JOBFILE/IDY.PASSWORD:1
```

Deletes the named file from Drive 1. The file has a password of PASSWORD.

```
KILL /BAS:0
```

Deletes from Drive 0 all files having the extension /BAS.



LIB
Display Library Commands

LIB

This command lists to the Display all the library commands. For help with a command, use HELP.

Example

```
LIB
```

LIST
List Contents of a File

LIST *file* (PRT,SLOW,ASCII)

> ***file*** **is the file specification.**
>
> **PRT tells TRSDOS to list to the Printer. If omitted, only the Video Display is used.**
>
> **SLOW tells TRSDOS to pause briefly after each record. If omitted, the listing is continuous.**
>
> **ASCII tells TRSDOS to list the file in ASCII format. If omitted, hexadecimal format is used.**

This routine lists the contents of a file. The listing shows both the hexadecimal contents and the ASCII characters corresponding to each value. For values outside the range (X'20', X'7F'), a period is displayed.

Use the ASCII option for text files and BASIC programs saved with the A option.

Note: Only ASCII codes X'00'-X'7F' are sent to the Printer. Bit 7 is always set to 0.

During the listing, press (@) to pause, (ENTER) to continue, or (BREAK) to exit.

Examples

```
LIST DATA/TXT (ASCII)
```

Lists the contents of DATA/TXT in ASCII format.

```
LIST FILE/A (SLOW)
```

Lists the contents of FILE/A, pausing after each record.

```
LIST PROGRAM/CMD (PRT)
```

Lists the file PROGRAM/CMD to the Printer.

LOAD
Load a Program File

LOAD *file*

file is a file specification for a file created by the DUMP command.

This command loads a machine-language program file into memory. After the file is loaded, TRSDOS returns to the TRSDOS READY mode.

You cannot use this command to load a BASIC program or any file created by BASIC. See the BASIC Reference Manual for instructions on loading BASIC programs.

Note: The file must load into the user area (X'7000'-TOP).

Examples

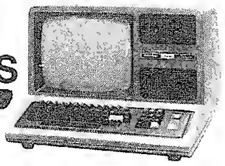
```
LOAD PAYROLL/PT1
```

Sample Use

Often several program modules must be loaded into memory for use by a master program. For example, suppose PAYROLL/PT1 and PAYROLL/PT2 are modules, and MENU is the master program. Then you could use the commands:

```
LOAD PAYROLL/PT1
LOAD PAYROLL/PT2
```

to get modules into memory, and then type: `MENU` to load and execute MENU.



MASTER
Set Master Read/Write Drive

MASTER (DRIVE = *a*)

> ***a*** **is the drive specification. If omitted Drive 0 is set as the master drive.**

This command allows you to assign a specified drive as the Master Read or Write drive in the system. When searching for a file, TRSDOS will start with the master drive.

If the file is not found on the specified drive, TRSDOS will continue searching on the next higher-numbered drive.

Example

After you enter the command: `MASTER (DRIVE=1)` Drive 1 becomes the master drive.

PATCH
Change the Contents of a Disk File

PATCH *file* (ADD = *aaaa*,FIND = *bb*,CHG = *cc*)

> ***file*** **is the file specification**
>
> **ADD = *aaaa* specifies the address at which the data is found. *aaaa* is a four-digit hexadecimal number.**
>
> **FIND = *bb* specifies the string you wish to find (or compare to). *bb* is a hexadecimal sequence.**
>
> **CHG = *cc* specifies the new contents for the byte(s). *cc* is a hexadecimal sequence.**
>
> **Note: This utility is for machine language programs *only*.**

This command lets you make minor corrections in any disk file, provided that:

1. You know the existing contents and location of the data you want to change.
2. You want to replace one string of code or data with another string of the same length.

You can use PATCH to make minor changes to your own machine-language programs; you won't have to change the source code, re-assemble it, and re-create the file.

Another application for PATCH is to allow you to implement any modifications to TRSDOS that may be supplied by Radio Shack. That way, you do not have to wait for a later release of the operating system.

Sample Use

Suppose you want to change seven bytes in a machine-language program file. First determine where the seven-byte sequence resides in RAM when the program is loaded. Then make sure your replacement string is the same length as that of the original string. For example, you might write down the information as follows:

File to be changed: VREAD

Start address: X'5280'

Sequence of code to be changed: X'CD2D25E5'

Replacement code: X'00000009'

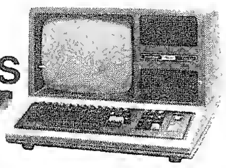
Then you could use the following command:

```
PATCH VREAD (ADD=5280,FIND=0CD2D25E5,CHG=00000009)
```

PAUSE
Pause Execution for Operator Action

PAUSE *message*

> ***message*** **is the message to be displayed during the pause execution. This is optional. If omitted, PAUSE will be displayed by itself.**



This command is intended for use inside a DO file so TRSDOS can print a message or reminder.

To continue after the pause, TRSDOS prompts you with the message:

```
PRESS <ENTER> TO CONTINUE
```

Example

```
PAUSE INSERT DISKETTE #21
```

TRSDOS displays PAUSE, next the message and then prompts you to press (ENTER) to continue execution.

```
PAUSE
PRESS <ENTER> TO CONTINUE
```

TRSDOS displays PAUSE and then next prompts you to press (ENTER) to continue. See BUILD and DO for sample uses.

PROT
Use or Change a Diskette's Master Password

PROT :*d* (PW,LOCK)

> **:*d* is an optional drive specification. If omitted, Drive 0 is used.**
>
> **PW tells TRSDOS you want to change the master password.**
>
> **LOCK tells TRSDOS to assign the master password to all unprotected user files. If omitted, the unprotected files remain unprotected.**

PROT lets you use the master password to protect all unprotected files at once, or to change the master password.

The master password will be needed to BACKUP the diskette, so be sure to remember it!

Note: The master password on the TRSDOS factory-release diskette is PASSWORD.

Examples

```
PROT :0 (PW)
```

Tells TRSDOS to change the master password on the Drive 0 diskette. TRSDOS will prompt you first for the old master password, then for the new master password.

```
PROT :1 (LOCK)
```

Tells TRSDOS to assign the master password to all unprotected user files. TRSDOS will first prompt you for the master password.

PURGE
Delete Files

PURGE :*d* (file-type)

:*d* is the drive which contains the disk to be purged.

file-type must be one of the following:

SYS	All System and User files (no Invisible)
INV	All Invisible and User files (No System)
ALL	All files on disk (User, System, Invisible)

If *file-type* is omitted, TRSDOS defaults to User files.

This command allows quick deletion of files from a particular diskette. To use PURGE, you must know the diskette's master password. (TRSDOS System diskettes are supplied with the password PASSWORD.)

When the command is entered, TRSDOS will ask for the diskette's password. Type in up to eight characters. Press (ENTER) if you typed fewer than eight characters. The System will then display user filenames one at a time, prompting you to KILL or leave each file.

Example

```
PURGE :1
```

TRSDOS will purge user files from Drive 1. This would include BASIC programs.

```
PURGE :1 (INV)
```

TRSDOS will purge all invisible files in Drive 1.

Note: System diskettes contain some files which are not shown in any of the directory listings. You may delete these files with a special form of PURGE:

PURGE * *:d (file-type)*

The asterisk tells TRSDOS to ask you if you want to delete the System files. If you do delete them, the diskette becomes a data diskette and may only be used in Drive 1, 2 or 3.

The other parts of this command are as explained previously. However, be sure to do the PURGE using Drive 1, 2 or 3, since the diskette will become ''non-system'' *during* the PURGE.

RELO
Change Where Program Loads into Memory

RELO *file* (ADD = *aaaa*)

file is the file specification.

ADD = *aaaa* specifies the new load address. *aaaa* is a four-digit hexadecimal number referring to an address in the user memory. *aaaa* must be in the user area of RAM.

This command allows you to change the address at which the program loads into memory. It does not change the program itself.

Note: This command may be useful in conjunction with DUMP.

Example

```
RELO PROGRAM/CMD (ADD=6578)
```

TRSDOS will load the program PROGRAM/CMD at the new memory address of 6578.

RENAME
Rename a File

RENAME *oldname newname*

oldname is the old file name.

newname is the new file name.

The file name may include a drive specification and or password.

The new file name should not include a drive specification or password.

This command lets you rename a file or program. Only the name/extension is changed; the data in the file and its physical location on the diskette are unaffected.

RENAME cannot be used to change a file's password protection. Use ATTRIB to do that.

RENAME also checks to see that the intended new name does not duplicate a filename currently on the same diskette. If it does, the command is cancelled and an error message is displayed.

Examples

```
RENAME MATHPAK MATHPAK/BAS
```

Tells TRSDOS to add the extension to the filename.

```
RENAME ABCDE/DAT ABCDEF/DAT
```

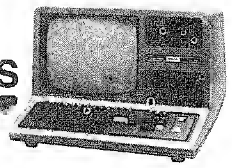
Tells TRSDOS to change the filename only.

```
RENAME PAYROLL1/TXT.GSR PAYROLL2/TXT
```

Tells TRSDOS to change the filename; the password is retained automatically.

```
RENAME FILE1:3 FILE2
```

Tells TRSDOS to change the filename of the file on Drive 3.



ROUTE
Routing I/O Devices

ROUTE (SOURCE=*aa*,DESTIN=*bb*)

SOURCE=*aa* specifies the source I/O device.

DESTIN=*bb* specifies the destination I/O device.

aa and *bb* may be any meaningful combination of the following two-letter abbreviations:

DO	(Display)
PR	(Printer)
KB	(Keyboard)
RI	(RS-232 Input)
RO	(RS-232 Output)

If the SOURCE and DESTINATION options are omitted, TRSDOS resets I/O Drivers to their original I/O routes. The SOURCE and DESTINATION devices must both be output or both be input.

This command allows you to route I/O devices automatically. For example, TRSDOS can route information from the Printer (PR) to the Display (DO).

Note: ROUTE cannot be used in conjunction with the DUAL command.

Examples

```
ROUTE (SOURCE=PR,DESTIN=DO)
```

TRSDOS will route your Printer output to the Display.

```
ROUTE
```

I/O drivers are returned to their original state.

For further details on routing I/O see "Routing Input/Output" in the Model III Manual.

SETCOM
Set Up RS-232-C Communications

SETCOM (OFF,WORD = *a*,BAUD = *b*,STOP = *c*,PARITY = *d*,*mode*)

OFF turns RS-232-C off.

WORD = *a* is the number of bit/byte desired. *a* must be either 5, 6, 7, or 8, depending on your needs. If omitted, the word length is not changed.

BAUD = *b* specifies the baud rate. *b* must be a decimal number between 50 and 9600. If omitted, the baud rate is not changed.

STOP = *c* specifies the number of stop bits. *c* must be either 1 or 2. If omitted, stop bits are not changed.

PARITY = *d* determines whether the parity is odd, even, or none. *d* must be 3 (none), 1 (odd), or 2 (even). If omitted, parity is not changed.

mode type either WAIT or NOWAIT

Options must be entered in the order shown.

If all the options are omitted, TRSDOS displays the current RS-232-C settings.

This command initializes RS-232-C communications via the serial channel. Before executing it, you should connect the communications device to the Model III.

See the Model III Operation Manual for a description of RS-232-C signals used.

See **Using the RS-232-C Interface** in the Model III Manual for further details.

Examples

```
SETCOM (WORD=7,BAUD=300,STOP=1,PARITY=3,WAIT)
```

This would set the RS-232-C to seven bit words, 300 baud, one stop bit, no parity, and place it in the WAIT mode.

```
SETCOM
```

The command without specifications will display the current settings.

The following program will allow you to use your Computer as a terminal. For further information, refer to the Operation section of your Model III Operation Manual.

Note: This program executes at 300 Baud.

```
5   DEFINT A-Z           'INTEGER VARIABLE FOR SPEED
10  POKE 16890, 0        'DON'T WAIT FOR SERIAL I/O
15  POKE 16888, (5*16)+5 'TX/RCV AT BAUD RATE 300
20  DEFUSRO = &HOO5A: REM SET UP CALL TO $RSINIT
40  X = USR0(0)
60  DEFUSRI = &H0050
65  DEFUSR2 = &H0055
70  CI = 16872           'CHARACTER INPUT BUFFER
80  CO = 16880           'CHARACTER OUTPUT BUFFER
90  ' CHECK FOR SERIAL INPUT
110 X = USR1(0)          'CALL $RSRCV
120 C$ = CHR$(PEEK(CI))  'LOOK AT INPUT BUFFER
130 PRINT C$             'IF C = 0, NOTHING HAPPENS
140 ' CHECK FOR KEYBOARD INPUT
150 C$ = INKEY$
160 IF C$ = "" THEN 110  'NO KEY, SO GO CHECK SERIAL
165 PRINT C$;            'SELF ECHO
170 POKE CO, ASC(C$)     'PUT CHARACTER INTO OUTPUT BUFFER
190 X = USR2(0)          'CALL $RSTX
200 GOTO 110             'GO CHECK SERIAL INPUT
```

TAPE
Tape/Disk Transfer

TAPE (S = *source*, D = *destination*)

source **and** ***destination*** **are abbreviations for the storage devices to be used:**

T	**Tape**
D	**Disk**
R	**Random access memory**

Note: TAPE can only be used with machine-language programs. BASIC programs must be CLOADed and CSAVEed.

This command transfers Z-80 machine-language programs from one storage device to another. The following transfers are possible:

- Tape to disk

- Disk to tape
- Tape to RAM

Examples

```
TAPE (S=T,D=D)
```

Starts a tape-to-disk transfer. TRSDOS will prompt you `CASS?`. Select the desired baud rate (`H` for high, `L` for low). TRSDOS will then prompt you to press (ENTER) when the recorder is ready to play to the Computer. When you press (ENTER), the tape will begin loading.

Note: If no asterisks flash, the recorder volume may need adjustment or the baud rate setting may be incorrect.

TRSDOS will read the file name from the tape and use that name for the disk file. It will copy the program to the first write-enabled diskette, starting with the master drive (see MASTER).

```
TAPE (S=D,D=T)
```

Starts a disk-to-tape transfer. TRSDOS will prompt you for the desired cassette baud rate, then for the diskette file specification. Then it will tell you to press (ENTER) when the cassette recorder is ready to record from the Computer.

```
TAPE (S=T,D=R)
```

Starts a tape-to-RAM transfer. TRSDOS will prompt you for the cassette baud rate, and will tell you to press (ENTER) when the recorder is ready to play to the Computer. After loading the program, TRSDOS will begin execution at the transfer address specified on the tape.

TIME
Reset or Get the Time

TIME *hh:mm:ss*

***hh:mm:ss* specifies the hour *hh*, minute *mm*, and second *ss*.**

Each must be a two-digit decimal number between the following ranges:

hh	0-23
mm	0-59
ss	0-59

If *hh:mm:ss* is given, TRSDOS resets the time.

If *hh:mm:ss* is not given, TRSDDS displays the current time.

This command lets you reset or display the time.

Time uses a 24-hour clock. For example, 1:00 P.M. is displayed as 13:00.

You initially set the time when TRSDOS is started up. After that, TRSDOS updates the time automatically, using its built-in clock.

When you request the time, TRSDOS displays it in this format: `14:15:31` for 2:15:31 P.M.

Examples

```
TIME
```

Displays the current time.

```
TIME 13:20:00
```

Resets the time to 1:20:00 P.M.

Note: If the clock is allowed to run past 23:59:59, it will re-cycle to zero, the date will be incremented, and the clock will continue to run.

WP
Write-Protect Via Software

WP (DRIVE = *d*)

_d_ specifies the disk drive to be protected. If omitted, all drives will be unprotected.

Diskettes can be protected from being overwritten by this command. It is a software write-protect rather than a hardware write-protect (such as the write-protect tab on the diskette).

Only one drive may be protected at a time.

To unprotect a drive, making it accessible to writing, simply enter the command WP without options or with a different drive number specified. The WP command will not override a write-protect tab.

Examples

```
WP (DRIVE=1)
```

TRSDOS will write-protect the disk in Drive 1.

```
WP
```

TRSDOS will eliminate write-protection on all drives.

TRSDOS Utility Commands

BACKUP
Create an Exact Copy of an Original Disk

BACKUP *:source :destination*

:source specifies the drive containing the original diskette. If omitted, TRSDOS will prompt you for this information.

:destination specifies the drive containing the diskette to receive the copy. If omitted, TRSDOS will prompt for it.

:source and *:destination* may reference the same drive.

BACKUP copies the contents of the source disk to the destination disk. This gives you a "safe" copy of the disk. Always keep an extra copy of data or programs you have stored on your disks.

Note: Both source and destination diskettes must be write-enabled.

TRSDOS will prompt you at each step after you type: `BACKUP`

If you omitted the source/destination-drive numbers, TRSDOS will begin with the prompts: `SOURCE DRIVE NUMBER`.

Type in the number of the drive that contains the source diskette and press (ENTER).

```
DESTINATION DRIVE NUMBER?
```

Type in the number of the drive that will contain the destination diskette and press (ENTER).

```
SOURCE DISK MASTER PASSWORD?
```

Type in the password assigned to your source diskette.

```
DISK CONTAINS DATA, USE DISK OR NOT?
```

Type in `Y` (Yes) or `N` (No).

```
DO YOU WISH TO RE-FORMAT THE DISK?
```

Type in `Y` (Yes) or `N` (No).

If you specified the source/destination drives, TRSDOS will request the PASSWORD, skipping the first two steps.

TRSDOS will then take charge of formatting and verifying the destination disk as well as letting you know if there are any errors or flawed tracks.

CONVERT
Model I to Model III File Conversion Utility

```
CONVERT
```

Model I formatted diskettes cannot be used in the Model III Disk System. However, the CONVERT utility can read a Model I diskette and copy its non-system files onto a Model III TRSDOS diskette. This diskette may then be used normally in the Model III Disk System. The original Model I diskette may still be used in a Model I Disk System, since it is unchanged by CONVERT.

CONVERT does not convert or change data; it converts the *file storage format*. For this reason, Model I Disk BASIC programs may require slight changes before they will run properly in the Model III Disk System. Model I machine-language programs may require major or minor changes before they will run in the Model III Disk System. You may make these changes on the Model I diskette before using CONVERT, or on the Model III diskette containing the converted files.

For hints on program conversion, see:

- **Technical Information** in this manual
- Technical Information in the Model III Manual
- The manual, *Instructions for Converting Specified Model I Programs for use on TRS-80 Model III*.

Drive Usage

In two-drive systems, the files must be copied onto a Model III system diskette in Drive 0; in three- or four-drive systems, the files may be copied onto a data diskette in Drive 1, 2 or 3.

During the conversion process, the Model I diskette is referred to as the "source"; the Model III diskette, the "destination." The source diskette cannot be in the same drive as that of the destination diskette.

Password Protection

CONVERT is designed to preserve the password security of each file that it transfers. To accomplish this and still allow the copying of protected files, CONVERT follows different procedures depending on the access and update passwords on each file.

In the simplest case, a file has blank access and update passwords. The copied file will be given blank passwords. (If you have a Model I Disk System with TRSDOS 2.3, you may use the PROT command to remove all passwords from all files. This will simplify the CONVERT process. Do this on the Model I system before you attempt to convert to Model III.)

In another case, the access and passwords are different. If the access password is blank and the update is not, then TRSDOS will prompt you for the update password. If you know the update password, type it in. The file will be copied with access and update passwords set to the old update password. If you don't know the update password, simply press (ENTER). The file will be copied with the access password set to blanks and the update password set to an unknown value.

If the access and update passwords are not blank and they are not the same, TRSDOS will not copy the file, but will print the message, `FILE SKIPPED`, and continue with the next file in the source directory.

Sample Use

Get the Model I diskette ready. If you have a Model I Disk System with TRSDOS 2.3, try to remove all passwords from all your files. This will prevent any problems with passwords. The password protection may be restored with the Model III ATTRIB or PROT commands after the conversion is complete.

Using the Model III Disk System, you must always have a Model III TRSDOS diskette in Drive 0. TRSDOS READY should be displayed. Type: `CONVERT` (ENTER).

The program will ask, `SOURCE DRIVE?`. Type in the number of the drive containing the Model I diskette, and press (ENTER). Then the program will ask, `DESTINATION DRIVE?`. Type in the drive number and press (ENTER). In two-drive systems, you must use Drive 0 as the destination.

During the conversion process, the name of each file will be displayed as it is copied. If password information is needed, TRSDOS will prompt you for it. If you know the update password, type it in and press (ENTER). The file will be copied and given the same update password. If you do not know the update password, simply press (ENTER), in which case the file will be copied and given an unknown update password.

If a file name on the source diskette is already used on the destination diskette, TRSDOS will print this message: `FILE EXISTS, USE IT?`. If you type `Y`, TRSDOS will copy the file. The previous contents of the Model III file will be lost. If you type `N`, TRSDOS will skip the file, and get the next one from the Model I diskette.

FORMAT
Prepare a Data Diskette

FORMAT ***:d***

> ***:d*** **specifies the disk drive which contains the diskette to be formatted. If** ***:d*** **is omitted, TRSDOS will prompt you for this information.**

This command lets you prepare data diskettes (either new or disks which contain undesired data or programs), leaving a maximum amount of space for your program and data files.

Note: Data diskettes may only be used in Drives 1, 2, and 3 except during a BACKUP or FORMAT.

FORMAT takes a blank (new or magnetically erased) diskette, records track/sector boundaries on it, then initializes it with and creates a directory.

When FORMAT detects a non-blank diskette, it will display a warning message:

```
DISK CONTAINS DATA, USE DISK OR NOT?
```

Type `Y` (Yes) and press (ENTER) if you do want to reformat, `N` (No) and press (ENTER) if you want to save the disk information.

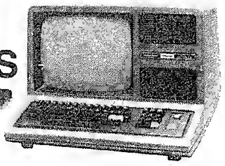
FORMAT will lock out any defective tracks to prevent data from being lost in these areas.

If you begin to get READ errors during access, reformat the disk.

Example

```
FORMAT :1
```

After you are prompted for `DISKETTE NAME?` and `MASTER PASSWORD?`, TRSDOS will format Drive 1.



HERZ50
Set Up for 50 Hz AC power (non-USA users)

DO HERZ50

starts the utility to change the system for 50 Hz operation. HERZ50 is a DO-file.

This utility is provided for customers in areas where the AC power is 50 rather than 60 Hz. It should not be used by any other customers. HERZ50 simply places a patch on the diskette that changes the clock speed for 50 Hz users.

HERZ50 is a DO-file that makes a change in the software of TRSDOS. Only the Drive 0 diskette is changed. Be sure it is write-enabled before you start the DO-file. Once the HERZ50 change is done, it will remain in effect for that diskette.

To perform the change, type:

```
DO HERZ50
```

Once the change has been made, you will need to reset the system to put the change into effect. This loads the new software into RAM.

LPC
Line Printer Control

LPC

The LPC utility program allows TRSDOS to ignore multiple carriage return commands. Without LPC, a top-of-form (LPRINT CHR$(12)) command will add an extra carriage return/line feed each time it is executed. Also, LPC masks the high bit of each data byte, allowing you to send certain intercepted codes to the printer. For instance, the BASIC statement LPRINT CHR$(140) will send code 140-128 = 12 (LPRINT CHR$(12)) to the Printer.

The printers that require LPC are:

Line Printer III (26-1156)
Line Printer VI (26-1166)
Daisy Wheel WP50 (26-1157)
Qume Daisy Wheel (26-1157A)
Daisy Wheel II (26-1158)

and all future printers.

Printers that do not require LPC:

26-1150, 1152, 1153, 1154, 1159, and the A version of LPIII (26-1156A).

You must load the LPC program before you load an application program. The easiest way to do this is to copy LPC onto your data/program diskette and then use the AUTO command to load LPC automatically each time you use the system. For instance, type:

```
COPY LPC:1 :0 (ENTER)
```

Then, to make LPC an AUTO command on the diskette, type:

```
AUTO LPC/CMD (ENTER)
```

Whenever you use your program diskette, LPC will automatically load into memory and you can use the program as usual.

LPC locates into the highest available memory. There is no need to set MEMORY SIZE to protect LPC. It "hides" itself. However, you still need to set memory if required by your application program. LPC will be killed if the CLEAR command is used.

Warning: Once the LPC utility program is loaded and installed, you should not reload it except after a reset. Reloading re-installs the program and uses up more space each time! LPC will not execute if the Printer has been routed elsewhere. Also, if LPC has been executed and then the Printer is routed elsewhere, the original printer driver will regain control after the routing.

MEMTEST
Test Memory

```
MEMTEST
```

This program tests your Model III's memory (read only and random access). In TRSDOS READY, just type MEMTEST and press (ENTER).

The program automatically tests all memory locations, no matter what memory size you have. First it checks read only memory (ROM); if everything is okay, it automatically goes on and checks random access memory (RAM). If all RAM checks out okay, the program continues.

If the program detects a ROM or RAM error, it will display a detailed message. Repeat the test to make sure it is a valid error condition. Write the message down and contact your nearest Radio Shack for assistance.

Note: MEMTEST changes the entire contents of RAM. Before running it, be sure you have saved any valuable code you may have in RAM.

XFERSYS
Transfer System Files

```
XFERSYS
```

XFERSYS lets you upgrade your version of Model III TRSDOS by copying all system files from a new release diskette (source) onto a previously released diskette (destination) (i.e., version 1.2 to version 1.3, etc.).

System files which already exist on the destination diskette are replaced by those from the source diskette. Files which do not exist on the destination diskette are added. User files (program and data) are unaffected.

Steps to upgrade a diskette

Make backup copies of all diskettes to be upgraded. This is an important precautionary step. These backup copies should be kept until the upgrading process is complete and confirmed.

Note: Both source and destination diskettes must be write-enabled.

Insert the new release of TRSDOS into Drive 0 and press the RESET button. Then type `XFERSYS` **(ENTER)**.

After the program heading appears, TRSDOS will prompt you with `DISKETTE TO CONVERT READY IN DRIVE 1 (Y/Q)?`. Type `Y` (yes) or `Q` (quit) and press **(ENTER)**.

The upgrading process will then take place. When the process is complete, TRSDOS will tell you so and take you back to TRSDOS READY.

Note: If an error occurs, including your trying to upgrade a non-system diskette, the operation will be cancelled and take you back to TRSDOS READY.

Technical Information

Contents of This Section:

Disk Organization

Each TRSDOS system diskette contains a TRSDOS system, a utility command library, and a file directory.

Each diskette is single-sided and has 40 tracks of information. Each track contains 18 sectors of 256 bytes each.

Normally, data read/write operations may be initiated only at sector boundaries, and must consist of exactly 256 bytes. However, TRSDOS allows the user to have maximum flexibility with minimal effort by automatically blocking and deblocking all file accesses to user-specified logical record lengths, even if this requires "spanning" of two sectors.

The system disk file structure allows maximum use of disk file space by automatically segmenting files across a diskette in several small pieces. These pieces are correlated into one logically contiguous file by the system without your needing to know the physical file location. This structure eliminates time-consuming disk-packing operations.

File Structure

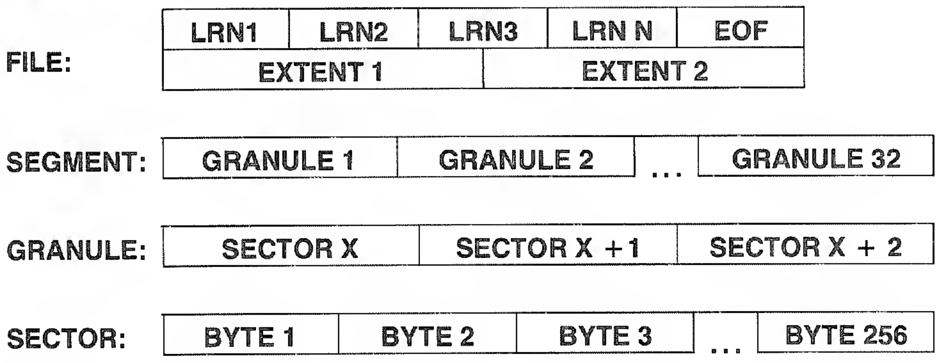
A TRSDOS file is composed of one or more segments of storage space. Each segment consists of from one to 32 physically contiguous granules of storage. A granule is the minimum allocatable unit of storage, and consists of three sectors (768 bytes).

1 GRANULE IS 3 SEGMENTS
FILES GO UP IN GRANULE STEPS.

Since a file is always lengthened by granules, a small amount of free storage is generally present at the end of every file. This free storage allows minor file additions to be made in space which is physically contiguous to the file.

Every time a disk file is extended (either initialized or lengthened), extra granules may be allocated to that file, depending on the file's accumulated length, diskette space, saturation, etc. These extra granules, along with all granules after the one containing the file's EOF mark, are recovered and returned to the system when the file is closed.

A TRSDOS file



LRN: Logical Record Number, used to specify an individual, user-defined logical record. Such a logical record is the smallest unit of information which can be addressed during disk input/output (a physical record is the unit which is actually read from or written to disk).

File: A group of logical records; the largest unit of information which can be addressed by a TRSDOS command.

Sector: A physical record, composed of 256 contiguous bytes.

Granule: The minimum allocatable unit of storage for any file.

Extent: One contiguous allocation of granules.

System Routines for Assembly-Language I/O

This information is provided for customers who wish to write their own assembly level I/O routines. An explanation of the calling sequence and parameters for each necessary I/O routine is given. A knowledge of Z-80 machine code is assumed.

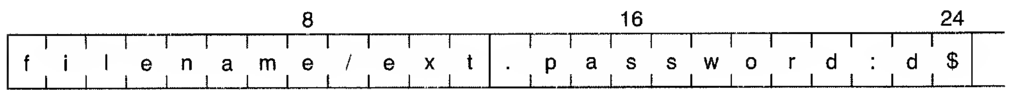
The following notations are standard in this section:

(HL) = *xxxx*	Registers HL contain the address of (point to) *xxxx* in machine format. (If address of *xxxx* = 34B2H then the values in the registers are: H = 34; L = B2). Other register pairs will also be indicated this way.
A = *xx*	Register A contains the numeric value of *xx* in binary form. Register A is used to return the TRSDOS error code for I/O calls. A complete list of error codes and their meanings appears at the end of this chapter. Other registers will also be indicated this way.

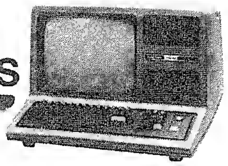
Z=OK	Zero flag is set (OK) if successful return from the system routines.
X'*nnnn*' or *nnnn*H	Hard RAM address in hex notation (e.g., 402D is X'402D').
LRL	Logical Record Length, 1-255 bytes only. You can define records any length you wish up to 255 bytes maximum. A length of zero is a special case for physical records only, and indicates the LRL=256 bytes.
BUFFER	256 user-designated bytes in RAM for TRSDOS to read sectors from or write sectors into. If LRL=0, this area is the responsibility of the user to manage before and after I/O. TRSDOS manages this area if LRL is between 1 and 255 bytes. Do not alter this area when using logical record processing.
UREC	(User Record) The address of the contiguous RAM byte-string assigned by the user as his logical record area. Its length must be equal to LRL. It is a different area from BUFFER.
LSB/MSB	Least-significant byte followed by most significant byte. This is the standard Z-80 format for addresses.
$*name*	The "$" is prefixed to all system locations and call routines, so they will not be confused with TRSDOS commands or utilities. For example, $OPEN.

DCB before $OPEN and after $CLOSE

The DCB (device control block) is defined as 50 contiguous bytes of RAM designated by the user. Before $OPEN and after $CLOSE, it is a left-justified, compressed (no spaces) ASCII string, as in a standard TRSDOS filespec. The string is terminated with a carriage return.



Notes: /ext, .password, and :d are optional.
$ stands for a carriage return (X'OD')



DCB while $OPEN

Address	Length	Explanation
DCB+0	3	Reserved
+3	2	Physical Buffer address (LSB/MSB)
+5	1	Offset to delimiter at end of current record
+6	1	File drive number residence
+7	1	Reserved
+8	1	EOF offset of last delimiter in last physical record
+9	1	LRL (logical record length)
+10	2	NRN (next record # — $OPEN sets = X'0000' — LSB/MSB)
+12	2	ERN (ending record # — (last in file) LSB/MSB)
+14	50	Reserved

NRN Next Record Number defines which record is to be read or written by the next system call for $READ or $WRITE. It is automatically incremented by one after each system call. In order to process random files, use the $POSN call to direct TRSDOS to the record you wish to transfer next.

ERN Ending Record Number is the last record number currently in the file. It is put into the directory at $CLOSE time, so if it is expected to be correct, the user *must* close his files after adding records to a file. This value may also be used to position to end of file so that new records may be added to the end of the file. To position to the end of file use a call to $POSN with a record number of ERN+1. $POSN is described later.

Physical and Logical Records in TRSDOS

A physical record is defined as one sector of disk. One sector of disk contains 256 user data bytes. The artificial term ''granule'' is defined to be 3 sectors of disk space. There are 6 granules on each of the 40 tracks on the disk. A granule is the least amount of space allocated by TRSDOS. For programming purposes, the physical records in a file are numbered from 0 to N. The largest record number (N) in a file will then be 3 times the number of granules allocated minus one ((3*G) − 1). All TRSDOS granule allocations are made as needed *at the time of write, not when the file is created.*

Bytes	Sectors	Granules	Tracks	Disk
256	1	—	—	—
768	3	1	—	—
4608	18	6	1	—
184320	720	240	40	1

Disk Space Table: For each 5¼″ Disk Drive

A logical record is defined by the user of TRSDOS. It may be anywhere from 1 to 255 bytes in length. Once a file is opened with a specific LRL (Logical Record

Length), the length is fixed until the file is closed. To change a file's LRL, you must CLOSE it and re-OPEN it with the new LRL.

Each opening of the file sets a single, fixed record-length. TRSDOS will "block" logical records into (or from) one physical record for maximum space utilization on the disk.

Blocking is putting more than one logical record into one physical record. For instance, four 64-byte logical records will fit into one 256-byte physical record. A logical record may be broken into two parts by TRSDOS in order to fill the last portion of one physical record entirely before beginning to use the next physical record (i.e. records are spanned). This occurs when the physical record length is not an even multiple of the logical record length.

If the user wishes to do his own blocking, he may specify a logical record length of 0 bytes at the time of INIT/OPEN and must himself manage the contents of the physical record buffer area of 256 bytes. TRSDOS *will not move* a logical record for the user if LRL = 0; in this particular case it will only read/write the physical record to/from the buffer. Once control is shifted to your program, you will have about 20 bytes of stack size left.

Fundamental TRSDOS I/O Calls

There are 17 fundamental TRSDOS routines involved in handling file I/O. These are:

$BACKSPACE
$CLOSE
$DIVIDE
$DMULT
$FILPTR
$INIT
$KILL
$OPEN
$POSEOF
$POSN
$PUTEXT
$RAMDIR
$READ
$REWIND
$SYNTAX
$VERF
$WRITE

The detailed calling sequences and discussions for each of these routines follow. Note that *all* of these system calls use register F and do not restore its value before return. In order to apply this data properly, you should read through all of these descriptions and clear up all of the points that are not obvious to you by using other reference materials. If you are successful in doing this you will find that TRSDOS is a workable tool for your programming ideas.

$INIT — 17440/X'4420'

$INIT is provided as an entry point to TRSDOS which will create a new file entry in the directory and open the DCB for this file. $INIT scans the directory for the filespec name given in the DCB. If the filespec name is found, $INIT simply opens the file for use. If the name is not found, a new file is created with the filespec name.

Entry Conditions

(HL) = BUFFER (see beginning of this section for notation)
(DE) = DCB
B = LRL
CALL $INIT

Exit Conditions

IY = changed
Z = OK
C carry flag is ON if a new file was created
A = TRSDOS error code. (Error codes listed at end of this chapter)

$OPEN—17444/X'4424'

$OPEN provides a way to open the DCB of a file which already exists in the directory. The DCB *must* contain the filespec of the file to be opened *before* entry to $OPEN.

Entry Conditions

(HL) = BUFFER
(DE) = DCB
B = LRL
CALL $OPEN

Exit Conditions

Z = OK
Z = 0 if file does not exist.
A = TRSDOS error code.
IY = changed

$POSN—17474/X'4442'

$POSN positions a file to read or write a randomly selected logical record. Since it deals with logical records, the proper computation is done to locate which physical record(s) contain the data. Following a $POSN with a $READ or $WRITE will transfer the record to/from RAM.

Note that positioning to logical record zero sets the file to read the first logical record in the file. To position to end of file in order to add new records onto the end, use the record number ERN + 1.

Entry Conditions

(DE) = DCB (must have been opened previously)
BC = Logical record number to position for.
CALL $POSN

Exit Conditions

Z = OK
A = TRSDOS error code.

$READ—17462/X'4436'

If LRLO, $READ transfers the logical record whose number is in the DCB as NRN into the RAM area addressed as UREC for the length LRL as defined at open time. The record comes from "BUFFER" defined at open time. If TRSDOS must read a new physical record to satisfy the request, it will do so. "Spanned" logical records will be re-assembled as necessary. $READ automatically increments NRN by 1 in the DCB after the transfer is completed. $INIT/OPEN set NRN = X'0000' in order to read the first record with the first READ.

If LRL = 0, $READ transfers one physical record into BUFFER, defined at open time, from the disk file. Registers HL are ignored. $READ increments NRN as above.

Entry Conditions

(HL) = UREC if LRL is not zero. Unused if LRL = 0.
(DE) = DCB
CALL $READ

Exit Conditions

Z = OK
A = TRSDOS error code. (EOF = X'1C' or X'1D')
(see errors 28,29 for EOF or NRF)

$WRITE—17465/X'4439'

If LRLO, $WRITE transfers the one logical record from the RAM area addressed as UREC for the length LRL as defined at open time. The record goes into the BUFFER which was defined at open time. If TRSDOS must write a physical record in order to satisfy the request, it will do so. ``Spanning'' will be handled by TRSDOS as necessary. At $INIT $OPEN time the DCB value of NRN is set to X'0000' so that the first record will be written. After each logical record is transferred, the NRN value in the DCB will be incremented by 1.

If LRL=0, $WRITE transfers one physical record from BUFFER into the disk file using the NRN in the DCB. BUFFER is defined at $INIT/OPEN time only. The DCB value NRN is updated as above, after the WRITE.

Entry Conditions

(HL) = UREC if LRL is not zero. Unused if LRL=0
DE = DCB
CALL $WRITE

Exit Conditions

Z = OK
A = TRSDOS error code.

$VERF—17468/X'443C'

The only difference between $VERF and $WRITE is that $VERF writes one physical record to disk and then reads it back into a special TRSDOS RAM area not defined by the user. This special area and the original write buffer are then compared byte by byte to assure that the record was successfully written.

Entry Conditions

(HL) = Same as $WRITE above.
(DE) = DCB
CALL $VERF

Exit Conditions

Z = OK
A = TRSDOS error code.

$PUTEXT—17483/X'444B'

This routine will add an extension to a filename if an extension does not already exist. An extension to a filename may be useful for identifying the type of data in the file.

Entry Conditions

(DE) = DCB
(HL) = The extension to be added to the file
CALL $PUTEXT

Exit Conditions

None

$BKSPC—17477/X'4445'

This routine positions the file record pointer to the previous record.

Entry Conditions

(DE) = DCB
CALL $BKSPC

Exit Conditions

Z = Valid position
NZ = Invalid position in file

$REWIND—17471/X'443F'

Point to the beginning of the file. This routine positions the file pointer to the first record in the file. This is useful when the same file must be processed more than once.

Entry Conditions

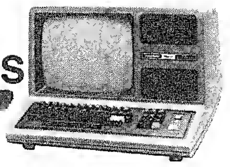
(DE) = DCB

Exit Conditions

Z = Good file specification
NZ = Bad file specification
CALL $REWIND

$POSEOF—17480/X'4448'

Point to the end of file. This routine positions the file pointer to the last record in the file. This may be used to extend a sequential access file.



Entry Conditions

(DE) = DCB
CALL $POSEOF

Exit Conditions

Z = Good file specification
NZ = Bad file specification

$SYNTAX—17436/X'441C'

Move a file specification to DCB. This routine takes a file specification and checks it for validity and moves it to a DCB so that the file may be opened.

Entry Conditions

(HL) = Filename
(DE) = DCB
CALL $SYNTAX

Exit Conditions

Z = Good file specification
NZ = Bad file specification

$DIVIDE—17489/X'4451'

The divide routine takes a 16-bit dividend and an eight-bit divisor. After division, the quotient replaces the 16-bit dividend and the remainder replaces the eight-bit divisor.

Entry Conditions

HL = Dividend
A = Divisor
CALL $DIVIDE

Exit Conditions

HL = Quotient
A = Remainder (0 indicates no remainder).

$DMULT — 17486/X'444E'

The multiply routine uses a 16-bit multiplicand and an eight-bit multiplier. After multiplication takes place, the product replaces the 16-bit multiplicand.

Entry Conditions

HL = Multiplicand
A = Multiplier
CALL $DMULT

Exit Conditions

H = High order byte
L = Middle order byte
A = Low order byte

H	L	A
High	Middle	Low

$RAMDIR — 17040/X'4290'

This routine allows you to examine a diskette directory (one entry or the entire directory) or the diskette's free space. The information is written into a user specified RAM buffer.

Only non-system files will be included in the RAM directory.

Entry Conditions

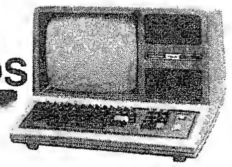
HL = RAM Buffer. If C=0, size = 1761 [max #*22+1]. If C=1 to 96, size =22. If C=255, size = 64.
B = Specified drive number
C = Function switch:

Contents of C	Results
0	Gets entire directory into RAM. (See RAM Directory Format).
1-96	Gets one specified directory record into RAM, if it exists. (See RAM Directory Format).
255	Gets free-space information (See RAM Directory Format).

CALL $RAMDIR

Exit Conditions

NZ = Error occurred.
Z = No error. (HL) = directory or free-space information.



RAM Directory Format

The directory is made up of records, one per file. All values are hexadecimal. Each record placed in user RAM is in the following format:

Byte Number	Contents
0-14	*filename/ext:d* (left-justified followed by spaces)
15	Protection Level, binary 0-6
16	Byte EOF, binary 0-255
17	Logical record length, binary 0-255
18-19	Last sector number in file, binary LSB, MSB
20-21	Number of Granules allocated (LSB,MSB) binary
22	"+" (marks the end of directory list after entire directory.)

Free Space Message Format

****nnnnn* Free Granules***

Where *nnnnn* is a decimal number. The entire message is ASCII-coded.

$FILPTR—17037/X'428D'

This routine provides information on any user file that is currently open. It enables you to obtain the drive number and the logical file number for any file and should be used in conjunction with $RAMDIR.

Entry Condition

(DE) = Data Control Block (DCB) defined when file was opened.
CALL $FILPTR

Exit Conditions

NZ = Error occurred.
Z = No error. The following registers are set up:
B = Which drive contains the file (0,1,2, or 3).
C = Logical file number (1-96)

Note: This operates with user files only.

$CLOSE—17448/X'4428'

$CLOSE closes a file from the last processing done. *It is very important to do a $CLOSE on every file opened before the program ends.* If you do not close a file, the directory entry for this file is incorrect if any new records have been written into the file. Other cases are not given here, but it is very important to TRSDOS that all of the "housekeeping" be complete for file management.

Entry Conditions

(DE) = DCB
CALL $CLOSE

Exit Conditions

Z = OK
A = TRSDOS error code.

$KILL—17452/X'442C'

$KILL deletes the directory entry for a file and releases the disk storage. The file may be open or closed; $KILL will operate in either case.

Entry Conditions

(DE) = DCB
CALL $KILL

Exit Conditions

Z = OK
A = TRSDOS error code.

Additional Routines and Storage Addresses

$JP2DOS—16429/X'402D'

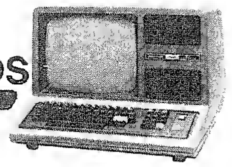
This routine transfers control to TRSDOS READY.

Entry Conditions

JP $JP2DOS

Exit Conditions

None



$DATE—12339/X'3033'
$TIME—12342/X'3036'

These routines return the date and time in ASCII format:

Date:	MM/DD/YY
Time:	HH/MM/SS

Entry Conditions

(HL) = Eight-byte buffer to receive the date/time text
CALL $DATE
CALL $TIME

Exit Conditions

(HL) = Date or time text

$DATLOC—16922/X'421A'
$TIMLOC—16919/X'4217'

These locations store the date and time in binary format:

$DATLOC (Three bytes):	YY DD YY
$TIMLOC (Three bytes):	SS MM HH

$ERRDSP—17417/X'4409'

This routine displays a TRSDOS error message determined by the contents of the accumulator (A). This register contains an error code (0 = no error) after completion of any system routine.

Entry Conditions

A = TRSDOS error code (see Table at the end of this section). In a TRSDOS error code, bits 6 and 7 are normally reset (off). So $ERRDSP interprets them as controls.

Bit #	Set	Not Set (Normal Condition)
7	Return to caller upon completion	Return to TRSDOS upon completion
6	Give detailed error message	Give error number only

CALL $ERRDSP

Exit Conditions

None

Sample Use

```
     CALL    $SYSRTN         ; ANY SYSTEM ROUTINE
     JR      Z,OKGO          ; CHECK FOR ERROR
; THE FOLLOWING INSTRUCTION SETS BIT 6 (DETAILED
; ERROR MESSAGE) AND BIT 7 (RETURN TO CALLER AFTER
; DISPLAYING MESSAGE
     OR      C0H             ; BINARY 11000000
     CALL    $ERRDSP         ; NOW CALL ERROR DISPLAY
; CONTINUE HERE UPON RETURN FROM ERROR DISPLAY
; YOUR ERROR HANDLER MAY GO HERE
;            :
;            :
;            :
OKGO  ;CONTINUATION AFTER $SYSRTN WITH NO ERROR
```

$DSPDIR—17433/X'4419'

This command displays the directory listing of all non-protected user files in a specified drive.

Entry Conditions

(X'4271') = ASCII-coded drive number "0," "1," "2," or "3"
CALL $DSPDIR

Exit Conditions

All registers are changed.

$COMDOS—17049/X'4299'

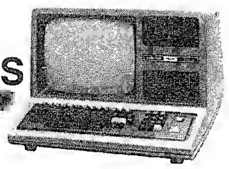
This routine executes a TRSDOS command and returns to TRSDOS READY.

Entry Conditions

(HL) = Text of TRSDOS command, terminated by X'0D.'
JP $COMDOS

Exit Conditions

None



$CMDDOS — 17052/X'429C'

This routine executes a TRSDOS command and returns to the caller.

Entry Conditions

(HL) = Text of TRSDOS command, terminated by X'0D.'

Exit Conditions

All registers are changed.

Caution: TRSDOS commands will overlay RAM up to X'6FFF.'

$CMDTXT — 16933/X'4225'

This is the start address of a buffer containing the last command line entered under TRSDOS READY. Using this buffer, your program may recover parameters that were included in the last command line.

For example, given a program named EDITOR/CMD, we want the operator to select an input file name when the program is loaded and executed from TRSDOS READY:

```
TRSDOS READY
EDITOR MYFILE
```

The program, EDITOR, can recover the name of the file in the $CMDTXT buffer.

Note: On entry to a program, (HL) = First non-blank character following the program name.

$MEMEND — 17425/X'4411'

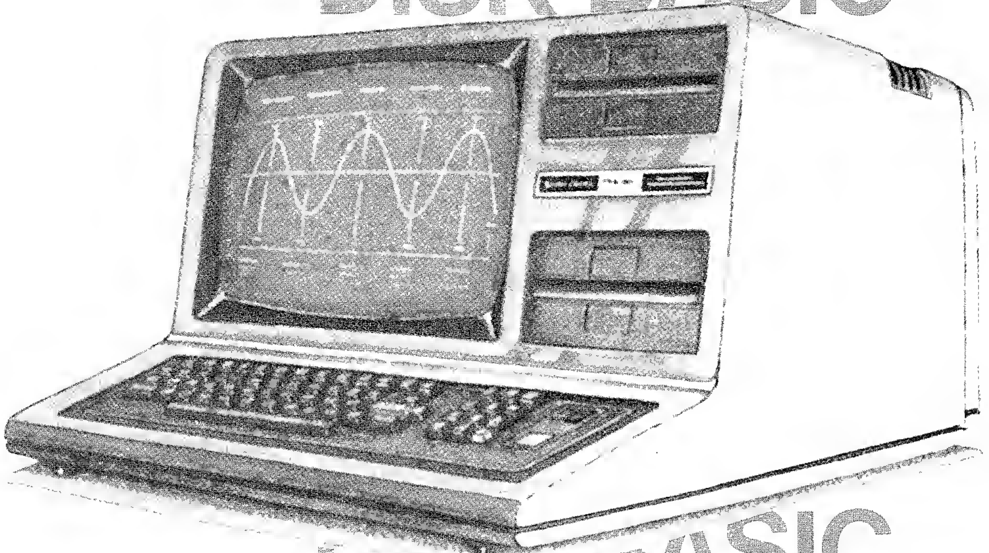
This storage location contains the highest address available. It is normally the same as the physical end of RAM, but you may change it for special purposes.

The address is in LSB, MSB sequence.

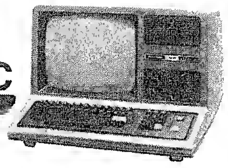
TRSDOS Error Codes/Messages

0	No Error Found
1	CRC Error During Disk I/O
2	Disk Drive Not In System
3	Lost Data During Disk I/O
4	CRC Error During Disk I/O
5	Disk Sector Not Found
6	Disk Drive Hardware Fault
7	**Undefined Error Code**
8	Disk Drive Not Ready
9	Illegal I/O Attempt
10	Required Command Parameter Not Found
11	Illegal Command Parameter
12	Time Out On Disk Drive
13	I/O Attempt To Non-System Disk
14	Write Fault On Disk I/O
15	Write Protected Disk
16	Illegal Logical File Number
17	Directory Read Error
18	Directory Write Error
19	Invalid File Name
20	GAT Read Error
21	GAT Write Error
22	HIT Read Error
23	HIT Write Error
24	File Not Found
25	File Access Denied Due to Password Protection
26	Directory Space Full
27	Disk Space Full
28	Attempt to Read Past EOF
29	Attempt to Read Outside of File Limits
30	No More Extents Available
31	Program Not Found
32	Invalid Drive Number
33	**Undefined Error Code**
34	Attempt to Use Non-program File as a Program
35	Memory Fault During Program Load
36	**Undefined Error Code**
37	File Access Denied Due to Password Protection
38	I/O Attempt to Unopen File
39	Invalid Command Parameter
40	File Already In Directory
41	Attempt to Open File Already Open

DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC
DISK BASIC



BASIC
DISK BASIC
DISK BASIC
DISK BASIC



Introduction

Start-Up

Under TRSDOS READY, type:

```
BASIC (ENTER)
```

TRSDOS will load BASIC and begin the ''initialization dialog.''

If you want to *recover* a Disk BASIC program after returning to TRSDOS for a DIR or other TRSDOS command, use this command under TRSDOS READY:

```
BASIC * (ENTER)
```

You will go directly to BASIC's READY mode without any initialization dialog. If you had a program in memory, it should still be there. You may not be able to run the program. To be safe, you should immediately save the program, go to TRSDOS, then start BASIC again (no asterisk).

Note: If you have overlaid user memory while in TRSDOS, your program will be erased. In such a case, you should not restart BASIC, but should use the normal BASIC start-up procedure.

Initialization

When you start Disk BASIC, you are first asked, HOW MANY FILES?. This lets you specify the maximum number of files that will be ''open'' or in use at once. (See OPEN.) Type in an appropriate number and press (ENTER), or simply press (ENTER) and BASIC will provide for three files.

For example, if your program requires one input file and one output file, you should ask for two files.

Note: Normally, BASIC will give all your data files a record length of 256. (See **File Access Techniques**.) If you wish to set the record length of each file *individually*, use the suffix V for ''Variable'' after the number of files.
For example,

```
HOW MANY FILES? 3V (ENTER)
```

tells BASIC to give you three **file-buffers**, and to let you set the record length of each file when that file is first opened.

Note: Disk BASIC automatically creates a buffer for loading, saving, and merging BASIC programs. This buffer exists in RAM below any data file buffers you may request. It is always available for program I/O, regardless of how you answer the FILES? question.

After you answer the FILES question, BASIC will ask: `MEMORY SIZE?` Simply press (ENTER) without typing a number. You will then have the maximum amount of RAM available for use by BASIC.

If you will want to load and use machine-language programs or routines, you will have to protect your BASIC memory from these machine-language programs.

In such a case, respond with the highest memory address (in decimal form) you want BASIC to use for storing and executing your BASIC programs. Addresses above the number you specify will then be protected from use by BASIC.

Example:

```
MEMORY SIZE? 32000 (ENTER)
```

causes BASIC to protect addresses above 32000. If you have 16K of RAM, this means that you'll have 32767-32000 = 767 bytes protected for storing your machine-language routines.

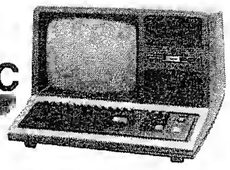
After you answer the `MEMORY SIZE?` question, Disk BASIC will display the following information:

1. Which version of Disk BASIC you are using
2. Copyright information
3. The number of free bytes available
4. The number of concurrent files you have requested.

To exit from Disk BASIC and return to the TRSDOS READY mode, type:

```
CMD"S" (ENTER)
```

This results in a normal return to TRSDOS, without re-initialization of the system. You may recover your program if you haven't changed user memory while in TRSDOS. Use BASIC *.



Enhancements to Model III BASIC

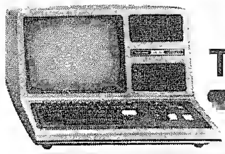
Disk BASIC adds many features which are not disk-related. They are listed below along with abbreviated descriptions. Detailed descriptions follow in alphabetical order.

&H	Hexadecimal-constant prefix
&O	Octal-constant prefix
Abbreviations	Many commands have abbreviations
CMD"A"	Return to TRSDOS with error message
CMD"B"	Enable/Disable (BREAK)
CMD"C"	Delete spaces and remarks from a program (compression)
CMD"D"	Display directory for specified drive
CMD"E"	Display previous TRSDOS error
CMD"I"	Return a command to TRSDOS
CMD"J"	Convert calendar date
CMD"L"	Load Z-80 subroutine or program file into RAM
CMD"O"	Alphabetizes (sorts) a string array only
CMD"P"	Check printer status
CMD"R"	Start real-time clock display
CMD"S"	Normal return to TRSDOS (jump to EXIT routine)
CMD"T"	Turn off real-time clock display
CMD"X"	Cross-reference of reserved words, string variables, or strings in a program
CMD"Z"	Duplicate output to Display and Printer
DEF FN	Define BASIC-statement function
DEF USR	Define the entry point for an external machine-language routine
INSTR	Instring function; find the substring in the target string
LINE INPUT	Input a line from keyboard
MID$=	Replace portion of the target string (used on left of equals sign)
NAME	Renumber a program in RAM
USR*n*	Call external routine ($n=0,1,2,\ldots,9$)

&H and &O (hex and octal constants)

Often it is convenient to use hexadecimal (base 16) or octal (base 8) constants rather than their decimal counterparts. For example, memory addresses and byte values are easier to manipulate in hex form. &H and &O let you introduce such constants into your program.

&H and &O are used as prefixes for the numerals that immediately follow them:



&H*dddd*

dddd is a 1 to 4 digit sequence composed of hexadecimal numerals 0,1,...9,A,B,...,F.

&O*dddddd*

ddddd is a sequence of octal numerals 0,1,...,7 and &O*dddddd*< =177777 decimal.

Note: The O can be omitted from the prefix &O. Therefore &O*dddddd*= &*ddddd*.

The constants always represent signed integers. Therefore any hex number greater than &H7FFF, or any octal number greater than &O77777, will be interpreted as a negative quantity. The following table illustrates this:

Octal	Hex	Decimal
&1	&H1	1
&2	&H2	2
&77777	&H7FFF	32767
&100000	&H8000	−32768
&100001	&H8001	−32767
&100002	&H8002	−32766
&177776	&HFFFE	−2
&177777	&HFFFF	−1

Hex and octal constants cannot be typed in as responses to an INPUT prompt or be contained in a DATA statement. Often the hex or octal constant must be enclosed in parentheses to prevent a syntax error from occurring.

Examples

```
PRINT &H5200, &O51000
```

prints the decimal equivalent of the two constants (both equal 20992).

```
POKE &H3C00, 42
```

puts decimal 42 (ASCII code for an asterisk) into video memory address hex 3C00.

Model III Disk BASIC Abbreviations

Abbreviation	Meaning
↑	List Previous Program Line
↓	List Next Program Line
.	List Current Program Line
,	Edit Current Program Line
SHIFT ↑	List First Program Line
SHIFT ↓ Z	List Last Program Line
L*xx*	List Program Line *xx*
E*xx*	Edit Program Line *xx*
D*xx*	Delete Program Line *xx*
A*xxx,xxxx*	Automatic Line Numbering Beginning at Line *xxx*, Incrementing by *xxxx.*

CMD "A"
Return to TRSDOS

```
CMD"A"
```

This command allows you to return to TRSDOS with an error message:

```
OPERATION ABORTED
```

Sample Use

After an input/output error occurs in a BASIC program, you might want to exit to TRSDOS and print a message.

```
CMD"A"
```

the following will be displayed:

```
OPERATION ABORTED
TRSDOS READY
```

+ + + + + + + + + + + + + + + + + +

CMD "B"
Enable/Disable BREAK Key

CMD"B", *"switch"*

switch is either ON or OFF. *switch* must be enclosed in quotation marks.

This command enables or disables the BREAK key. While the function is "OFF," the BREAK key will be ignored except during cassette or printer output or during serial input/output.

The BREAK key will remain disabled even after the program has ended. To enable the BREAK key, use the CMD"B","ON" command. Returning to TRSDOS via the CMD"S" or CMD"I" commands will also enable the BREAK key.

Examples

```
CMD"B","OFF"
```

Disables the BREAK key.

```
CMD"B","ON"
```

Returns the BREAK key to its normal function.

CMD "C"
Compress Program

CMD "C", *options*

options may be either R (delete remarks) or S (delete spaces). If both options are omitted, remarks and spaces are deleted. If only one is omitted, only the specified action is taken.

This command allows you to compress a program so that it requires less RAM and less storage space on diskette. You can elect to remove all remark statements (beginning with REM or ') or to delete all spaces between BASIC keywords. Spaces inside quotes will *not* be deleted.

Example

Your program reads as follows:

```
850 RESTORE: ON ERROR GOTO 800  'DOG PROGRAM
860 READ COMPANY$               'PET STORE
870 PRINT RIGHT$(COMPANY$,2),: GOTO 860
880 END
```

If you want to delete the Remarks (lines 850 and 860), type in the command:

```
CMD"C",R
```

and the program will now read:

```
850 RESTORE: ON ERROR GOTO 800
860 READ COMPANY$
870 PRINT RIGHT$(COMPANY$,2),:GOTO 860
880 END
```

If you then wanted to delete the spaces, type in:

```
CMD"C",S
```

and the program would read:

```
850 RESTORE:ONERRORGOTO800
860 READCOMPANY$
870 PRINTRIGHT$(COMPANY$,2),:GOTO860
880 END
```

You could obtain the same results by typing:

```
CMD"C"
```

Note: Always provide the closing quotes on string literals in your BASIC program. Otherwise CMD"C" may not function properly. For example, in

```
10 PRINT "THIS IS A TEST"
```

the second quote should be used even though omitting it will not cause an error.

CMD"D"
Display the Directory of a Specified Drive

CMD"D:*d*"

** *d* is the drive specification**

By entering the command CMD"D:*d*", you can obtain a specified diskette's directory from BASIC without returning to TRSDOS. Only unprotected, visible files will be displayed. The drive specification is not optional and must be specified for all drives, including Drive 0.

Example

If you type in the command:

```
CMD"D:1"
```

the directory for Drive 1 will be displayed.

CMD"E"
Display Previous TRSDOS error

CMD"E"

This command displays the last TRSDOS error message. If no errors have occurred prior to the command, the message `NO ERROR FOUND` will be displayed.

Example

If you have a two-drive system (0 and 1) and you type:

```
SAVE "PROGRAM:3"
```

Disk BASIC will return a `DISK I/O ERROR`. To find out what kind of I/O error occurred, type: `CMD"E"` (ENTER) and Disk BASIC will return with `DISK DRIVE NOT IN SYSTEM`.

CMD"I"
Execute TRSDOS Commands from Disk BASIC

CMD"I", *command*

command **is a string expression containing a TRSDOS command or a Z-80 program file name. If it is a string constant, it must be enclosed in quotes.**

You may execute TRSDOS commands directly from BASIC by using CMD"I".

This is similar to CMD"S", except that it lets you include a command or Z-80 program for TRSDOS to execute.

As long as BASIC is not overwritten by the execution of the program or command, control will return to BASIC; otherwise, control will return to TRSDOS. (TRSDOS commands all overlay BASIC; your Z-80 program may not if it loads *above* BASIC.)

Example

```
CMD"I","PROGRAM"
```

returns you to TRSDOS and executes the program file PROGRAM.

```
CMD"I",A$
```

returns you to TRSDOS and executes the command contained in A$.

CMD"J"
Calendar Date Conversion

CMD"J", *source, destination*

source is a string expression containing the date to be converted. Its contents may be in either of two formats:

A) *mm/dd/yy*

B) *-yy/ddd*

Format A gives the date in month-day-year sequence. Format B gives the day of the year (from 1 to 365 or 366 for leap years). In format B, the hyphen is required.

destination is a string variable to contain the *converted* date. If *source* is in format A, *destination* will contain the day of year. If *source* is in format B, *destination* will contain the date in format A.

This command converts dates back and forth between two formats: the standard month, day, year, sequence; and a year, day of year, sequence. The content of the source string determines which way the conversion goes.

Example

```
CMD"J", "11/30/80", D$
```

Returns the day of the year in D$.

```
CMD"J", "-79/300", D$
```

Returns the month, day, year, equivalent in D$ (the date for the 300th day of 1979).

Sample Program

```
10 CLEAR 50
20 LINE INPUT"ENTER FIRST DATE (MM/DD/YY) "; FD$
30 LINE INPUT"ENTER SECOND DATE (MM/DD/YY) ";SD$
40 CMD"J", FD$, D1$
50 CMD"J", SD$, D2$
60 Y1 = VAL(RIGHT$(FD$,2))
70 Y2 = VAL(RIGHT$(SD$,2))
80 J1 = VAL(RIGHT$(D1$,3))
90 J2 = VAL(RIGHT$(D2$,3))
100 S1 = Y1*365 + J1
110 S2 = Y2*365 + J2
120 PRINT "THE INTERVAL BETWEEN DATES IS";
130 PRINT ABS(S1-S2); "DAYS ";
140 PRINT "(IGNORING LEAP-YEARS)."
150 INPUT "<ENTER> TO CONTINUE"; A$
160 GOTO 20
```

CMD"L"
Load Z-80 Routine into RAM

CMD"L",*routine*

> ***routine*** **is a string expression containing a file specification for a Z-80 routine or program created by the DUMP command. If *routine* is a string constant, it must be enclosed in quotes.**

CMD"L" loads a Z-80 (machine-language) routine into RAM. It would normally be used to load a Z-80 subroutine which is to be accessed directly from BASIC.

The Z-80 routine should load into high-RAM and must not overlay the memory protect area reserved when you first entered BASIC (i.e., the `MEMORY SIZE?` prompt). If you do not overlay BASIC or TRSDOS, control will return to BASIC after the program is loaded.

Example

The command:

```
CMD"L","PROG"
```

will load a program file named PROG into RAM.

```
CMD"L",P$
```

will load a program which has been specified as P$.

CMD"O"
Sort ("Order") an Array

CMD"O",*x*,*array (start)*

> ***x*** **is an integer variable containing the number of items to be sorted.**
>
> ***array (start)*** **specifies an array element. The *array* contains the data to be sorted, and *start* is the subscript of the first element to be sorted. The *array* must be one-dimensional, string type. The string elements in *array* may be of any length.**

This command sorts (orders) a one-dimensional string array, i.e., a list. You may sort all or part of the array, depending on the values you give to *x* and *start*.

Sample Program

```
10 CLEAR 10 * 25 + 50   'ROOM FOR 10 WORDS + EXTRA
20 DIM A$(9)            'LIST OF TEN (0-9)
30 FOR WD = 0 TO 9
40 PRINT "ENTER WORD #"; WD+1
50 INPUT A$(WD)
60 NEXT WD
70 N%=10: CMD"O", N%, A$(0)
80 PRINT "HERE IS THE SORTED LIST"
90 FOR WD=0 TO 9
```

```
100 PRINT A$(WD)
110 NEXT WD
```

CMD"P" Check Printer Status

CMD"P",*status*
status is a string variable

CMD"P" makes it possible for Disk BASIC to check the status of the printer.

Unlike the video display, the printer is not always available. It may be disconnected, offline, out of paper, etc. In such cases, when you try to output information to the printer, the Computer will wait until the printer becomes available. It will appear to "hang up." To regain keyboard control (and cancel the printer operation), press (BREAK).

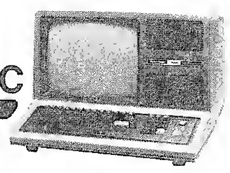
Suppose you have a program which uses printer output. If a printer is not available, you don't want the Computer to stop and wait for it to become available. Instead, you may want to print a message such as PRINTER UNAVAILABLE and go on to some other operation.

To accomplish this, you need to check the printer status. CMD"P" can be used to check the printer's status at any time. It returns the contents as an ASCII-coded decimal number. The specific value of this number depends upon the type of printer you are using as well as its status at any particular time. The value may then be printed or examined by the program.

Only the four most significant bits are used in this "status byte." In binary, these must be: "0011" or else the print operation will not be attempted. To check for this "go" condition, AND the status byte with 240 and compare the result with 48. The meaning of each status bit depends on which printer you use. See the printer owner's manual for bit designations.

Sample Program

```
10 CMD"P",X$
20 ST% = VAL(X$) AND 240
30 IF ST% <> 48 THEN PRINT "PRINTER UNAVAILABLE": STOP
40 PRINT "PRINTER AVAILABLE"
50 REM PROGRAM MAY NOW CONTINUE
```



CMD"R"
Turn On Clock-Display

```
CMD"R"
```

This command controls the real-time clock display in the upper-right corner of the Video Display. When it is on, the 24-hour time will be displayed and updated once each second, regardless of what program is executing.

Note: The real-time clock is always running (except during cassette or disk I/O), regardless of whether the display is on or off.

Example

To turn on the clock display type: `CMD"R"` To turn the display off, type: `CMD"T"`

CMD"S"
Return to TRSDOS

```
CMD"S"
```

To exit from Disk BASIC, returning control to TRSDOS, simply type in the command:

```
CMD"S"
```

To return to BASIC and recover your program, use `BASIC *`. However, recovery will not always be possible. See **BASIC** *.

Example

The BASIC prompt lets you know you are in Disk BASIC.

```
READY
>
```

To exit, type in:

```
CMD"S"
```

and the TRSDOS prompt will appear.

```
TRSDOS READY
++++++++++++++++
```

CMD"T"
Turn Off Clock-Display

```
CMD"T"
```

This command turns off the real-time clock display function.

However, the clock continues to run.

Example

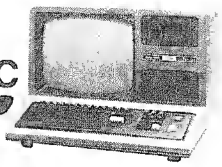
To stop the clock display update type: `CMD"T"`

To start the display, type: `CMD"R"`

CMD "X"
Cross-reference of Program Lines

CMD "X", *target*

target is either a BASIC reserved word (such as PRINT) or a string-literal. If it is a reserved word, it must not be enclosed in quotes; if it is a string-literal, it must be enclosed in quotes.



This command finds all occurrences of a reserved word or other string literal in the resident program. The ''finds'' are listed on the display as five-digit line numbers.

To search for any BASIC reserved word (including reserved arithmetic operators), use the keyword as-is. To search for anything else (including variable-names and text), enclose the text inside quotes.

For example, suppose you have the following program in memory:

```
10 PRINT "THIS IS A TEST"
20 INPUT "PRESS <ENTER> FOR THE NEXT PRINT MESSAGE"; Z$
30 A = A + 1
40 PRINT "+++++++"
```

`CMD "X", PRINT` will find all occurrences of PRINT, except for cases where PRINT was part of a quoted string: lines 10 and 40.

`CMD "X", "PRINT"` will find all occurrences of ''PRINT'' as a string literal: line 20.

`CMD "X", +` will list line 30, but `CMD "X", "+"` will list line 40. `CMD "X", "A"` will list lines 10, 20, and 30. Notice that variables and text are both treated as string literals.

CMD "Z"
Duplicate Output to Video and Printer

CMD"Z", *"switch"*

switch is either ON or OFF. _switch_ must be enclosed in quotation marks.

This command enables or disables dual video/printer output. While the function is ''ON,'' all video output is copied to the printer, and all printer output is copied to the video. (The printer *must* be on-line when you turn dual output ''ON.'')

Video and printer output may differ due to intrinsic differences in the printer and video devices.

Examples

```
CMD "Z", "ON"
```

Turns dual video/printer output on.

```
CMD"Z", "OFF"
```

Turns dual video/printer output off.

DEF FN
Define Function

DEF FN *function name (argument-1, . . .) = formula*

- *function name* is any valid variable name.
- *argument-1* and subsequent arguments are used in defining what the function does.
- *formula* is an expression usually involving the argument(s) passed on the left side of the equals sign.

The DEF FN statement lets you create your own function. That is, you only have to call the new function by name, and the associated operations will automatically be performed. Once a function has been defined with the DEF FN statement, you can call it simply by inserting FN in front of *function name*. You can use it exactly as you might use one of the built-in functions, like SIN, ABS, and STRING$.

The type of variable used for *function name* determines the type of value the function will return. For example, if *function name* is single precision, then that function will return a single-precision value, regardless of the precision of the arguments.

The particular variables you use as arguments in the DEF FN statement *(argument-1, . . .)* are not assigned to the function. When you call the function later, any variable name of the same type can be used.

Furthermore, using a variable as an argument in a DEF FN statement has no effect on the value of that variable. So you can use that particular variable in another part of your program without worrying about interference from DEF FN.

The function can be defined with no arguments at all, if none are required. For example:

```
DEF FNR = RND (90) + 9
```

defines a function to return a random value between 10 and 99.

Examples

```
DEF FNR(A,B) = A + INT((B - (A - 1)) * RND(0))
```

This statement defines function FNR which returns a random number between integers A and B. The values for A and B are passed when the function is "called," i.e., used in a statement like:

```
Y = FNR(R1, R2)
```

If R1 and R2 have been assigned the values 2 and 8, this line would assign a random number between 2 and 8 to Y.

```
DEF FNL$(X) = STRING$(X, "-")
```

Defines function FNL$ which returns a string of hyphens, X characters long. The value for X is passed when the function is called:

```
PRINT FNL$(3)
```

This line prints a string of 30 hyphens.

Here's an example showing DEF FN used for a complex computation — in double-precision.

```
DEF FNX#(A#, B#) = (A# - B#) * (A# - B#)
```

Defines function FNX# which returns the double-precision value of the square of the difference between A# and B#. The values for A# and B# are passed when the function is called:

```
S# = FNX#(A#, B#)
```

We assume that values for A# and B# were assigned elsewhere in the program.

Sample Program

```
710 DEF FNV(T) = (1087 + SQR(273 + T))/16.52
720 INPUT "AIR TEMPERATURE IN DEGREES CELSIUS"; T
730 PRINT "THE SPEED OF SOUND IN AIR OF" T "DEGREES
    CELSIUS IS" FNV(T) "FEET PER SECOND."
```

DEFUSR
Define Point of Entry for USR Routine

DEFUSR*n* = *address*

***n* equals one of the digits 0, 1,...,9; if *n* is omitted, 0 is assumed. *address* specifies the entry address to a machine-language routine. *address* must be in the range [−32768,32767]. *address* may be any numeric expression or constant from −32768 to 32767.**

DEFUSR lets you define the entry points for up to 10 machine-language routines. In non-Disk BASIC, the addresses were POKEd into RAM. This POKE method cannot be used in Disk BASIC.

Examples

```
DEFUSR3 = &H7D00
```

assigns the entry point X'7D00', 32000 decimal, to the USR3 call. When your program calls USR3, control will branch to your subroutine beginning at X'7D00'.

```
DEFUSR = (BASE + 16)
```

assigns start address (BASE + 16) to the USR0 routine.

Note: When decimal addresses are given, they are evaluated as signed two-byte integers. So, for addresses above 32767, use *desired decimal address −65536*. See **USR*n***.

INSTR
Search for Specified String

INSTR *(position, string 1, string 2)*

***position* specifies the position in *string 1* where the search is to begin. *position* is optional; if it is not supplied, search automatically begins at the first character in *string 1*. (Position 1 is the first character in *string 1*.)**

***string 1* is the string to be searched.**

***string 2* is the substring you want to search for.**

This function lets you search through a string to see if it contains another string. If it does, INSTR returns the starting position of the substring in the target string; otherwise, zero is returned. Note that the entire substring must be contained in the search string, or zero is returned. Also, note that INSTR only finds the first occurrence of a substring at the position you specify.

Examples

In these examples, A$ = "LINCOLN":

```
INSTR(A$, "INC")
```

returns a value of 2.

```
INSTR (A$, "12")
```

returns a zero.

```
INSTR(A$, "LINCOLNABRAHAM")
```

returns a zero. For a slightly different use of INSTR, look at

```
INSTR (3, "1232123", "12")
```

which returns 5.

Sample Program

This program gets search and target text from the keyboard, then locates all occurrences of the target text in the search text. Line 90 is just for "show."

```
10 CLEAR 1000
20 CLS
30 INPUT "SEARCH TEXT"; S$
40 INPUT "TARGET TEXT"; T$
45 CLS
50 C = 0 : P = 1 'P = POSITION, C = COUNT
60 F = INSTR(P,S$,T$)
70 IF F = 0 THEN 120
80 C = C + 1
90 PRINT @0,LEFT$(S$,F-1) + STRING$(LEN(T$),191) +
   RIGHT$(S$,LEN(S$)-F-LEN(T$)+1)
100 P = F + LEN(T$)
110 IF P <= LEN(S$) - LEN(T$) + 1 THEN 60
120 PRINT "FOUND "; C; "OCCURRENCES"
```

LINE INPUT
Input a Line from Keyboard

LINE INPUT *"prompt" ;variable*

prompt is a prompting message.

variable is the name that will be assigned to the line you type in.

LINE INPUT (or LINEINPUT — the space is optional) is similar to INPUT, except:

- The Computer will not display a question mark when waiting for your operator's input.
- Each LINE INPUT statement can assign a value to just one variable.
- Commas and quotes your operator can use as part of the string input.
- Leading blanks are not ignored — they become part of *variable*.
- The only way to terminate the string input is to press (ENTER).

LINE INPUT is a convenient way to input string data without having to worry about accidental entry of delimiters (commas, quotation marks, colons, etc.). The (ENTER) key serves as the only delimiter. If you want anyone to be able to input information into your program without special instructions, use the LINE INPUT statement.

Some situations require that you input commas, quotes and leading blanks as part of the data. LINE INPUT serves well in such cases.

Examples

```
LINE INPUT A$
```

Input A$ without displaying any prompt.

```
LINE INPUT "LAST NAME, FIRST NAME? ";N$
```

Displays a prompt message and inputs data. Commas will not terminate the input string, as they would in an input statement.

Sample Program

```
200 REM  CUSTOMER SURVEY
205 CLEAR 1000
207 PRINT
```

```
210 LINE INPUT "TYPE IN YOUR NAME "; A$
220 LINE INPUT "DO YOU LIKE YOUR COMPUTER? "; B$
230 LINE INPUT "WHY? "; C$
235 PRINT
240 PRINT A$ : PRINT
250 IF B$= "NO" THEN 270
260 PRINT "I LIKE MY COMPUTER BECAUSE "; C$ :END
270 PRINT "I DO NOT LIKE MY COMPUTER BECAUSE "; C$
```

Notice that when line 210 is executed, a question mark is not displayed after the statement, "Type in your name." Also, notice on line 230 you can answer the question "Why" with a statement full of delimiters, commas and quotes.

MID$ = Replace Portion of String

MID$ *(oldstring, position, length) = replacement-string*

oldstring **is the variable-name of the string you want to change.**

position **is the numeric expression specifying the position of the first character to be changed.**

length **is a numeric expression specifying the number of characters to be replaced.**

replacement-string **is a string expression to replace the specified portion of oldstring.**

Note: If *replacement-string* is shorter than *length*, then the entire *replacement-string* will be used.

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

Note that the length of the resultant string is always the same as the original string.

Examples

A$ = "LINCOLN" in the examples below:

```
MID$(A$, 3, 4) = "12345": PRINT A$
```

which returns LI1234N.

```
MID$(A$, 1, 2) = "": PRINT A$
```

which returns LINCOLN.

```
MID$(A$, 5) = "12345": PRINT A$
```

returns LINC123.

```
MID$(A$, 5) = "01": PRINT A$
```

returns LINC01N.

```
MID$(A$, 1, 3) = "***": PRINT A$
```

returns ***COLN.

Sample Program

```
770 CLS: PRINT: PRINT
780 LINE INPUT "TYPE IN A MONTH AND DAY MM/DD, "; S$
790 P = INSTR(S$, "/")
800 IF P = 0 THEN 780
810 MID$(S$, P, 1) = CHR$(45)
820 PRINT S$ " IS EASIER TO READ, ISN'T IT?"
```

This program uses INSTR to search for the slash (''/''). When it finds it (if it finds it), it uses MID$= to substitute a '' – '' (CHR$(45)) for it.

NAME
Renumber the Current Program

NAME *newline, startline, increment*

> ***newline*** **specifies the new line number of the first line to be renumbered. If omitted, 10 is used.**
>
> ***startline*** **specifies the line number in the original program where renumbering will start. If omitted, the entire program will be renumbered.**
>
> ***increment*** **specifies the increment to be used between each successive line number. If omitted, 10 is used.**

Examples

```
NAME
```

Renumbers the entire program: 10, 20, 30, . . .

```
NAME 6000,5000,100
```

Renumbers all lines numbered from 5000 up; the first renumbered line will become 6000, and the following lines will be incremented by 100. All line references within your program will be renumbered also.

USRn
Call to User's External Subroutine

USR*n* (*nmexp*)

> **where *n* specifies one of ten available USR calls, *n*=0,1,2,...,9. If *n* is omitted, zero is assumed.**
>
> ***nmexp* is an integer from −32768 to 32767 and is passed as an integer argument to the routine.**

These functions (USR0 through USR9) transfer control to machine-language routines previously defined with DEFUSR*n* statements.

When a URS call is encountered in a statement, control goes to the address specified in the DEFUSR*n* statement. This address specifies the entry point to your machine-language routine.

Note: If you call a USR*n* routine before defining the routine entry point with DEFUSR*n*, an ILLEGAL FUNCTION CALL error will occur.

You can pass one argument and retrieve one output value directly via the USR argument; or you can pass and retrieve arguments indirectly via POKE and PEEK statements.

Example

```
10 DEFUSR1=&H7D00
20 REM...MORE PROGRAM LINES HERE
100 A=USR1(X)
```

The effect of this sequence is to:

1. Define USR as a routine with an entry point at hex 7D00 (line 10).
2. Transfer control to the routine; the value X can be passed to the routine if the routine makes the CALL described below (line 100).
3. When the routine returns to BASIC, the variable A may contain the value passed back from the routine (if your routine makes the JUMP described below); otherwise A will be assigned the value of X (line 100).

Passing arguments to and from USR routines

There are several ways to pass arguments back and forth between your BASIC main program and your USR routines: the two major ways are listed below.

1. POKE the argument(s) into fixed RAM locations. The machine-language routine can then access these values and place results in other RAM locations. When the routine returns control to BASIC, your program can PEEK into these addresses to pick up the "output" values. **This is the only way to pass two or more arguments back and forth.**
2. Pass one argument to the routine as the argument in the USRn call, then use special ROM calls to access this argument and return a value to BASIC. **This method is limited to sending one argument and returning one value** (both are integers).

ROM Calls

CALL 0A7FH	Puts the USR argument into the HL register pair; H contains MSB, L contains LSB. This CALL should be the first instruction in your USR routine.
JP 0A9AH	Use this JUMP to return to BASIC; the integer in HL becomes the output of the USR call. If you don't care about returning HL, then execute a simple RETurn instruction instead of this JUMP.

Listed below is an assembled program to white out the display (an "inverse" CLEAR key!). Don't type it in. Type in the BASIC program that follows it.

```
                00100 ;
                00110 ; ZAP OUT SCREEN USR FUNCTION
                00120 ;
7D00            00130           ORG     7D00H
                00140 ;
                00150 ; EQUATES
                00160 ;
3C00            00170 VIDEO     EQU     3C00H           ;START OF VIDEO RAM
00BF            00180 WHITE     EQU     0BFH            ;ALL WHITE GRAPHICS BYTE
03FF            00190 COUNT     EQU     3FFH            ;NUMBER OF BYTES TO MOVE
                00200 ;
                00210 ; PROGRAM CHAIN MOVES X'BF' INTO ALL OF VIDEO RAM
                00220 ;
```

```
7D00 21003C      00230 ZAP       LD      HL,VIDEO       ;SOURCE ADDRESS
7D03 36BF        00240           LD      (HL),WHITE     ;PUT OUT 1ST BYTE
7D05 11013C      00250           LD      DE,VIDEO+1     ;DESTINATION ADDRESS
7D08 01FF03      00260           LD      BC,COUNT       ;NUMBER OF ITERATIONS
7D0B EDB0        00270           LDIR                   ;DO IT TO IT!!!
                 00280 ;
7D0D C9          00290           RET                    ;RETURN TO BASIC
7D00             00300           END     ZAP
```

This routine can be POKEd into RAM and accessed as a USR routine. First start BASIC and answer the `MEMORY SIZE` question with `31999`. Then run the program.

```
100 '         PROGRAM: USR1
110 ' EXAMPLE OF A USER MACHINE LANGUAGE FUNCTION
115 ' DEPRESS THE '@' KEY WHILE NUMBERS ARE PRINTING TO STOP
120 '
130 ' ******* POKE MACHINE PROGRAM INTO MEMORY *******
140 '
150 DEFUSR1 = &H7D00
160 FOR X = 32000 TO 32013   '7D00 HEX EQUAL 32000 DECIMAL
170     READ A
180     POKE X, A
190 NEXT X
192 '
194 ' ******* CLEAR SCREEN & PRINT NUMBERS 1 THRU 100 *******
196 '
200 CLS
205 PRINT TAB(15); "WHITE-OUT USER ROUTINE": PRINT
210 FOR X = 1 TO 100
220     PRINT X;
225     A$ = INKEY$: IF A$ = "@" THEN END
230 NEXT X
240 '
250 ' ******* JUMP TO WHITE-OUT SUBROUTINE *******
260 '
270 X = USR1 (0)
280 FOR X = 1 TO 1000: NEXT X   'DELAY LOOP
290 GOTO 200
300 '
310 ' ******* DATA IS DECIMAL CODE FOR HEX PROGRAM *******
320 '
330 DATA 33,0,60,54,191,17,1,60,1,255,3,237,176,201
```

Run the program. An equivalent BASIC white out routine takes a long time by comparison!

Disk-Related Features

Disk BASIC provides a powerful set of commands, statements and functions relating to disk I/O under TRSDOS. These fall into two categories:

1. File manipulation: dealing with files as units, rather than with the distinct records the files contain.
2. File access: preparing data files for I/O; reading and writing to the files.

Under the heading, **File Manipulation**, we will discuss the following commands.

KILL	Delete a program or data file from the disk
LOAD	Load a BASIC program from disk
MERGE	Merge an ASCII-format BASIC program on disk with one currently in RAM
RUN*"program"*	Load and execute a BASIC program stored on disk
SAVE	Save the resident BASIC program on disk

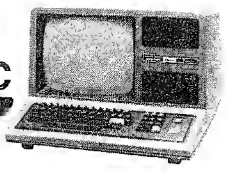
Under the heading, **File Access**, we will discuss the following statements and functions.

Statements

OPEN	Open a file for access (create the file if necessary)
CLOSE	Close access to the file
INPUT #	Read from disk, sequential mode
LINE INPUT#	Read a line of data, sequential mode
PRINT#	Write to disk, sequential mode
FIELD	Assign field sizes and names to random-access file buffer
GET	Read from disk, random access mode
PUT	Write to disk, random access mode
LSET	Place value in specified buffer field, add blanks on the right to fill field
RSET	Place value in specified buffer field, add blanks on the left to fill field

Functions

CVD	Restore double-precision number to numeric form after GETting from disk
CVI	Restore integer to numeric form after GETting from disk
CVS	Restore single-precision number to numeric form after GETting from disk
EOF	Check to see if end of file encountered during read
LOC	GET current record number.



LOF	Return number of last record in file
MKD$	Convert double-precision number to string so it can be PUT on disk
MKI$	Convert integer to string so it can be PUT on disk
MKS$	Convert single-precision number to string so it can be PUT on disk

File Manipulation

KILL
Delete a File from the Disk

KILL *exp$*

&emsp;*exp$* defines a file specification for an existing file.

This command works like the TRSDOS KILL command—see TRSDOS **Library Commands.**

Example

```
KILL"OLDFILE/BAS.PSW1"
```

deletes the file specified from the first drive which contains it.

Do not KILL an open file, or you may destroy the contents of the diskette. (First, CLOSE the open file.)

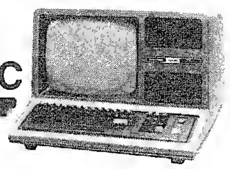
LOAD
Load BASIC Program File from Disk

LOAD *exp$* [,R]

** where *exp$* defines a filespec for a BASIC program file stored on disk.**

** R tells BASIC to RUN the program after it is loaded.**

This command loads a BASIC program file into RAM; if the R option is used, BASIC will proceed to RUN the program automatically; otherwise, BASIC will return to the command mode.



LOAD without the R option clears all variables and closes all open files. LOAD with the R option clears all variables but does not close the open files.

LOAD with the R option is equivalent to the command RUN *exp$*,R. Either of these commands can be used inside programs to allow program chaining—one program calling another, etc.

Example

```
LOAD"PROG1/BAS:2"
```

Clears resident BASIC program and loads PROG1/BAS from Drive 2; returns to BASIC command mode.

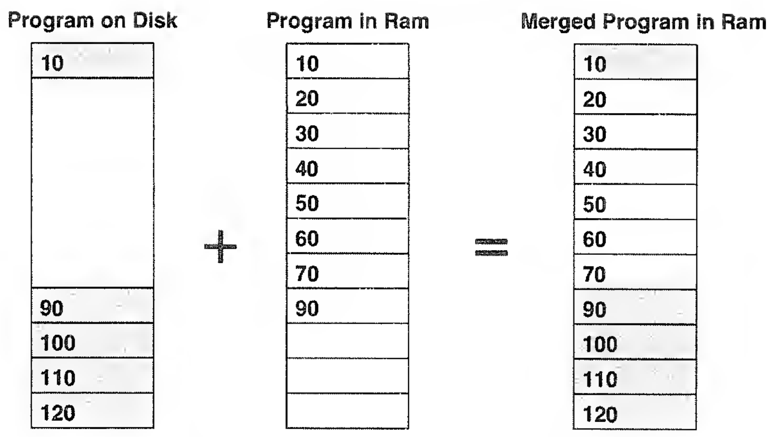
MERGE
Merge Disk Program with Resident Program

MERGE *exp$*

***exp$* is file specification for an ASCII-format BASIC disk file, e.g., a program saved with the A-option.**

MERGE is similar to LOAD—except that the resident program is not erased before the new program *exp$* is loaded. Instead, the new program is merged into the resident program.

That is, program lines in *exp$* will simply be inserted into the resident program in sequential order. If line numbers in *exp$* coincide with line numbers in the resident program, the resident lines will be replaced by those from *exp$*.



Sample Use

Save this program in ASCII format.

```
1000 REM . . . SUBROUTINE TO SAY HELLO
1010 PRINT "HELLO!"
1020 RETURN
```

Type NEW (ENTER), then type in this program.

```
100 CLS
110 PRINT "LET'S CALL THE SUBROUTINE . . ."
120 PRINT "DIALING NOW . . ."
130 FOR I=1 TO 1000 : NEXT
140 GOSUB 1000
150 PRINT "BACK FROM SUBROUTINE."
160 END
```

Now type MERGE *"file"* using the file name given to the first file. List the program. Then run it.

RUN"program"
Load and Execute a Program from Disk

RUN *file*[,R]

file **is the name of a BASIC program file. It is a string expression. (If a string constant is used, it must be enclosed in quotes.) The ,R option causes BASIC to leave open files open. If omitted, open files are closed before the program is run.**

This command loads and executes a BASIC program stored on disk. It may be used inside a program to allow chaining (one program calling another).

Examples

```
RUN "PROG"
```

Loads and executes PROG (all open files are closed first).

```
A$="NEWPROG"
RUN A$, R
```

Loads and executes NEWPROG (all open files remain open).

SAVE
Save Program onto Disk

SAVE *file* [,A]

> ***file*** **is the name of a BASIC program file. It is a string expression. (If a string constant is used, it must be enclosed in quotes.)**
>
> **A causes the file to be stored in ASCII rather than compressed format.**

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

Using compressed format takes up less disk space and is faster during both SAVES and LOADS. Using the ASCII option makes it possible to do certain things that cannot be done with compressed format BASIC files.

For example:

- The MERGE command requires that the disk file be in ASCII form.
- Programs which read in other programs as data will typically require that the data programs be stored in ASCII.
- The TRSDOS command APPEND also requires that disk files be in ASCII form.

[illegible]
13569 / 6512

Examples

```
SAVE"FILE1/BAS. JOHNQDOE:3"
```

saves the resident BASIC program in compressed format with the file name FILE1, extension /BAS, password .JOHNQDOE; the file is placed on Drive :3.

```
SAVE"MATHPAK/TXT",A
```

saves the resident program in ASCII form, using the name MATHPAK/TXT, on the first nonwrite-protected diskette.

Upon completion of a SAVE, BASIC returns in the command mode.

File Access

This section is divided into four parts:

1. Creating files and assigning buffers — OPEN and CLOSE
2. Statements and functions
3. Sequential I/O techniques
4. Random I/O techniques

If this is your first experience with disk file access, you should concentrate on parts 1, 3 and 4, perhaps just skimming through part 2 to get a general idea of how the functions and statements work. Later you can go back to part 2 and learn the details of statement and function syntax.

Creating Files and Assigning Buffers

During the initialization dialog, you type in a number in response to HOW MANY FILES? The number you type in tells BASIC how many buffers to create to handle your disk accesses (reads and writes).

Each buffer is given a number from 1 to 15. If you type:

```
HOW MANY FILES? 3V (ENTER)
```

BASIC sets aside 3 buffers, numbered 1,2,3.

You can think of a buffer as a waiting area that data must pass through on the way to and from the disk file. When you want to access a particular file, you must tell BASIC which buffer to use in accessing that file. You must also tell BASIC what kind of access you want — sequential output, sequential input, or random input/output.

All this is done with the OPEN statement, and "undone" with the CLOSE statement.

OPEN
Open a File

OPEN *mode, buffer, file, record-length*

mode **is a string expression containing one of the following:**

I Sequential input starting at the first record. If the file is not found, it will be created.

- **O** Sequential output starting at the first record. If the file is not found, it will be created.
- **E** (Extend) Sequential output starting at end of file. If the file is not found, it will be created.
- **R** Random input/output. If the file is not found, it will be created.

If *mode* is a constant, it must be enclosed in quotes. COULD BE A VARIABLE !

buffer is a numeric expression specifying which buffer is to be used.

file is a string expression containing the file specification. If a constant is used, it must be enclosed in quotes. AS ABOVE ?

record-length is a numeric expression from 0 to 256 specifying the logical record length. 0 is the same as 256. This option may only be used if variable-length records were requested during initialization (How Many Files?). If record-length is omitted, 256 is used. *record-length* is used with Random access *only*.

This statement lets you create a file, write data into it, update it, and read it. For details on file access, see Methods of Access later in this section.

If *file* includes a drive specification, BASIC will use only the specified drive. If no drive is specified, BASIC will search for a matching file, starting with the master drive (usually Drive 0).

Examples

```
OPEN "O", 1, "DATAFILE"
```

Opens DATAFILE (creates it if it doesn't already exist) for sequential output. Output will be done through buffer #1. Records will be 256 bytes long. Since the "O" mode is specified, output will start at the first record in the file. If "E" is used instead of "O", output will start at the end of the file.

```
OPEN "R", 2, "PAYROLL/A:1", 64
```

Opens/creates PAYROLL/A for random input/output. Access will be through buffer #2. Records will be 64 bytes long (if BASIC was initialized for variable-length records).

RECORD LENGTH

```
BUFFER = 3: FILE$ = "DATA": RECLN = 128
OPEN "R", BUFFER, FILE$, RECLN
```

½ SECTOR

Opens/creates DATA for random input/output. Access will be through buffer #3. Records will be 128 bytes long (if BASIC was initialized for variable-length records).

CLOSE
Close Access to the File

CLOSE [*nmexp* [,*nmexp*. . .]]

nmexp has a value from 1 to 15, and refers to the file's buffer number (assigned when the file was opened). If _nmexp_ is omitted, all open files will be closed.

This command terminates access to a file through the specified buffer(s). If *nmexp* has not been assigned in a previous OPEN statement, then

CLOSE *nmexp*

has no effect.

Examples

```
CLOSE 1,2,8
```

Terminates the file assignments to buffers 1, 2 and 8. These buffers can now be assigned to other files with OPEN statements.

```
CLOSE FIRST%+COUNT%
```

Terminates the file assignment to the buffer specified by the sum (FIRST% + COUNT%).

Do not remove a diskette which contains a file opened for writing (mode = O, E, or R). First close the file. This is because the last 256 bytes of data may not have been written to disk yet. Closing the file will write the data, if it hasn't already been written.

Any modification to the resident program (NEW, editing, LOAD, MERGE, etc.) will cause open files to be closed.

INPUT# Sequential Read from Disk

INPUT# *nmexp*, *var*[,*var*. . .]

where *nmexp* specifies a sequential input file buffer, *nmexp*=1,2,...,15.

***var* is the variable name to contain the data from the file.**

This statement inputs data from a disk file. The data is input sequentially. That is, when the file is first opened, a pointer is set to the beginning of the file. Each time data is input, the pointer advances. To start over reading from the beginning of the file, you must close the file and re-open it.

INPUT# doesn't care how the data was placed on the disk—whether a single PRINT# statement put it there, or whether it required 10 different PRINT# statements. What matters to INPUT# are the positions of the terminating characters and the EOF marker.

To INPUT# data successfully from disk, you need to know ahead of time what the format of the data is. Here is a description of how INPUT# interprets the various characters it encounters when reading data.

When inputting data into a variable, BASIC ignores leading blanks; when the first non-blank character is encountered, BASIC assumes it has encountered the beginning of the data item.

The data item ends when a terminating character is encountered or when a terminating condition occurs. The particular terminating characters vary, depending on whether BASIC is inputting to a numeric or string variable.

Special Note

Here's an important exception to keep in mind in reading the following material.

When (ENTER) (a carriage return) is preceded by (↓) (a line feed), the (ENTER) is not taken as a terminator. Instead, it becomes a part of the data item (string variable) or is simply ignored (numeric variable).

(To enter the (↓) character from the keyboard, press the down-arrow character. To enter the (ENTER) character, press (ENTER).)

This exception applies to all cases noted below where (ENTER) is said to be a terminator.

Numeric Input

Suppose the data image on disk is

```
 1.234 -33  27 (ENTER)
```

(ENTER) denotes a carriage-return character (ASCII code decimal 13).

Then the statement

```
INPUT#1, A,B,C
```

or the sequence of statements

```
INPUT#1,A: INPUT#1,B: INPUT#1,C
```

will assign the values as follows:

A = 1.234
B = −33
C = 27

This works because blanks and (ENTER) serve as terminators for input to numeric variables. The blank before 1.234 is a "leading blank," therefore it is ignored. The blank after 1.234 is a terminator; therefore BASIC starts inputting the second variable at the − character, inputs the number −33, and takes the next two blanks as terminators. The third input begins at the 2 and ends with the 7.

String Input

When reading data into a string variable, INPUT ignores all leading blanks; the first non-blank character is taken as the beginning of the data item.

If this first character is a double-quote ("), then INPUT will evaluate the data as a quoted string: it will read in all subsequent characters up to the next double-quote. Commas, blanks, and (ENTER) characters will be included in the string. The quotes themselves do not become a part of the string.

If the first character of the string item is not a double-quote, then INPUT will evaluate the data as an unquoted string: it will read in all subsequent characters up to the first comma, or (ENTER). If double-quotes are encountered, they will be included in the string.

For example, if the data on disk is:

PECOS, TEXAS"GOOD MELONS"

Then the statement

```
INPUT#1, A$,B$,C$
```

would assign values as follows:

A$ = PECOS

B$ = TEXAS "GOOD MELONS"

C$ = null string

If a comma is inserted in the data image before the first double quote, C$ will get the value, GOOD MELONS.

These are very simple examples just to give you an idea of how INPUT works. However, there are many other ways to input data—different terminators, different target variable types, etc.

Rather than taking a shotgun approach and trying to cover them all, we'll give a generalized description of how input works and what the terminating characters and conditions are, and then provide several examples.

When BASIC encounters a terminating character, it scans ahead to see how many more terminating characters it can include with the first terminator. This ensures that BASIC will begin looking for the next data item at the correct place.

The list below defines the various terminating sets INPUT# will look for. It will always try to take-in *the largest set possible*.

Numeric-input terminator sets

end of file encountered
255th data character encountered
,(comma)
ENTER
ENTER ↓
[. . .] [ENTER]
[. . .] [ENTER ↓]

Quoted-string terminator sets

end of file encountered
255th data character encountered
" (double quote)
" [. . .] [,]
" [. . .] [ENTER]
" [. . .] [ENTER ↓]

Unquoted-string terminator sets

end of file encountered
255th data character encountered
,
ENTER [↓]

Figure 13 describes how INPUT# assigns data to a variable.

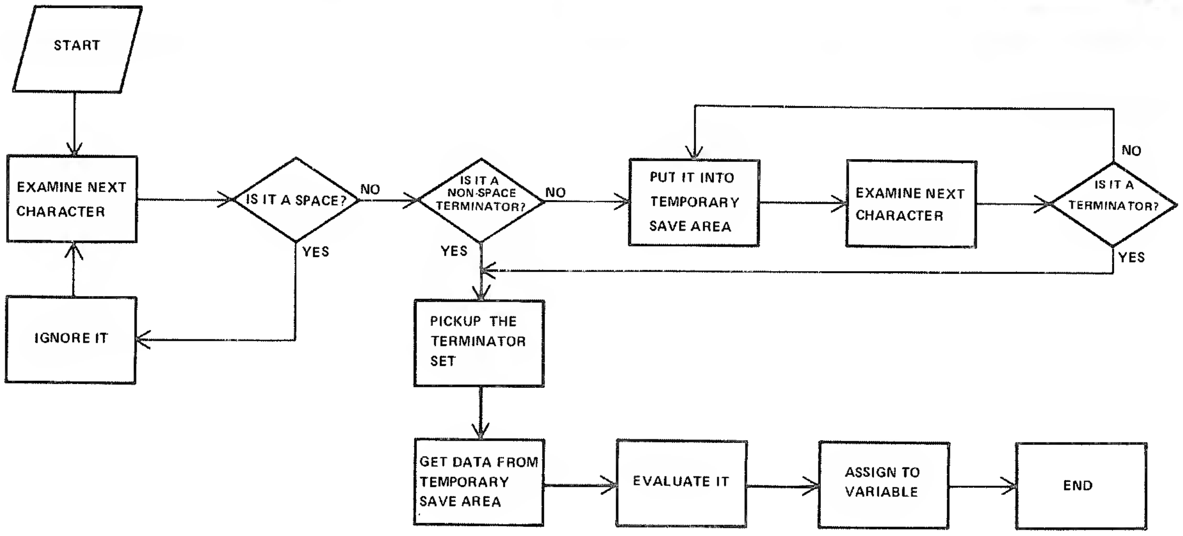


Figure 13. Input process.

The following table shows how various data images will be read-in by the statement:

```
INPUT#1,A,B,C
```

Ex.#	Image on disk	Values assigned

1	`123,45` (ENTER)(↓) `8,2E4` `7000`(ENTER)	A=123.45
		B=82000
		C=7000

2	`3`(↓)(ENTER) `4` (ENTER)`5` (ENTER) `A12` eof	A=34
		B=5
		C=0

3	`1,,2,3,4` (ENTER)	A=1
		B=0
		C=2

4	`1,3,` eof	A=1
		B=3
		C=0 eof error

(eof = end of file):

In Example 2 above, why does variable C get the value 0? When the input reaches the end of file, it terminates that last data item, which then contains "A12." This is evaluated by a routine just like the BASIC VAL function—which returns a zero since the first character of "A12" is a non-numeric.

In Example 3, when INPUT# goes looking for the second data item, it immediately encounters a terminator (the comma); therefore, variable B is given the value zero.

The following table shows how various data images on disk will be read by the statement:

```
INPUT#1,A$,B$
```

Ex.#	Image on disk	Values assigned
1	`"ROBERTS,J."ROBERTS,M.N` eof	A$:ROBERTS,J. B$:ROBERTS,M.N.
2	`ROBERTS,J.,   ROBERTS,M.N.` ENTER	A$:ROBERTS B$:J.
3	`THE WORD "QUO",12345.789` ENTER	A$:THE WORD "QUO" B$:12345.789
4	`BYTE`↓ ENTER `UNIT OF MEMORY` eof	A$:BYTE↓ENTER UNIT OF MEMORY B$:null (eof error)

In Example 3, the first data item is an unquoted string, therefore, the double-quotes are not terminators, and become part of A$.

In Example 4, the ENTER is preceded by an ↓, therefore it does not terminate the first string; both ↓ and ENTER are included in A$.

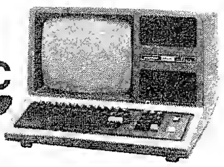
LINE INPUT#
Read a Line of Text from Disk

LINE INPUT#*nmexp,var$*

where *nmexp* specifies a sequential output file buffer, *nmexp*=1,2,...,15.

***var$* is the variable name to contain the string data.**

Similar to LINE INPUT from keyboard, this statement reads a "line" of string data into *var$*. This is useful when you want to read an ASCII-format BASIC program file as data, or when you want to read in data without following the usual restrictions regarding leading characters and terminators.



LINE INPUT (or LINEINPUT — the space is optional) reads everything from the first character up to:

1. an ENTER character which is not preceded by ↓
2. the end of file
3. the 255th data character (this 255 character is included in the string)

Other characters encountered — quotes, commas, leading blanks, ↓ ENTER pairs — are included in the string.

For example, if the data looks like:

```
10 CLEAR 500 ENTER
20 OPEN"I",1,"PROG" ENTER
.
.
.
```

then the statement

```
LINEINPUT#1,A$
```

could be used repetitively to read each program line, one line at a time.

PRINT#
Sequential Write to Disk File

PRINT#*nmexp*,[USING *format$;*] *exp*[*p exp . . .*]

where *nmexp* specifies a sequential output file buffer, *nmexp*=1,2,...,15.

***format$* is a sequence of field specifiers used with the USING option.**

***p* is a delimiter placed between every two expressions to be PRINTed to disk; either a semi-colon or comma can be used (semi-colon is preferable).**

***exp* is the expression to be evaluated and written to disk.**

This statement writes data sequentially to the specified file. When you first open a file for sequential output, a pointer is set to the beginning of the file, therefore your first PRINT# places data at the beginning of the file. At the end of each PRINT# operation, the pointer advances, so the values are written in sequence.

A PRINT# statement creates a disk image similar to what a PRINT to display creates on the screen. Remember this, and you'll be able to set up your PRINT# list correctly for access by one or more INPUT statements.

PRINT# does not compress the data before writing it to disk; it writes an ASCII-coded image of the data.

For example, if A = 123.45

```
PRINT#1,A
```

will write a nine-byte character sequence onto disk:

```
 123.45  (ENTER)
```

The punctuation in the PRINT list is very important. Unquoted commas and semi-colons have the same effect as they do in regular PRINT to display statements.

For example, if A = 2300 and B = 1.303, then

```
PRINT#1,A,B
```

places the data on disk as

```
 2300             1.303  (ENTER)
```

The comma between A and B in the PRINT# list causes 10 extra spaces in the disk file. Generally you wouldn't want to use up disk space this way, so you should use semi-colons instead of commas.

```
PRINT#1,A;B
```

writes the data as:

```
 2300   1.303 (ENTER)
```

PRINT# with numeric data is quite straightforward—just remember to separate the items with semi-colons.

PRINT# with string data requires more care, primarily because you have to insert delimiters so the data can be read back correctly. In particular, you must separate string items with explicit delimiters if you want to INPUT# them as distinct strings.

For example, suppose:

`A$="JOHN Q. DOE"` and `B$="100-01-001"`

Then:

```
PRINT#1, A$;B$
```

would produce this image on disk:

```
JOHN Q. DOE100-01-001 (ENTER)
```

which could not be INPUT back into two variables.

The statement:

```
PRINT#1, A$;",";B$
```

would produce:

```
JOHN Q. DOE, 100-01-001
```

which could be INPUT# back into two variables.

This method is adequate if the string data contains no delimiters — commas or (ENTER) — characters. But if the data does contain delimiters or leading blanks that you don't want to ignore, then you must supply explicit quotes to be written along with the data. For example, suppose `A$="DOE, JOHN Q."` and `B$="100-01-001"`

If you use

```
PRINT#1,A$;",";B$
```

the disk image will be:

```
DOE, JOHN Q.,100-01-001 (ENTER)
```

When you try to input this with a statement like

```
INPUT#2,A$,B$
```

A$ will get the value DOE, and B$ will get JOHN Q. — because of the comma after DOE in the disk image.

To write this data so that it can be input correctly, you must use the CHR$ function to insert explicit double quotes into the disk image. Since 34 is the decimal ASCII code for double quotes, use CHR$(34) as follows:

```
PRINT#1,CHR$(34);A$;CHR$(34);B$
```

this produces the disk image

```
"DOE, JOHN Q."100-01-001 (ENTER)
```

which can be read with a simple

```
INPUT#2,A$B$
```

Note: You can also use the CHR$ function to insert other delimiters and control codes into the file, for example:

CHR$(10)	(↓) Line Feed
CHR$(13)	carriage return ((ENTER)character)
CHR$(11) or CHR$(12)	line-printer top-of-form

USING Option

This option makes it easy to write files in a carefully controlled format.

For example, suppose:

```
A$="LUDWIG"
```

```
B$="VAN"
C$="BEETHOVEN"
```

Then the statement

```
PRINT#1,USING"!. !. %  %";A$;B$;C$
```

would write the data in nickname form:

```
L.V.BEET <ENTER>
```

(In this case, we didn't want to add any explicit delimiters.) See the PRINT USING description in the *LEVEL II BASIC Reference Manual* for a complete explanation of the field-specifiers.

Random Access Statements

FIELD
Organize a Random File-Buffer into Fields

FIELD *nmexp,nmexp1* AS *var1$* [,*nmexp2* AS *var2$*. . .]

nmexp specifies a random access file buffer, _nmexp_=1,2,...,15.

nmexp1 specifies the length of the first field.

var1$ defines a variable name for the first field.

nmexp2 specifies the length of the second field.

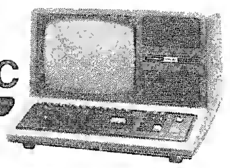
var2$ defines a variable name for the second field.

. . . Subsequent _nmexp_ AS _var$_ pairs define other fields in the buffer.

Note: The sum of all the field-lengths must not exceed the record length, and should equal the record length.

Before FIELDing a buffer, you must use an OPEN statement to assign that buffer to a particular disk file (you must use random access mode). Then use the FIELD statement to organize a random file buffer so that you can pass data from BASIC to disk storage and vice-versa.

Each random file buffer has up to 256 bytes which can store data for transfer from disk storage to BASIC or from BASIC to disk. (When variable-length files are used, maximum may be from 1 to 256.) However, you need a way to access this



buffer from BASIC so that you can either read the data it contains or place new data in it. The FIELD statement provides the means of access.

You may use the FIELD statement any number of times to ''re-organize'' a file buffer. FIELDing a buffer does not clear the contents of the buffer; only the means of accessing the buffer (the field names) are changed. Furthermore, two or more field names can reference the same area of the buffer.

Examples

```
FIELD 1, 128 AS A$, 128 AS B$
```

This statement tells BASIC to assign the first 128 bytes of the buffer to the string variable A$ and the remaining 128 bytes to B$. If you now print A$ and B$, you will see the contents of the buffer. Of course, this value would be meaningless unless you have used GET to read a 256-byte record from disk.

Note: All data — both strings and numbers — must be placed into the buffer in string form. There are three pairs of functions (MKI$/CVI,MKS$/CVS,MKD$/CVD) for converting numbers to strings and vice-versa. See ''Functions'' below.

```
FIELD 3, 16 AS NM$, 25 AS AD$, 10 AS CY$, 2 AS ST$,7 AS ZP$
```

The first 16 bytes of buffer 3 are assigned the buffer name NM$; the next 25, AD$; the next 10, CY$; the next 2, ST$ and the next 7, ZP$. The remaining 196 bytes of the buffer are not fielded at all.

More on field names

Field names, like NM$,AD$,CY$,ST$, and ZP$, are not string variables in the ordinary sense. They do not consume the string space available to BASIC.

Instead, they point to the buffer field which you assigned with the FIELD statement. That's why you can use:

```
100 FIELD 1,255 AS A$
```

without worrying about whether 255 bytes of string space are available for A$.

If you use a buffer field name on the left side of an ordinary assignment statement, that name will no longer point to the buffer field; therefore, you won't be able to access that field using the previous field name.

For example,

```
A$=B$
```

nullifies the effect of the FIELD statement above (line 100).

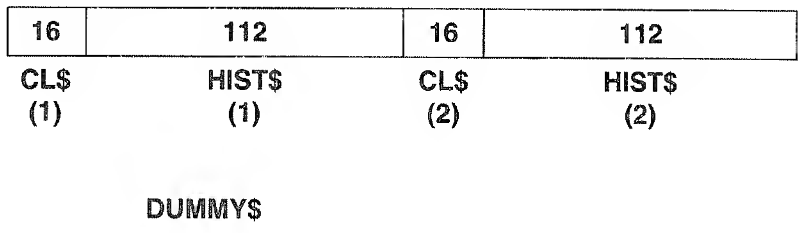
During random input, the GET statement places data into the 255-byte buffer, where it can be accessed using the field names assigned to that buffer. During random output, LSET and RSET place data into the buffer, so you can then PUT the buffer contents into a disk file.

Often you'll want to use a dummy variable in a FIELD statement to "pass over" a portion of the buffer and start fielding it somewhere in the middle. For example:

```
FIELD 1, 16 AS CLIENT$(1), 112 AS HIST$(1)
FIELD 1, 128 AS DUMMY$, 16 AS CLIENT$(2), 112 AS HIST$(2)
```

In the second FIELD statement, DUMMY$ serves to move the starting position of CLIENT$(2) to position 129. In this manner, two identical "subrecords" are defined on buffer number 1. We won't actually use DUMMY$ to place data into the buffer or retrieve it from the buffer.

The buffer now looks like this:



GET
Read a Record from Disk—Random Access

GET *nmexp1*[,*nmexp2*]

nmexp1 specifies a random access file buffer, *nmexp1*=1,2,...,15.

nmexp2 specifies which record to GET in the file; if omitted, the current record will be read.

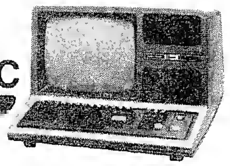
This statement gets a data record from a disk file and places it in the specified buffer. Before GETting data from a file, you must open the file and assign a buffer to it. That is, a statement like:

OPEN "R",*nmexp1*,*filespec*

is required *before* the statement:

GET *nmexp1*,*nmexp2*

GET tells BASIC to read record *nmexp2* from the file and place it into the *nmexp1* buffer. If you omit the record number in GET, BASIC will read the current record.



The "current record" is the record whose number is one higher than that of the last record accessed. The first time you access a file via a particular buffer, the current record is set equal to 1.

For example:

Program statement	**Effect**
1000 OPEN"R",1,"NAME/BAS"	Open NAME/BAS for random access using buffer 1
1010 FIELD 1,...	Structure buffer
1020 GET 1	GET record 1 into buffer 1
1025 REM...ACCESS BUFFER	
1030 GET 1,30	GET record 30 into buffer 1
1035 REM...ACCESS BUFFER	
1040 GET 1,25	GET record 25 into buffer 1
1046 REM...ACCESS BUFFER	
1050 GET1	GET record 26 into buffer 1

If you are using variable-length records (not fixed-length), an attempt to GET past the end of file will produce an error.

If you are using fixed-length records, the same attempt will return a null record and no error will occur. To prevent this from occurring, you can use the LOF function to determine the number of the highest numbered record.

PUT
Write a Record to Disk—Random Access

PUT *nmexp1*[,*nmexp2*]

nmexp1 specifies a random access file buffer, _nmexp_=1,2,...,15.

nmexp2 specifies the record number in the file, _nmexp2_ is the record you want to write. If _nmexp2_ is omitted, the current record number is assumed.

This statement moves data from a file's buffer into a specified place in the file. Before PUTting data in a file, you must:

1. OPEN the file, thereby assigning a buffer and defining the access mode (must be R);

2. FIELD the buffer, so you can
3. place data into the buffer with LSET and RSET statements.

When BASIC encounters the statement:

PUT *nmexp,nmexp2*

it does the following:

- Gets the information needed to access the disk file
- Checks the access mode for this buffer (must be R)
- Acquires more disk space for the file if necessary to accommodate the record indicated by *nmexp2*
- Copies the buffer contents into the specified record of the disk file
- Updates the current record number to equal *nmexp2 + 1*

The "current record" is the record whose number is one higher than the last record accessed. The first time you access a file via a particular buffer, the current record is set equal to 1.

If the record number you PUT is higher than the end-of-file record number, then *nmexp2* becomes the new end-of-file record number.

LSET and RSET
Place Data in a Random Buffer Field

LSET *var$* = *exp$* and RSET *var$*= *exp$*

&emsp;*var$* is a field name.

&emsp;*exp$* contains the data to be placed in the buffer field named by *var$*.

These two statements let you place character-string data into fields previously set up by a FIELD statement.

For example, suppose NM$ and AD$ have been defined as field names for a random file buffer. NM$ has a length of 18 characters, and AD$ has a length of 25 characters.

Now we want to place the following information into the buffer fields so it can be written to disk:

```
name:       JIM CRICKET, JR.
address:    2000 EAST PECAN ST.
```

This is accomplished with the two statements:

```
LSET NM$="JIM CRICKET,JR. "
LSET AD$="2000 EAST PECAN ST. "
```

This puts the data in the buffer as follows:

```
| JIM CRICKET,JR.        |      | 2000 EAST PECAN ST.          |
          NM$                                 AD$
```

Note that filler spaces were placed to the right of the data strings in both cases. If we had used RSET instead of LSET statements, the filler spaces would have been placed on the left. This is the **only** difference between LSET and RSET.

For example:

```
RSET NM$="JIM CRICKET,JR. "
RSET AD$="2000 EAST PECAN ST. "
```

places data in the fields as follows:

```
|      JIM CRICKET,JR.   |      |          2000 EAST PECAN ST. |
          NM$                                 AD$
```

If a string item is too large to fit in the specified buffer field, it is always truncated on the right. That is, the extra characters on the right are ignored.

CVD, CVI and CVS
Restore String to Numeric Form

CVD*(exp$)*

> ***exp$*** **defines an eight-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<8, an ILLEGAL FUNCTION CALL error occurs; if LEN*(exp$)*>8, only the first eight characters are used.**

CVI*(exp$)*

> ***exp$*** **defines a two-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<2, an ILLEGAL FUNCTION CALL error occurs; if LEN*(exp$)*>2, only the first two characters are used.**

CVS*(exp$)*

***exp$*** **defines a four-character string; *exp$* is typically the name of a buffer field containing a numeric string. If LEN*(exp$)*<4, an ILLEGAL FUNCTION CALL error occurs; if LEN*(exp$)*>4, only the first four characters are used.**

These functions let you restore data to numeric form after it is read from disk. Typically the data has been read by a GET statement, and is stored in a random-access file buffer.

The functions CVD, CVI, and CVS are inverses of MKD$, MKI$, and MKS$, respectively.

For example, suppose the name GROSSPAY$ references an eight-byte field in a random-access file buffer, and after GETting a record, GROSSPAY$ contains a MKD$ representation of the number 13123.38.

Then the statement:

```
PRINT CVD(GROSSPAY$)-TAXES
```

prints the result of the difference, 13123.38 – TAXES. Whereas the statement:

```
PRINT GROSSPAY$-TAXES
```

will produce a TYPE MISMATCH error, since string values cannot be used in arithmetic expressions.

Using the same example, the statement

```
A#=CVD(GROSSPAY$)
```

assigns the numeric value 13123.38 to the double-precision variable A#.

EOF
End-Of-File Detector

EOF*(nmexp)*

nmexp **specifies a file buffer, *nmexp*=1,2,...,15.**

This function checks to see whether all characters up to the end-of-file marker have been accessed, so you can avoid INPUT PAST END errors during sequential input.

Assuming *nmexp* specifies an open file, then EOF*(nmexp)* returns 0 (false) when the EOF record has not yet been read, and -1 (true) when it has been read.

Examples

```
IF EOF(5) THEN PRINT"END OF FILE"FILENM$
IF EOF(NM%) THEN CLOSE NM%
```

The following sequence of lines reads numeric data from DATA/TXT into the array A( ). When the last data character in the file is read, the EOF test in line 30 "passes," so the program branches out of the disk access loop, preventing an INPUT PAST END error from occurring. Also note that the variable I contains the number of elements input into array A( ).

```
5 DIM A(100) 'ASSUMING THIS IS A SAFE VALUE
10 OPEN "I",1, "DATA/TXT"
20 I%=0
30 IF EOF(1) THEN 70
40 INPUT#1,A(I%)
50 I%=I%+1
60 GOTO 30
70 REM PROGRAM CONTINUES HERE AFTER DISK INPUT
```

LOC
Get Current Record Number

LOC*(file number)*

***file number* is a numeric expression specifying the buffer for a currently-open file.**

LOC is used to determine the current record number, i.e., the number of the last record read since the file was opened. LOC is only valid after a GET.

Example

```
PRINT LOC(1)
```

Sample Program

```
1310 A$ = "WILLIAM WILSON"
1320 GET 1, X: X=X+1
1330 IF N$ = A$ THEN PRINT "FOUND IN RECORD" LOC(1): CLOSE:
     END
1340 GOTO 1320
```

This is a portion of a program. Elsewhere the file has been opened and fielded. N$ is a field variable. If N$ matches A$ the record number in which it was found is printed.

LOF
Get End-Of-File Record Number

LOF*(nmexp)*

nmexp specifies a random access buffer _nmexp_=1,2,...,15.

This function tells you the number of the last, i.e., highest numbered, record in a file. It is useful for both sequential and random access.

For example, during random access to a pre-existing file, you often need a way to know when you've read the last valid record. LOF provides a way.

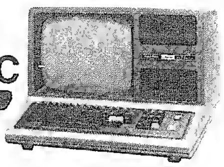
LOF is valid as soon as a previously created file is opened. If a file is extended, LOF is not valid until a GET is executed.

Examples:

```
10 OPEN "R",1,"UNKNOWN/TXT"
20 FIELD 1,255 AS A$
30 FORI%=1 TO LOF(1)
40 GET 1,I%
50 PRINT A$
60 NEXT
```

In line 30, LOF(1) specifies the highest record number to be accessed.

Note: If you attempt to GET record numbers beyond the end-of-file record, BASIC simply fills the buffer with hexadecimal zeros, and no error is generated.



When you want to add to the end of a file, LOF tells you where to start adding:

```
100 I%=LOF(1)+1 'HIGHEST EXISTING RECORD
110 PUT 1,I%    'ADD NEXT RECORD
```

MKD$, MKI$, and MKS$ Convert Data, Numeric-to-String

MKD$(*nmexp*)

> ***nmexp*** **is evaluated as a double-precision number.**

MKI$(*nmexp*)

> ***nmexp*** **is evaluated as an integer, −32768<=*nmexp*<32768; if *nmexp* exceeds this range, an ILLEGAL FUNCTION CALL error occurs. Any fractional component in *nmexp* is truncated.**

MKS$(*nmexp*)

> ***nmexp*** **is evaluated as a single-precision number.**

These functions change a number to a "string." Actually the byte values which make up the number are not changed; only one byte, the internal data-type specifier, is changed, so that numeric data can be placed in a string variable.

That is:

MKD$ returns an eight-byte string.
MKI$ returns a two-byte string.
MKS$ returns a four-byte string.

Examples

```
LSET TALLY$=MKI$(I%)
```

Field name TALLY$ would now contain a two-byte representation of the integer I%.

```
A$=MKI$(8/I)
```

A$ becomes a two-byte representation of the integer portion of 8/I. Any fractional portion is ignored. Note that A$ in this case is a normal string variable, not a buffer-field name.

Suppose BASEBALL/BAT (a non-standard file extension) has been opened for random access using buffer 2, and the buffer has been FIELDed as follows:

field:	NM$	YRS$	AVG$	HR$	AB$	ERNING$
length:	16	2	4	2	4	4

NM$ is intended to hold a character string; AVG$, AB$ and ERNING$, converted single-precision values; YRS$ and HR$, converted integers.

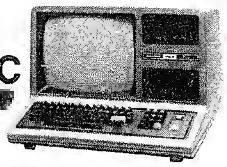
Suppose we want to write the following data record:

SLOW LEARNER played 38 years; lifetime batting average .123;
career homeruns, 11; at bats, 32768;...,earnings − 13.75.

Then we'd use the make-string functions as follows:

```
1000 LSET NM$="SLOW LEARNER"
1010 LSET YRS$=MKI$(38)
1020 LSET AVG$=MKS$(.123)
1030 LSET HR$=MKI$(11)
1040 LSET AB$=MKS$(32768)
1050 LSET ERNING$=MKS$(-13.75)
```

After this sequence, you can write SLOW LEARNER's information to disk with the PUT statement. When you read it back from disk with GET, you will need to restore the numeric data from string to numeric form, using CVI and CVS functions.



Methods of Access

Disk BASIC provides two means of file access:

- Sequential — in which you start reading or writing data at the beginning of a file; subsequent reads or writes are done at following positions in the file.
- Random — in which you start reading or writing at any record you specify. (Random access is also called direct access.)

Sequential access is stream-oriented; that is, the number of characters read or written can vary, and is usually determined by delimiters in the data. Random access is record-oriented; that is, data is always read or written in fixed-length blocks called records.

To do any input/output to a disk file, you must first open the file. When you open the file, you specify what kind of access you want:

- O for sequential output
- I for sequential input
- R for random input/output
- E (Extend) for sequential output starting at the end of file.

You also assign a file buffer for BASIC to use during file accesses. This number can be from 1 to 15, but must not exceed the number of concurrent files you requested when you started BASIC from TRSDOS. For example, if you started BASIC with 3 files, you can use buffer numbers 1, 2, and 3. Once you assign a buffer number to a file, you cannot assign that number to another file until you Close the first file.

Examples

```
OPEN "O", 1, "TEST"
```

Creates a sequential output file named TEST on the first available drive; if TEST already exists, its previous contents are lost. Buffer 1 will be used for this file.

```
OPEN "I", 2, "TEST"
```

Opens TEST for sequential input, using buffer 2.

```
OPEN "R", 1, "TEST"
```

Opens TEST for direct access, using buffer 1. If TEST does not exist, it will be created on the first available drive. Since record length is not specified, 256-byte records will be used.

```
OPEN "R", 1, "TEST", 40
```

Same as preceding example, but 40-byte records will be used.

```
OPEN "E", 1, "TEST"
```

Opens TEST sequentially for write and positions to EOF.

Sequential Access

This is the simplest way to store data in and retrieve it from a file. It is ideal for storing free-form data without wasting space between data items. You read the items back in the same order in which they were written.

There are several important points to keep in mind.

1. You must start writing at the beginning of the file. If the data you are seeking is somewhere inside, you have to read your way up to it.
2. Each time you Open a file for sequential output, the file's previous contents are lost, unless you use "E" instead of "O" for the mode.
3. To update (change) a sequential file, read in the file and write out the updated data to a *new* output file.
4. Data written sequentially usually includes delimiters (markers) to signify where each data item begins and ends. To read a file sequentially, you must know ahead of time the format of the data. For example: Does the file consist of lines of text terminated with carriage returns? Does it consist of numbers separated by blank spaces? Does it consist of alternating text and numeric information?
5. Sequential files are always written as ASCII-coded text, one byte for each character of data. For example, the number:

   ```
    1.2345
   ```
   requires 8 bytes of disk storage, including the leading and trailing blanks that are supplied. The text string:

   ```
   JOHNSON, ROBERT
   ```

   requires 15 bytes of disk storage.
6. Sequential files are always written with a record length of 256.

Sequential Output: An Example

Suppose we want to store a table of English-to-metric conversion constants:

English unit	Metric equivalent
1 inch	2.54001 centimeters
1 mile	1.60935 kilometers
1 acre	4046.86 sq. meters
1 cubic inch	0.01638716 liter
1 U.S. gallon	3.785 liters
1 liquid quart	0.9463 liter
1 lb (avoir)	0.45359 kilogram

First we decide what the data image is going to be. Let's say we want it to look like this:

english unit ▶ *metric unit, factor* (ENTER)

For example, the stored data would start out:

```
IN->CM, 2.54001   (ENTER)
```

The following program will create such a data file.

Note: X'0D' represents a carriage return.

```
10 OPEN "O",1,"METRIC/TXT"
20 FOR I%=1 TO 7
30      READ UNIT$, FACTR
40      PRINT#1, UNIT$; ","; FACTR
50 NEXT
60 CLOSE
70 DATA IN->CM, 2.54001, MI->KM, 1.60935, ACRE->SQ.KM,
   4046.86 E-6
80 DATA CU.IN->LTR, 1.638716E-2, GAL->LTR, 3.785
90 DATA LIQ.QT->LTR, 0.9463, LB->KG, 0.45359
```

Line 10 creates a disk file named METRIC/TXT, and assigns buffer 1 for sequential output to that file. The extension /TXT is used because sequential output always stores the data as ASCII-coded text.

Note: If METRIC/TXT already exists, line 10 will cause all its data to be lost. Here's why: Whenever a file is opened for sequential output, the end-of-file (EOF) is set to the beginning of the file. In effect, TRSDOS "forgets" that anything has ever been written beyond this point. To avoid this, you could use E instead of O in line 10.

Line 40 prints the current contents of UNIT$ and FACTR to the file. Since the string items do not contain delimiters, it is not necessary to print explicit quotes around them. The explicit comma is sufficient.

Line 60 closes the file. The EOF is at the end of the last data item, i.e., 0.45359, so that later, during input, BASIC will know when it has read all the data.

Sequential Input: An Example

The following program reads the data from METRIC/TXT into two "parallel" arrays, then asks you to enter a conversion problem.

```
5 CLEAR 500
10 DIM UNIT$(9), FACTR(9)    'allows for up to 10 data pairs
20 OPEN"I",1,"METRIC/TXT"
25 I%=0
30 IF EOF(1) THEN 70
40 INPUT#1, UNIT$(I%),FACTR(I%)
```

```
50 I%=I%+1
60 GOTO 30
70 CLOSE   '   Conversion factors have been read-in
100 CLS: PRINT TAB(5)"*** English to Metric Conversions ***"
110 FOR ITEM%=0 TO I%-1
120     PRINT TAB(9);USING"(## )   %          %   ";ITEM%,
    UNIT$(ITEM%)
130 NEXT
140 PRINT @ 704, "Which conversion (0-6)";
150 INPUT CHOICE%
160 INPUT"Enter English quantity";V
170 PRINT"The Metric equivalent is" V*FACTR(CHOICE%)
180 INPUT"Press <ENTER> to continue";X
190 PRINT @ 704, CHR$(31)   'clear to end of frame
200 GOTO 140
```

Line 20 opens the file for sequential input. Input begins at the beginning of the file.

Line 30 checks to see that the end-of-file record hasn't been reached. If it has, control branches from the disk input loop to the part of the program that uses the newly acquired data.

Line 40 reads a value into the string array UNIT$( ), and a number into the single-precision array FACTR( ). Note that this INPUT list parallels the PRINT# list that created the data file (see the section ''Sequential Output: An Example''). This parallelism is not required, however. We could just as successfully have used:

```
40 INPUT#1, UNIT$(I%): INPUT#1,FACTR(I%)
```

How to update a file

Suppose you want to add more entries into the English-Metric conversion file. You could simply re-Open the file with mode = E and PRINT# the extra data. Or, you might want to leave the old file intact and output a new file:

1. Open the file for sequential input (Mode = I)
2. Open another new data file for sequential output (Mode = O)
3. Input a block of data and update the data as necessary
4. Output the data to the new file
5. Repeat steps 3 and 4 until all data has been read, updated, and output to the new file; then go to step 6
6. Close both files

Sequential Line Input: An Example

Using the line-oriented input, you can write programs that edit other BASIC program files: renumber them, change LPRINTS to PRINTS, etc.—as long as these "target" programs are stored in ASCII format.

The following program counts the number of lines in any ASCII—format BASIC disk file with the extension /TXT.

```
10 CLEAR 300
20 INPUT"WHAT IS THE NAME OF THE PROGRAM"; PROG$
30 IF INSTR(PROG$,"/TXT")=0 THEN 110 'require /TXT extension
40 OPEN"I", 1, PROG$
50 I%=0
60 IF EOF(1) THEN 90
70 I%=I%+1: LINE INPUT#1, TEMP$
80 GOTO 60
90 PRINT PROG$" IS" I% "LINES LONG."
100 CLOSE: GOTO 20
110 PRINT "FILESPEC MUST INCLUDE THE EXTENSION '/TXT'"
120 GOTO 20
```

For BASIC programs stored in ASCII, each program line ends with a carriage return character not preceded by a line feed. So the LINE INPUT in line 70 automatically reads one entire line at a time, into the variable TEMP$. Variable I% actually does the counting.

To try out the program, first save any BASIC program using the A (ASCII) option (See SAVE). Use the extension /TXT.

Random Access Techniques

Random access offers several advantages over sequential access:

- Instead of having to start reading at the beginning of a file, you can read any record you specify.
- To update a file, you don't have to read in the entire file, update the data, and write it out again. You can rewrite or add to any record you choose, without having to go through any of the other records.
- Random access is more efficient—data takes up less space and is read and written faster.
- Opening a file for direct access allows you to write and read from the file via the same buffer.
- Random access provides many powerful statements and functions to structure your data. Once you have set up the structure, direct input/output becomes quite simple.

The last advantage listed above is also the ''hard part'' of direct access. It takes a little extra thought.

For the purposes of direct access, you can think of a disk file as a set of boxes —like a wall of post-office boxes. Just like the post office receptacles, the file boxes are numbered. We call these boxes ''records.''

You can place data in any record, or read the contents of any record, with statements like:

`PUT 1,5` write buffer-1 contents to record 5

`GET 1,5` read the contents of record 5 into buffer-1

In **Figure 14**, we assume a record length of 256.

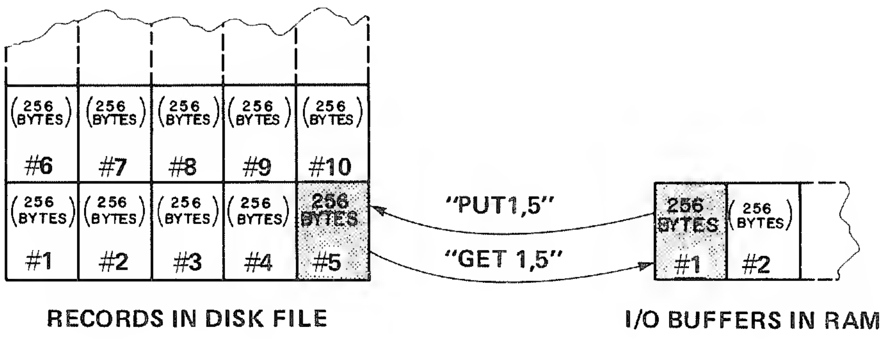
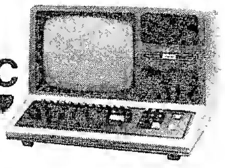


Figure 14. GET and PUT.



The buffer is a waiting area for the data. Before writing data to a file, you must place it in the buffer assigned to the file. After reading data from a file, you must retrieve it from the buffer.

As you can see from the sample PUT and GET statements above, data is passed to and from the disk in records. The size of each record is determined by an Open statement.

Storing Data in a Buffer

You must place the entire record into the buffer before putting its contents into the disk file.

This is accomplished by 1) dividing the buffer up into fields and naming them, then 2) placing the string or numeric data into the fields.

For example, suppose we want to store a glossary on disk. Each record will consist of a word followed by its definition. We start with:

```
100 OPEN"R", 1, "GLOSSARY/BAS"
110 FIELD 1, 16 AS WD$, 240 AS MEANING$
```

Line 100 opens a file named GLOSSARY/BAS (creates it if it doesn't already exist); and gives buffer 1 direct access to the file.

Line 110 defines two fields onto buffer 1:

WD$	consists of the first 16 bytes of the buffer;
MEANING$	consists of the last 240 bytes.

WD$ and MEANING$ are now **field-names**

What makes field names different? Most string variables point to an area in memory called the string space. This is where the value of the string is stored.

Field names, on the other hand, point to the buffer area assigned in the FIELD statement. So, for example, the statement:

```
10 PRINT WD$; ":"; MEANING$
```

displays the contents of the two buffer fields defined above.

These values are meaningless unless we first place data in the buffer. LSET, RSET and GET can all be used to accomplish this function. We'll start with LSET and RSET, which are used in preparation for disk output.

Our first entry is the word ''left-justify'' followed by its definition.

```
100 OPEN"R", 1, "GLOSSARY/BAS"
110 FIELD 1, 16 AS WD$, 240 AS MEANING$
120 LSET WD$="LEFT-JUSTIFY"
130 LSET MEANING$="TO PLACE A VALUE IN A FIELD FROM LEFT TO
    RIGHT; IF THE DATA DOESN'T FILL THE FIELD, BLANKS ARE
    ADDED ON THE RIGHT; IF THE DATA IS TOO LONG, THE EXTRA
    CHARACTERS ON THE RIGHT ARE IGNORED. LSET IS A LEFT-
    JUSTIFY FUNCTION."
```

Line 120 left-justifies the value in quotes into the first field in buffer 1. Line 130 does the same thing to its quoted string.

Note: RSET would place filler-blanks to the *left* of the item. Truncation would still be on the right.

Now that the data is in the buffer, we can write it to disk with a simple PUT statement:

```
140 PUT 1,1
150 CLOSE
```

This writes the first record into the file GLOSSARY/BAS.

To read and print the first record in GLOSSARY/BAS, use the following sequence:

```
160 OPEN"R", 1, "GLOSSARY/BAS"
170 FIELD 1, 16 AS WD$, 240 AS MEANING$
180 GET 1,1
190 PRINT WD$: PRINT MEANING$
200 CLOSE
```

Line 160 and 170 are required only because we closed the file in line 150. If we hadn't closed it, we could go directly to line 180.

Random Access: A General Procedure

The previous example shows the necessary sequences to read and write using random access. But it does not demonstrate the primary advantages of this form of access—in particular, it doesn't show how to update existing files by going directly to the desired record.

The program below, GLOSSACC/BAS, develops the glossary example to show some of the techniques of random access for file maintenance. But before looking at the program, study this general procedure for creating and maintaining files via random access.

Step	See GLOSSACC/BAS, Line Number
1. Open the file	110
2. Field the buffer	120
3. Get the record to be updated	140
4. Display current contents of the record (use CVD, CVI, CVS before displaying numeric data)	145-170
5. LSET and RSET new values into the fields (use MKD$, MKI$, MKS$ with numeric data before setting it into the buffer)	210-230
6. PUT the updated record	240
7. To update another record, continue at step 3. Otherwise, go to step 8.	250-260
8. Close the file	270

```
10 REM ...... GLOSSACC/BAS ...
100 CLS : CLEAR 300
110 OPEN "R", 1, "GLOSSARY/BAS"
120 FIELD 1, 16 AS WD$, 238 AS MEANING$, 2 AS NX$
130 INPUT "WHAT RECORD DO YOU WANT TO ACCESS"; R%
140 GET 1, R%
145 NX% = CVI(NX$)    'SAVE LINK TO NEXT ALPHABETICAL ENTRY
150 PRINT "WORD :     "WD$
160 PRINT "DEF'N : " : PRINT MEANING$
170 PRINT "NEXT ALPHABETICAL ENTRY: RECORD #:" NX% : PRINT
180 W$ = "" : INPUT "TYPE NEW WORD <ENTER> OR <ENTER> IF OK";
    W$
190 D$ = "" : PRINT "TYPE NEW DEF'N <ENTER> OR <ENTER> IF
    OK?" : LINE INPUT D$
200 INPUT "TYPE NEW SEQUENCE NUMBER OR <ENTER> IF OK"; NX%
210 IF W$ <> "" THEN LSET WD$ = W$
220 IF D$ <> "" THEN LSET MEANING$ = D$
```

```
230 LSET NX$ = MKI$ (NX%)
240 PUT 1, R%
245 R% = NX% 'USE NEXT ALPHA. LINK AS DEFAULT FOR NEXT RECORD
250 CLS : PRINT " TYPE <ENTER> TO READ NEXT ALPHA. ENTRY,":
    PRINT" OR RECORD # <ENTER> FOR SPECIFIC ENTRY,": INPUT "
    OR 0 <ENTER> TO QUIT"; R%
260 IF 0<R% THEN 140
270 CLOSE
280 END
```

Notice we've added a field, NX$, to the record (line 120). NX$ will contain the number of the record which comes next in alphabetical sequence. This enables us to proceed alphabetically through the glossary, provided we know which record contains the entry which should come first.

For example, suppose the glossary contains:

record#	word (WD$)	defn,	pointer to next alpha. entry (NX$)
1	LEFT-JUSTIFY	...	3
2	BYTE	...	4
3	RIGHT-JUSTIFY	...	0
4	HEXADECIMAL	...	1

When we read record 2 (BYTE), it tells us that record 4 (HEXADECIMAL) is next, which then tells us record 1 (LEFT-JUSTIFY) is next, etc. The last entry, record 3 (RIGHT-JUSTIFY), points us to zero, which we take to mean "The End."

Since NX$ will contain an integer, we have to first convert that number to a two-byte string representation, using MKI$ (line 230 above).

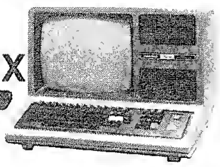
The following program displays the glossary in alphabetical sequence:

```
300 REM ... GLOSSOUT/BAS ...
310 CLS : CLEAR 300
320 OPEN "R", 1, "GLOSSARY/BAS"
330 FIELD 1, 16 AS WD$, 238 AS MEANING$, 2 AS NX$
340 INPUT "WHICH RECORD IS FIRST ALPHABETICALLY"; N%
350 GET 1, N%
360 PRINT : PRINT WD$
370 PRINT MEANING$
380 N% = CVI(NX$)
390 INPUT "PRESS <ENTER> TO CONTINUE"; X
400 IF N% <> 0 THEN 350
410 CLOSE
420 END
```

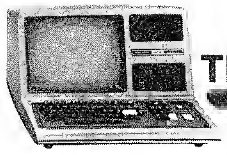
Disk BASIC Error Codes/Messages

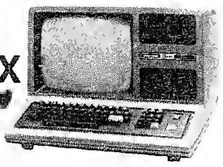
51	Field overflow
52	Internal error
53	Bad file number
54	File not found
55	Bad file mode
58	Disk I/O error
62	Disk full
63	Input past end
64	Bad record number
65	Bad file name
67	Direct statement in file
68	Too many files
69	Disk write-protect
70	File access

Note: Disk errors cannot be simulated via the ERROR statement.



Index





Figures and Tables

Radio Shack Software License

The following are the terms and conditions of the Radio Shack Software License for copies of Radio Shack software either purchased by the customer, or received with or as part of hardware purchased by customer:

A. Radio Shack grants to CUSTOMER a personal, non-exclusive, paid up license to use the Radio Shack computer software programs received. Title to the media on which the software is recorded (cassette and/or disk) or stored (ROM) is transferred to the CUSTOMER, but not title to the software.

B. In consideration for this license, CUSTOMER shall not reproduce copies of such software programs except to produce the number of copies required for personal use by CUSTOMER (if the software allows a backup copy to be made), and to include Radio Shack's copyright notice on all copies of programs reproduced in whole or in part.

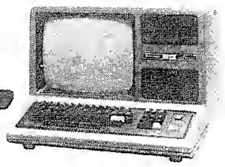
C. CUSTOMER may resell Radio Shack's system and applications software (modified or not, in whole or in part), provided CUSTOMER has purchased one copy of the software for each one resold. The provisions of this Software License (paragraphs A, B, and C) shall also be applicable to third parties purchasing such software from CUSTOMER.

Important Note

All Radio Shack computer programs are licensed on an "as is" basis without warranty.

Radio Shack shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by Radio Shack, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

Good data processing procedure dictates that the user test the program, run and test sample sets of data, and run the system in parallel with the system previously in use for a period of time adequate to insure that results of operation of the computer or program are satisfactory.



Service Policy

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

LIMITED WARRANTY

For a period of 90 days from the date of delivery, Radio Shack warrants to the original purchaser that the computer hardware unit shall be free from manufacturing defects. This warranty is only applicable to the original purchaser who purchased the unit from Radio Shack company-owned retail outlets or duly authorized Radio Shack franchisees and dealers. This warranty is voided if the unit is sold or transferred by purchaser to a third party. This warranty shall be void if this unit's case or cabinet is opened, if the unit has been subjected to improper or abnormal use, or if the unit is altered or modified. If a defect occurs during the warranty period, the unit must be returned to a Radio Shack store, franchisee, or dealer for repair, along with the sales ticket or lease agreement. Purchaser's sole and exclusive remedy in the event of defect is limited to the correction of the defect by adjustment, repair, replacement, or complete refund at Radio Shack's election and sole expense. Radio Shack shall have no obligation to replace or repair expendable items.

Any statements made by Radio Shack and its employees, including but not limited to, statements regarding capacity, suitability for use, or performance of the unit shall *not* be deemed a warranty or representation by Radio Shack for any purpose, nor give rise to any liability or obligation of Radio Shack.

EXCEPT AS SPECIFICALLY PROVIDED IN THIS WARRANTY OR IN THE RADIO SHACK COMPUTER SALES AGREEMENT, THERE ARE NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS OR BENEFITS, INDIRECT, SPECIAL, CONSEQUENTIAL OR OTHER SIMILAR DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR OTHERWISE.

7-80

RADIO SHACK **A DIVISION OF TANDY CORPORATION**

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U.K.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

8749167-581-SP

PRINTED IN U.S.A.