TRS-80™ Model III

Disk System Owner's Manual

(Preliminary Version)

Radio Shack®
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

SOFTWARE REGISTRATION CARD

IMPORTANT: In order that you can receive notification of modifications or updates of this program you MUST complete this card and return it immediately. This card gets you information only and is NOT a warranty registration.

Name _____

Company _____

Address _____

City _____

State _____ **Zip** _____

12010181
Version/Date

Cat. No. 26-1063

Purchase Date _____

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 _____

12010181
Version/Date

Cat. No. 26-1063

NEW ADDRESS:

Name _____

Company _____

Address _____

City _____

State _____ **Zip** _____

OLD ADDRESS:

Name _____

Company _____

Address _____

City _____

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.**

PLACE
STAMP
HERE

Computer Merchandising
P.O. Box 2910
Fort Worth, Texas 76102

Attn: Software Registration

PLACE
STAMP
HERE

Computer Merchandising
P.O. Box 2910
Fort Worth, Texas 76102

Attn: Software Registration

Radio Shack
A Division of Tandy Corporation

RESEARCH AND DEVELOPMENT 817-390-3011
1000 TWO TANDY CENTER, FORT WORTH, TEXAS 76102

September, 1980

To Our Customers

To make your new TRS-80 Model III Disk System available as quickly as possible, we've assembled this preliminary manual so you can begin using your Computer system right away.

If you'll fill out and return the enclosed claim card, we'll send the permanent Disk System Owner's Manual to you as soon as it's available.

Addendum for the Model III Disk System Owner's Manual (Preliminary Version)

Please add the following notes and changes to your manual.

PAGE 111-115, CONVERT UTILITY. Here is a suggestion to simplify the transfer of your Model I TRSDOS files onto a Model III TRSDOS diskette.

Using a Model I Disk System, remove all passwords from the diskette to be converted. You can do this with the PROT command, as described in the Model I TRSDOS/Disk BASIC Owner's Manual, page 4-23.

Now that the passwords are removed, you may use the Model III CONVERT utility and then use PROT or ATTRIB to restore password protection, as described on pages 43 and 93 of the Model III Disk Manual.

Converting Data Diskettes with a Two-Drive System

With a two-drive system, you must convert the Model I files onto a Model III TRSDOS "full system" in Drive 0. (To make sure there is enough room for all the files, you may want to delete all non-system files from the Model III system diskette in drive 0.)

After you have completed the convert/copy process, you may copy the files onto a Model III TRSDOS "data diskette" in drive 1.

FORMAT Utility

At the beginning of the formatting process, there is a delay of about 30 seconds while TRSDOS searches for data on the destination diskette. After this delay, TRSDOS will begin initializing the diskette tracks.

Page 33. General Notes on Entering TRSDOS Commands

In TRSDOS commands which contain file specifications or option lists, there must be exactly one blank space after the command and after each file name.

Examples

NOTE: In the following examples, "~" represents a mandatory single blank space.

```
CLOCK~(OFF)

ATTRIB~PAYROLL/BAS.SECRET~(N,ACC=,UPD=PETE,PROT=READ)

COPY~FILE/A~FILE/B
```

"Wild-Card" File Name Specifications in COPY and KILL commands

COPY and KILL can find all files with a specified extension. This is called a wild-card capability. To use it, specify the extension only. TRSDOS will find and use all files with the specified extension, regardless of the file name. (In the COPY command, you must also include a destination drive number after the wild-card specification.)

The command:

```
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.

The command:

```
KILL~/TXT:0
```

Tells TRSDOS to kill all Drive 0 files which have the extension /TXT.

PAGES 43-44, ATTRIB COMMAND. 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 is 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.

Page 60. DEBUG can only be used on programs in the user area, X'5600' to TOP (decimal 22016 to TOP).

Page 71. In the directory ATRB column, fourth character:

0 = Full access
1 = Kill and all privileges below
2 = Rename and all privileges below
3 = This designation is not used
4 = Write and all privileges below
5 = Read and all privileges below
6 = Execute only
7 = No access

In the number of records and extents columns, a zero (not an asterisk) indicates none have been allocated.

Page 76. In the DUMP command syntax, the START address must be greater than X'6FFF'.

Page 99. In the ROUTE syntax, there must be a comma after the 'SOURCE=aa' field:

ROUTE~(SOURCE=aa,DESTIN=bb)

Page 126. Change line 3.

Was:	CALL	44364
Change to:	CALL	4436H

Pages 127 - 134. Change the jump vector addresses for PUTEXT, BACKSPACE, POSEOF, DIVIDE, DMULT, RAMDIR, FILPTR.

Routine	Jump Vector Decimal	Hexadecimal
PUTEXT	17483	444B
BACKSPACE	17477	4445
POSEOF	17480	4448
DIVIDE	17489	4451
DMULT	17486	444E
RAMDIR	17040	4290
FILPTR*	17037	428D

* In FILPTR, the contents of register A is not significant on entry to the routine. ("A = 424C" is an error.)

Page 135. Supplementary Information. Change addresses.

Item (2). X'4225' contains the address of the 64-byte buffer which contains the last command entered.

Item (3). The correct call address for the time of day is X'3036'. The correct call address for the date is X'3033'.

Additional Information:

Address X'4411' contains the address of the end of RAM.

New System Calls

DSPDIR X'4419'
Display Directory

To display the directory of non-protected user files, set up the entry conditions and execute a call to DSPDIR.

On entry, X'4271' should contain the ASCII-coded drive number, "0", "1", "2", or "3".

On exit, all registers are changed.

COMDOS X'4299'
Execute a TRSDOS Command and Jump to TRSDOS READY

Set up the entry conditions and execute a call to COMDOS.

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

CMDDOS X'429C'
Execute a TRSDOS Command and Return to Caller

Set up the entry conditions and execute a call to CMDDOS.

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

On exit, all registers are changed.

CMD "R" and CMD"T" (pages 144, 164, 167) switch the clock display on and off, respectively. They do not turn the internal clock on or off.

Page 56. The CREATE command fills each allocated sector with binary zeroes. 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.

Page 160. The syntax for CMD "O" is:

```
CMD "O", x, array(start)
```

where 'x' is an integer variable containing the number of items to be sorted, and 'array(start)' specifies an array element. The array contains the data to be sorted, and 'start' is the subscript indicating where the sort should start.

The array must be one-dimensional, string type. The string elements in the array may be of any length.

Page 170. Change the syntax for the CMD "Z" command.

```
CMD "Z", switch
```

where 'switch' is a string expression containing either of two values: ON or OFF. If 'switch' is a constant, it must be enclosed in quotation marks.

For example:

```
CMD "Z", "ON"
```

turns dual routing on.

```
CMD "Z", "OFF"
```

turns it off.

BASIC Command to Disable/Enable <BREAK>

```
CMD "B", switch
```

where 'switch' is a string expression containing either of two values: ON or OFF. If 'switch' is a constant, it must be enclosed in quotation marks.

This command lets you protect a program from being interrupted by the <BREAK> key. After the command,

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

the <BREAK> key will be ignored unless it is pressed during cassette or printer output or serial input/output.

The BREAK key will remain disabled after the program has ended. It will be re-enabled when you return to TRSDOS via CMD "S" or CMD "I".

BASIC Program Renumbering Command

```
NAME newline, startline, increment
```

where '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 an entire program: 10, 20, 30, ...

```
NAME 6000,5000,100
```

Renumbers all lines numbered from 5000 up; the first renumbered

line will become 6000, and an increment of 100 will be used between subsequent lines.

BASIC Disk File Access

PAGE 121. When you start Disk BASIC, you are asked,

HOW MANY FILES?

In addition to determining the number of files open at once, this question lets you select either 256-byte records or variable-length* records. If you select variable-length records, you determine the record length of each file when the file is opened.

* In this manual, the term "variable-length" simply means the length is set when the file is opened, and may be from 1 to 256. For any given file, each record in the file has the same length.

If you want to use 256-byte records in all files, type in the desired number of files and press <ENTER>. To use files with record lengths from 1 to 256, add the letter "V" immediately after the number of files (no comma).

PAGE 192. OPEN Statement. To the list of values for 'expl$', add the following:

expl$=	access mode
E	(Extend). Sequential output, starting at the end of file.

Use this access mode to add to the end of an existing sequential file. For example,

```
OPEN "E", 1, "TEST"
```

Opens the file TEST for sequential output. Then first PRINT # statement will add data to the end of the file.

PAGE 206. FIELD Statement. The sum of all the field-lengths should equal the record length of the file (256 unless you are using variable-length files). The sum must not exceed the record length.

PAGE 94. PURGE Command. On system diskettes, there are certain system files that do not appear in the directory listing. To convert a system diskette into a data diskette, you may use a special form of PURGE:

```
PURGE * (options)
```

The asterisk tells TRSDOS to ask you if you want to delete the system files. The options are as defined in the syntax block. If you delete the system files, the diskette may no longer be used in drive 0.

TRS-80™

TRS-80 Model III

Disk System Owner's Manual

(Preliminary Version)

Radio Shack®

Contents

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 (TM), 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 184,320 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 TRSDOS diskette.

Model III Manuals

There are four 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 Quick Reference Card

For Disk Operation:

This Disk Manual supplements the Model III Manual. Use it 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 exprienced 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.

Part I OPERATION

Installation

First set up the Computer according to the instructions in Section 1, Chapter 2 of 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, and you can proceed to the Operation section in this manual.

If you have a three- or four-drive system, you need to connect the external drives. Refer to the disk drive owner's manual for specific instructions.

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

Notice that the 26-1164 drive may be used as system 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. 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.

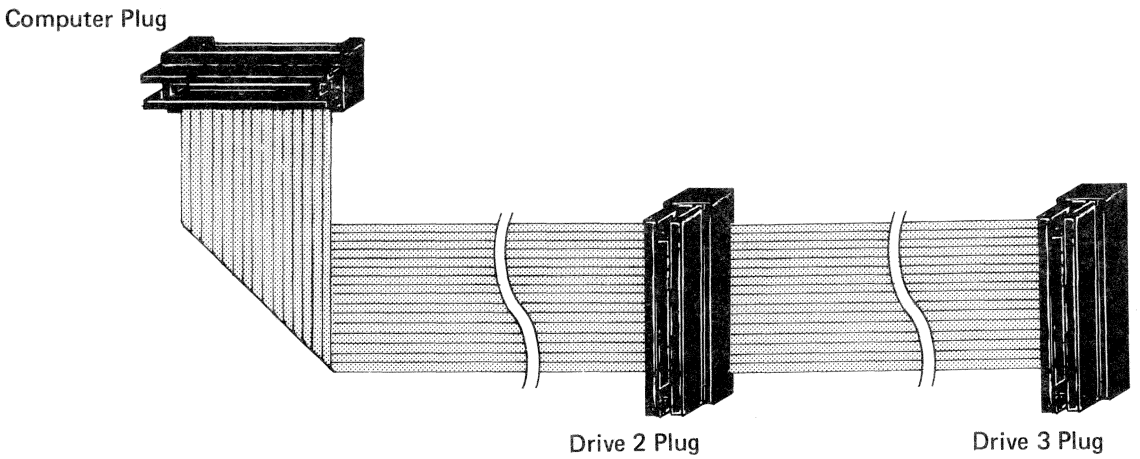
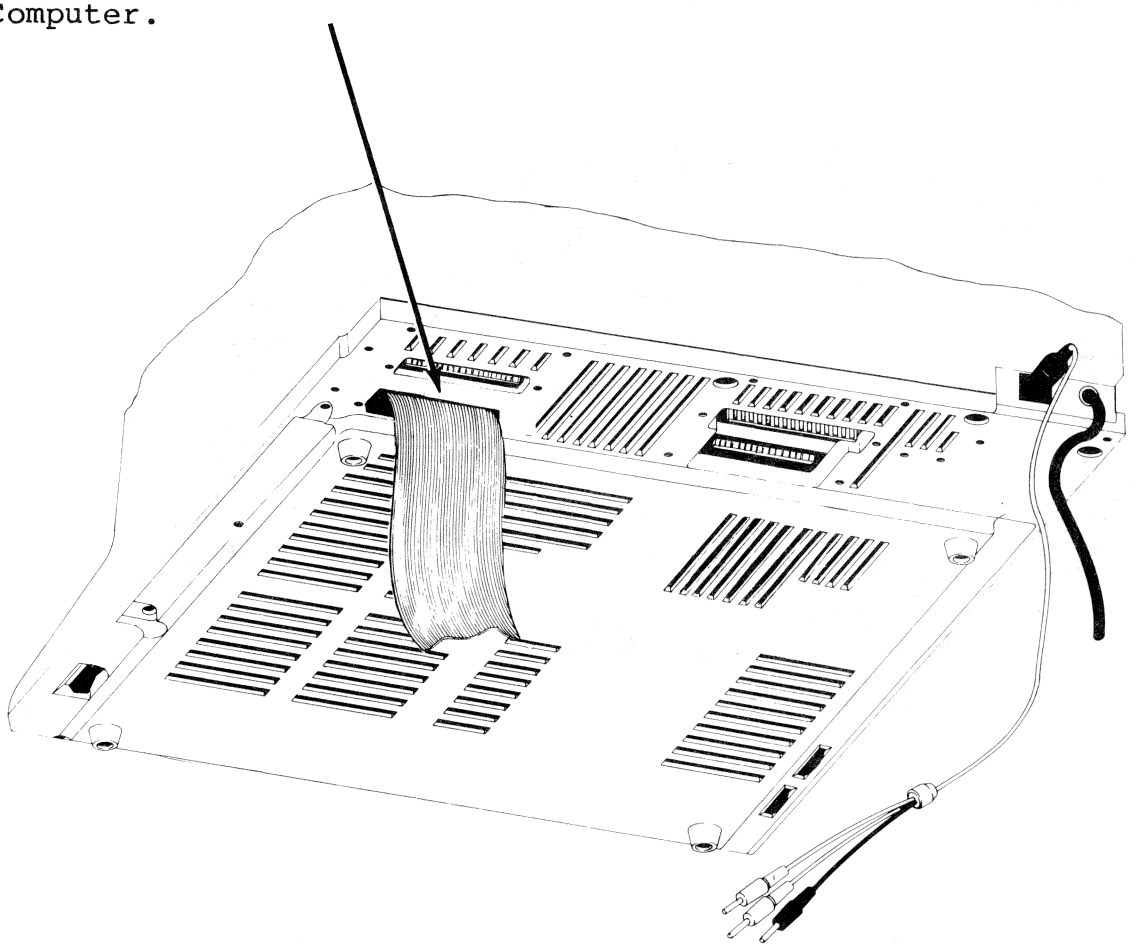


Figure 1. External Disk Cable with Plugs Labeled.

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.



Operation

First, take a few minutes to become familiar with the various elements of your Disk System. Refer to Figures 3 and 4. This is very important. If you try to use the Computer without having a little background information, you could damage a diskette.

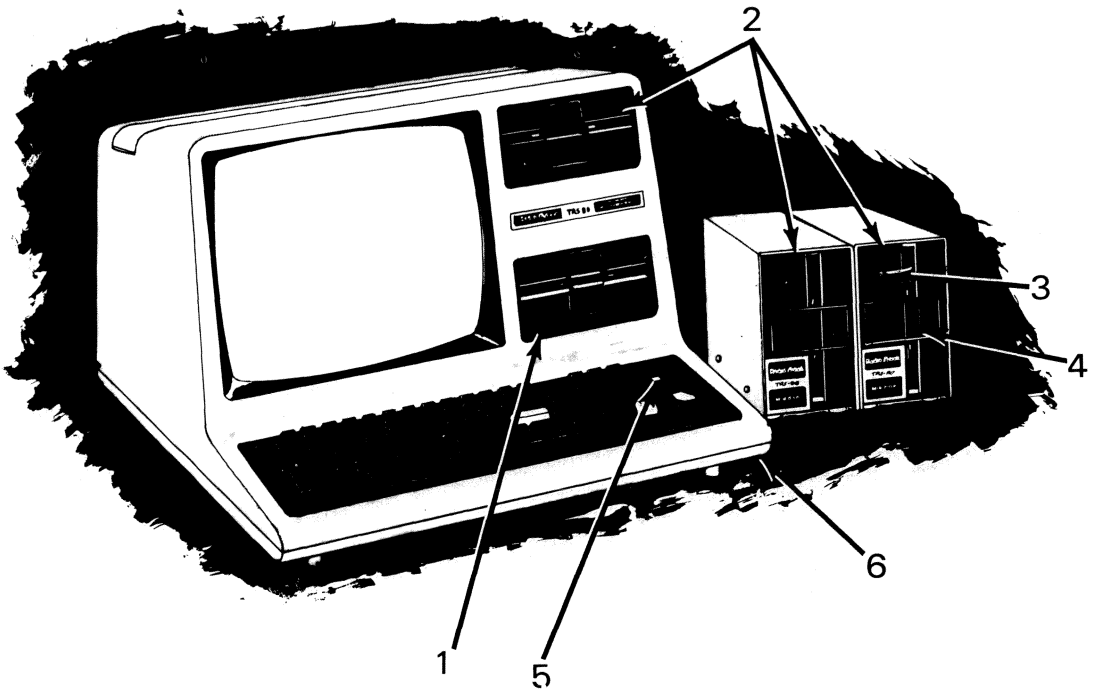


Figure 3. The Model III Disk System with Disk Expansion Unit (optional/extra) (see the following page for detailed explanation).

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.

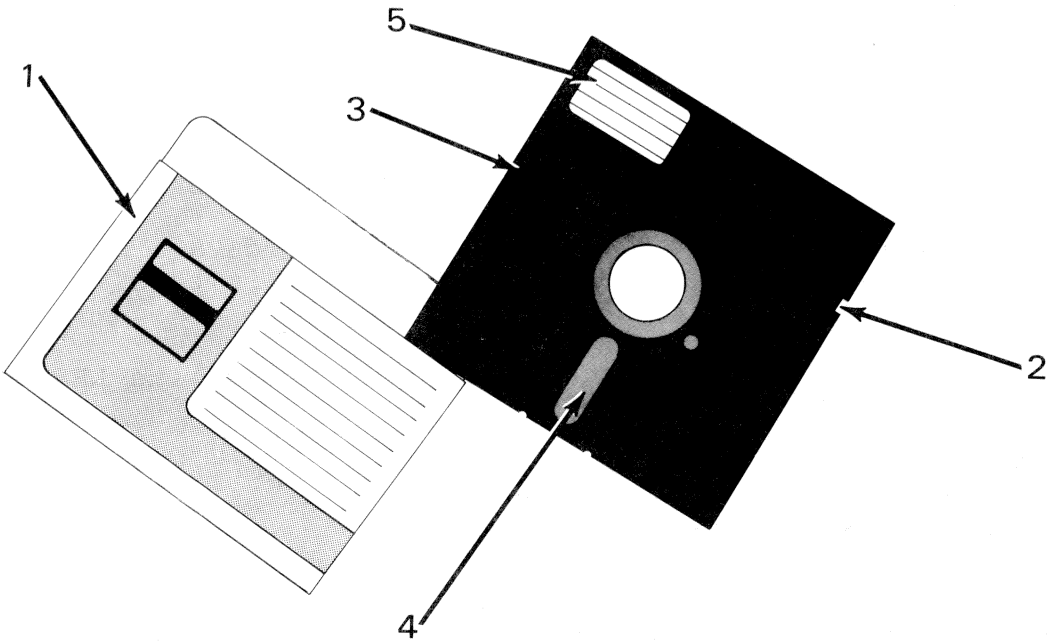


Figure 4. A Diskette. (Catalogue Number 26-0305 or 26-0405)

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. 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.

Care of 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.

Starting the System

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 5.

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.

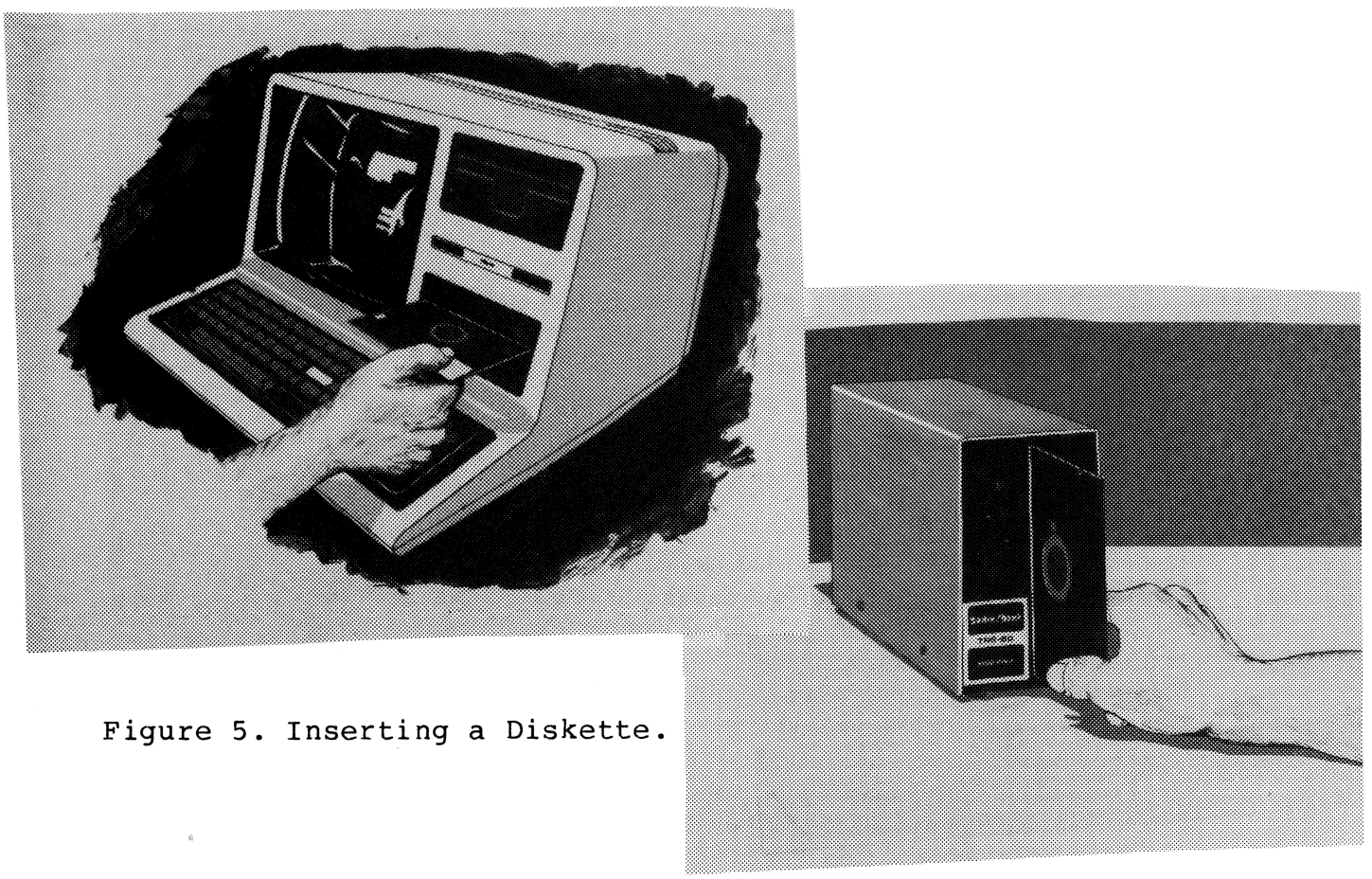


Figure 5. Inserting a Diskette.

If nothing happens on the display, or if the message:

Diskette?

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?

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,

DOS Ready

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

Important Disk Operations

In this section we will describe two very important operations:

1. Duplicating the TRSDOS diskette (BACKUP)
2. Initializing a data diskette (FORMAT)

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 in Chapter 9; 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. (BACKUP will actually copy any diskette, not just a TRSDOS diskette.)

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 the backup process.

2. Start TRSDOS as explained in the previous section. DOS READY should be displayed.

3. Type:

    BACKUP <ENTER>

3. TRSDOS will now load and start the backup program. It will ask you:

    SOURCE Drive Number?

You should specify the drive which contains the original TRSDOS diskette by typing:

    0 <ENTER>.

4. Next TRSDOS will ask:

    DESTINATION Drive Number?

Now specify the drive which will be used for making the duplicate TRSDOS.

4-A. If you have two or more drives in your system, type:

1 <ENTER>

4-B. If you have a single-drive system, type:

Ø <ENTER>

5. TRSDOS will ask:

SOURCE Disk Master Password?

Type:

PASSWORD <ENTER>

(PASSWORD is the password of the supplied diskette.)

6. Now the backup 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 sucessful backup operation, TRSDOS will display the message:

Insert SYSTEM Diskette <ENTER>

Be sure you have a TRSDOS diskette in drive 0, then press <ENTER>.

The backup is now complete. We suggest you save the original TRSDOS and use the duplicate as your working copy. If anything

should happen to the working copy, you can make another one from the original.

Making a Data Diskette

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. By necessity, 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 will be lost. The resultant diskette contains no system files and may only be used in drive 1, 2 or 3.

1. In the DOS READY mode, type:

    FORMAT <ENTER>

2. TRSDOS will start the formatter program and ask you a series of questions:

    Which Drive is to be Used?

Insert a blank diskette into drive 1. Type:

    1 <ENTER>

If the diskette is already formatted, TRSDOS will warn you:

    Diskette contains DATA, Use or Not?

The warning is needed since formatting a diskette erases all previous informatin from the diskette. Type N <ENTER> to cancel the format operation; type Y <ENTER> to continue it.

Diskette Name?

(TRSDOS is the name of the supplied diskette.)

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?

(PASSWORD is the password of the supplied diskette.)

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 diskette backups and total 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!

3. TRSDOS will now format and verify the diskette.

4. Upon completion, TRSDOS will display the message:

Insert SYSTEM Disk (ENTER)

The TRSDOS diskette should already be in drive 0, so simply press <ENTER>.

The data diskette is now ready for use in either drive 1, 2 or 3.

Quick Instruction 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 the program in a disk file
4. Loading it from the disk file

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

Starting Disk BASIC

Under DOS 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. For a sample programming session, you may use the one in the Model III Manual, Section 1, page 3/8.

After you have typed in and run this or some other program, you are ready to save it in a disk file.

Note: You are automatically in High Baud when you enter Disk BASIC. If you need Low Baud, use the following command sequence:

```
PATCH BASIC/CMD (ADD=5202,FIND=00,CNG=FF) <ENTER>
```

Consequently, you will be prompted with:

```
Cass?
```

whenever you enter Disk BASIC.

You should then enter either H (High) or L (Low) to choose the Baud you need.

You should note, however, that the system diskette has been altered and you will be prompted whenever you enter Disk BASIC (except when you enter BASIC *).

To change the system diskette back to its original state (i.e., automatically in High baud), simply enter the PATCH again but reverse the FIND and CHG values.

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 it 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.

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, Section 1, page 13/1.

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 both 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 ellipsis, 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 or 'lowercase within single-quotes'

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 (defined later) after LIST.

... (ellipses)

Indicates that the preceding items can be repeated. For example:

```
ATTRIB filename (option,...)
```

indicates that several options may be repeated inside the parenthesis.

↵

This special symbol is used occasionally to indicate a

blank-space character (ASCII code 32 decimal, 20 hexadecimal).

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).

Specifications

Diskettes	5 1/4" mini-diskettes Radio Shack Cataloge Number 26-305 or 26-405 (pkg of three)
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

Part II TRSDOS

PART II -- TRSDOS

General Information

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 "DOS 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 6 illustrates the relationships between these three roles:

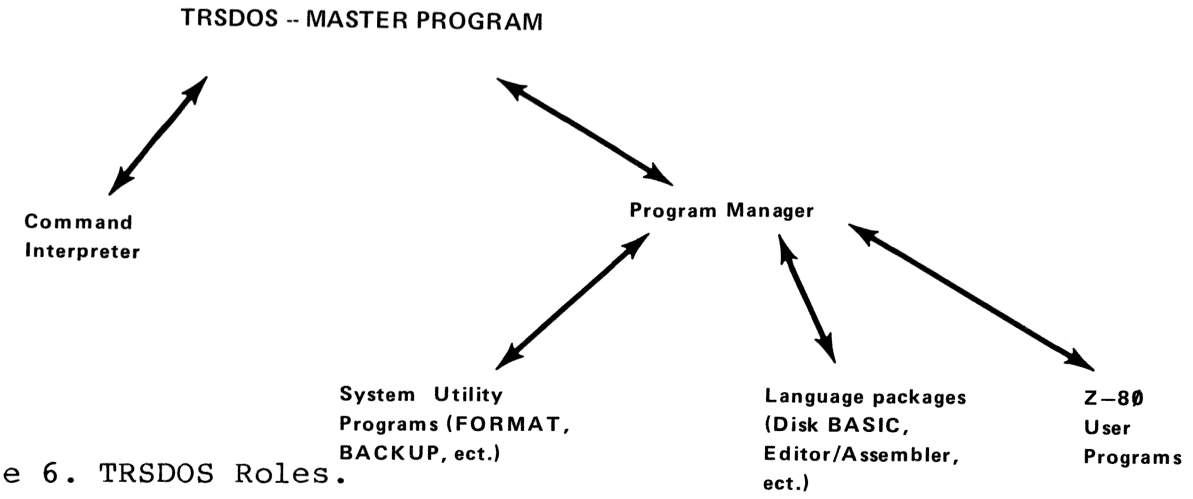


Figure 6. TRSDOS Roles.

Where Does BASIC Fit In?

If you refer to Figure 6, 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 the 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 from the beginning.

In the disk system, however, TRSDOS is in control when you start-up. You then 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. Its "resident" module consists of ~input/output drivers~, tables, the ~command interpreter~, and other essential routines.

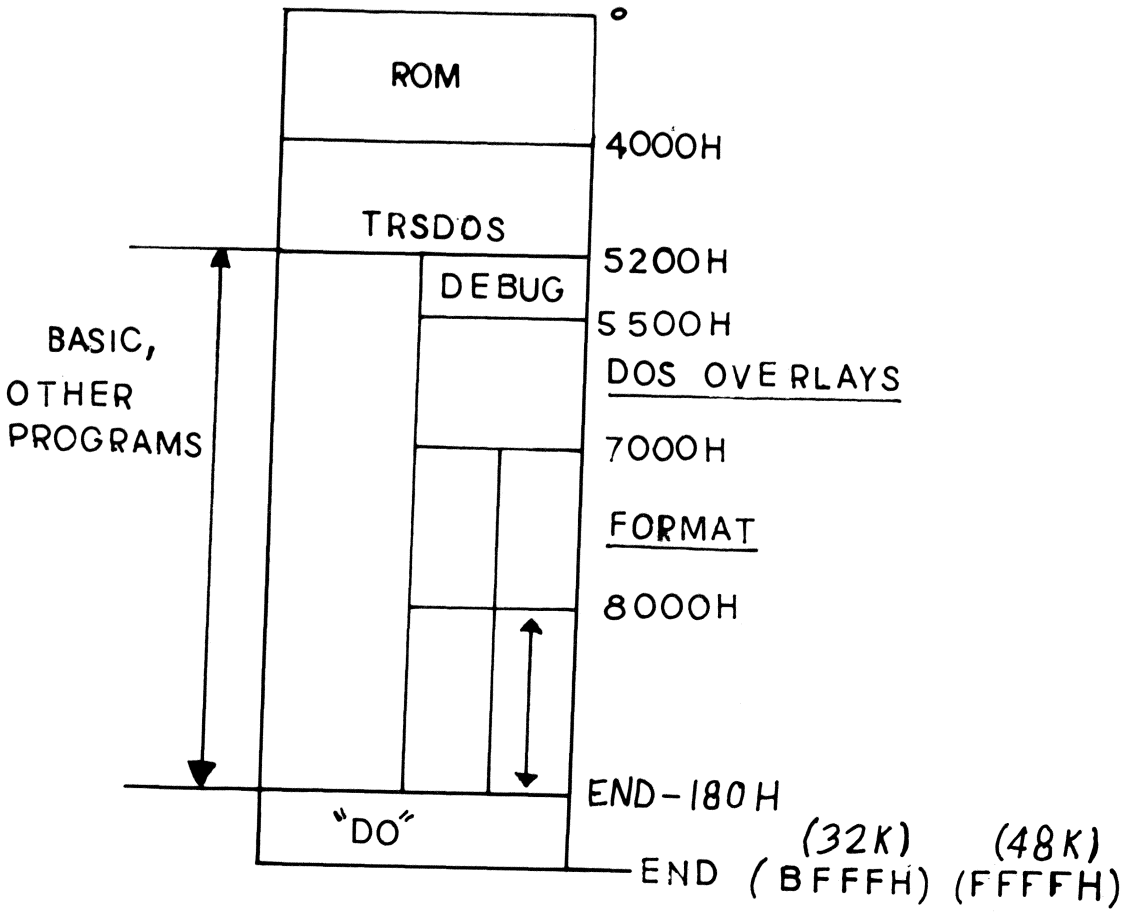
Other modules are loaded as needed, and replaced (or "overlayed") by still other modules when they are no longer needed. These are referred to as "overlays".

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

For specific details on the modules and their use of RAM, see ~Technical Information~.

The Memory Map on the following page illustrates how TRSDOS utilizes the available memory space.

Figure 7. TRSDOS Memory Map



Using TRSDOS

Entering a Command

Whenever the TRSDOS prompt,

```
DOS 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>
```

and TRSDOS will clear the Display and the DOS 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:

- First it checks to see if what you've typed is the name of a TRSDOS command. If it is, TRSDOS executes it immediately...
- ...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 (such as filename/CMD).
- 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 (described later)--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 (you might compare it to the grammar or structure of an English sentence). The syntax tells you how to put keywords (such as DIR, LIST, CREATE, etc.) together with the necessary parameters and punctuation for each keyword.

In this book, we present general syntax inside shaded boxes, so they're easy to recognize.

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

```
HELP command
```

'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 (see Library Commands).

Type of 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 paratheses around options must be included unless the option itself is omitted.

'comment' is an optional file used to document the purpose of the command-line. Comments are useful inside automatic command input files (see BUILD and DO commands).

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 one of the following:
a blank space or spaces.
a comma.
TO surrounded by blank spaces.

'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.** (Just keep in mind that a disk is nothing more than a file cabinet, only smaller.)

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 particular file specification. The password does not. It simply controls access to the file.

File Names

A filename 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	INVNTORY	DATA11/BAS
NAMES/A12	AUGUST/A15	WAREHOUS
TEST	TEST1	TEST1/A

are all valid and distinct filenames.

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-lanuage Command file
/REL	for a Relocatable machine-language program
/DAT	for data

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

Another advantage is that TRSDOS can recognize certain extensions as having a certain function. For example, if a file has the extension /CMD, then 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 3 until it finds the file.

Anytime you omit a drive-specification, TRSDOS will follow this sequence.

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 to look for and 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 lookup sequence. When TRSDOS has to write a file, it will skip over any WP as well as any notch-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 simply by referring to the filename--he must also use the appropriate password for that file.

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 the user total access to the information

- an "Access word", which grants the user limited access to the information (see ATTRIB)

When you create a file, the Update and Access words become the password you specify. You can change them later with the PROT or ATTRIB commands.

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

Throughout this book, we'll refer to two distinct kinds of diskettes--distinct, that is, in terms of the information they contain, not in the way they look.

The first of these is called the System Diskette. The TRSDOS disk which you use in drive 0 is such a disk.

The System Disk contains the master program and operating system that enables your Computer to do the job you want it to do.

A Systems Diskette must always be used in drive 0.

A Data Diskette, on the other hand, contains the data or programs you have organized and saved.

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

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 gain access to the information stored on the disk just as the file password allows you to access a particular file on the disk.

If you know a disk's Master Password, it's possible to change the password (see PROT).

System File vs User File

TRSDOS makes use of two types of files--system and user files. (In general, a file is an organized collection of information.)

System files are those files used by TRSDOS and contained on the System diskette.

User files, however, are those files which you, the user, have organized and saved. These files may be on a data diskette or you may save them on a system diskette, if there is space.

TRSDOS Library Commands

APPEND
Append files

APPEND source-file destination-file

'source-file' is the specification for the file which is to be appended (added) to the second file.

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

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

Note: The record lengths must match. See DIR for more information on 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	PAYROLL/B
Atkins, W.R.	Lewis, G.E. .....
Baker, J.B.	Miller, L.O. .....
Chambers, C.P.	Peterson, B.....
Dodson, M.W.	Rodriguez, F. .....
Kickamon, T.Y.	

You can combine the two files with the commmand:

APPEND PAYROLL/B PAYROLL/A

PAYROLL/A will now look like this:

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.

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 update 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).
RENAME	Rename, read, execute, and write.
WRITE	Read, execute, and write.
READ	Read and execute.
EXEC	Execute only (any attempt to press <BREAK>, LIST, LLIST, CSAVE, etc., will erase program from memory).

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 access 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.

Note: A level of READ or above is required to load and run a BASIC 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 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.

The AUTO command-line is displayed after it is set to serve as an acknowledgement.

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.

Important Note: To Override an AUTO...

You can by-pass 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.

BUILDing New Files

When the file you specify does not exist, 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 <SHIFT> <← > for erasures and corrections.

To end the BUILD file, simply press <BREAK>.

First type:

BUILD 'filename'

You will then be prompted with the message:

TYPE IN UP TO 63 CHARACTERS
PRESS <BREAK> TO EXIT

You then type in no more than 63 characters. To exit from your BUILD command, you must then press <BREAK>. You may enter as many lines as necessary.

For example, 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 0 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 memory is cleared and reset to end of memory and the Display is cleared and all I/O drivers are reset (see Memory Requirement of TRSDOS).

This command gets you off to a fresh start.

Depending on the options you select, this command will:

- Zero user memory (loads binary zero into each memory address above 6000)
- Clears the Display
- Un-protect all memory
- Reset the stack (This is done everytime the prompt appears.)

(See Memory Requirements of TRSDOS for more information on mem-protect.)

Example

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

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

Sample Use

```
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: The real-time clock is always running, regardless of whether the clock display is on or not except during cassette and disk I/O..

Examples

CLOCK

Turns on the clock-display.

CLOCK (OFF)

Turns clock-display off.

See TIME and DATE

CLS
Clear the Screen

```
CLS
```

This command clears the Display. Use it to erase information you don't want others to see; for example, file specifications which include passwords.

Example

```
CLS
```

Sample Use

```
CREATE PERSONNEL/BAS.SECURE (LRL=255)
CLS
```

COPY
Copy a File

```
COPY source-file destination-file
          or
COPY source-file :d
          or
COPY /EXT:d :d
```

'source-file' is the name of the file which is to be copied

'destination-file' is the name of the destination file. Can be a drive-number, :d. If so, TRSDOS will default to the name extension of the original file.

':d' is the drive of the copy. If omitted, TRSDOS will search through all available drives.

'/EXT:d :d' is the name of the file-extension as well as source- /destination-drives.

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.

Examples

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

Copies OLDFILE/BAS into a new file name 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:Ø :1
```

This command copies FILE/EXT to disk 1.

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 Preallocated 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. LRL=is an option and is not necessary for file creation.
REC='bbb' is the number of Records to allow for. 'bbb' is a number between zero and 255 (or whatever the disk has space for). If omitted, no records are allocated. REC= is an option.

This command lets you create a file and pre-allocte (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.

Note: With pre-allocated files, TRSDOS will allocate extra space when you exceed the pre-allocated amount during a write operation.

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 during the 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=255)

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

CREATE NAMES/TXT.IRIS (LRL=30,REC=50)

Creates a file named NAMES/TXT protected by the password IRIS. The file will be large enough to contain 50 records, each 30 bytes long.

```
CREATE PAYROLL/BAS
```

Creates a file named PAYROLL/BAS with all of the 255-byte records available on the disk.

Sample Use

Suppose you are going to store personnel information on no more than 250 employees. Each data record would 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 won't 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 option is omitted, TRSDOS displays the current date.

If option 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

Whenever the command is entered, monitor is ON.

Q turns monitor OFF.

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

The debug monitor is designed to allow you to correct problems in 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)

DEBUG uses the memory area from 4E00 to 54FF (see TRSDOS MEMORY MAP).

(DEBUG can only by used on programs in the user area X'5500' to TOP).

Examples

DEBUG

Turns DEBUG ON.

Q

Turns DEBUG OFF and un-protects memory.

Command Description

Debug commands are usually entered by pressing a single key. Unlike conventional TRSDOS commands, 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 address (see R and J commands, for instance). Instead of pressing <ENTER> after the address is typed in, you will have to press <SPACE-BAR>.

Once you have entered the 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 =
```

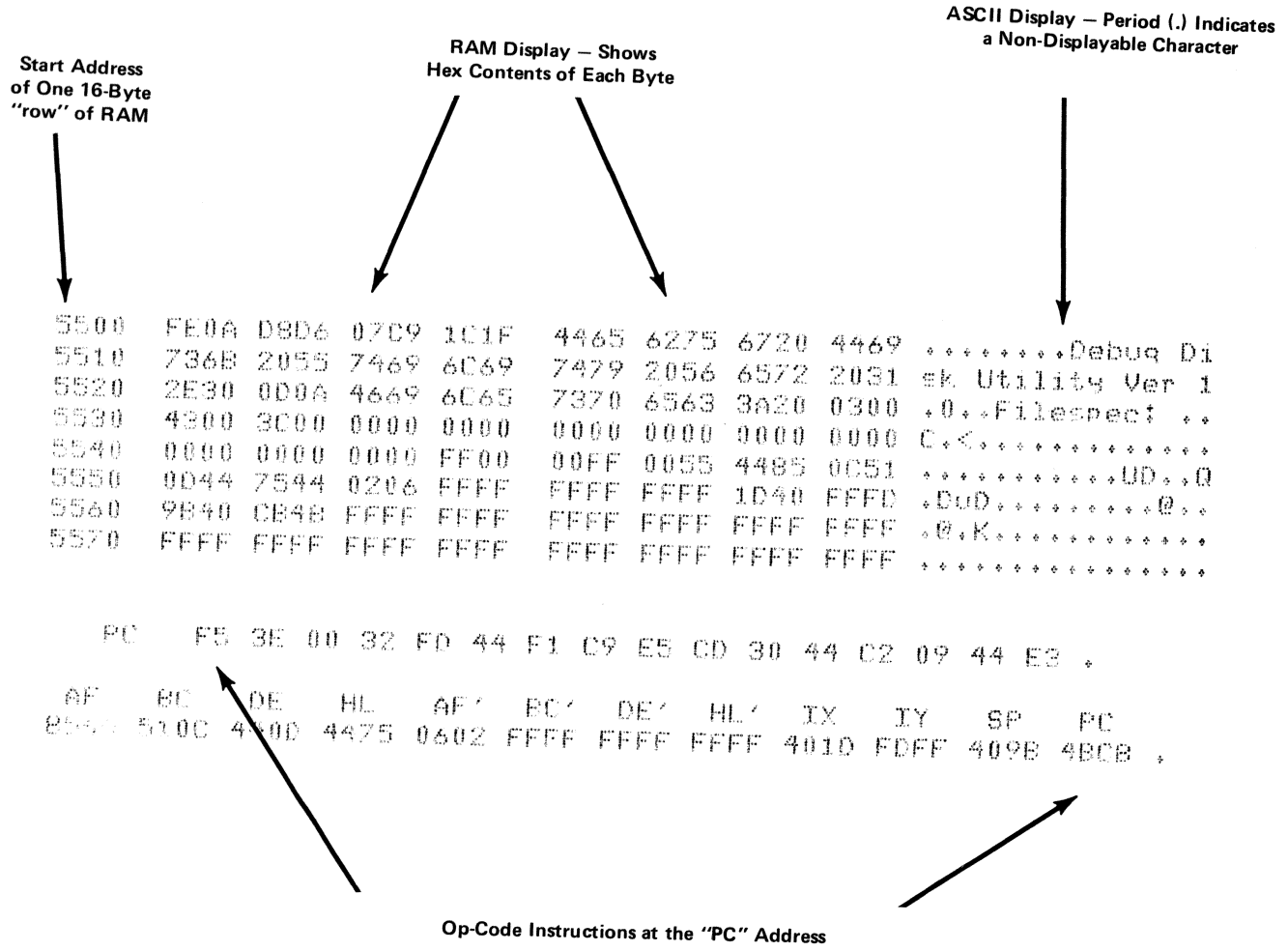
You should type in the hexadecimal address of the memory location you wish to see.

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.

The following is a typical half-screen format.



S (Full-screen Display)

By pressing <S>, the contents of a 256-byte block of memory are displayed 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 Display is on the Video.

M (Modify RAM)

Press <M> to change the display format 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.

There are two ways to exit the Modify mode. Pressing <ENTER> will allow TRSDOS to accept all changes made. Pressing <BREAK> will restore memory to its original contents.

R (Change Register Contents)

Type:

R'aa,bbbb' <SPACE-BAR>

where 'aa' is the name of the register pairs AF, BC, DE, HC, or PC.

where 'bbbb' is the four-digit hexadecimal value which will be the new register content If fewer than four digits are typed in before pressing <SPACE-BAR>, leading zeros are assumed.

I (Single-step)

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

The instruction of 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. Furthermore, breakpoints cannot be set in ROM.

C (Single-step)

If the contents at the memory location is a call instruction and you wish to complete the entire CALL/RET sequence, press <C>. The call is then executed and control is returned to DEBUG when the subroutine returns. Otherwise, this instruction acts just like the I command.

Just as with the I command, you will not be able to step through a call or jump into a ROM address; neither can you set breakpoints in ROM.

U (Update)

Pressing <U> causes the Display to be updated repeatedly.

Press any key to exit 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 Transfer of Control)

Press J to transfer control from one location to another.

Debug will respond with the prompt:

```
J ADDRESS? =
```

You should type in two sets of hexadecimal numbers in the following format:

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

where 'aaaa' is the four-digit hexadecimal number which specifies the address where the execution begins. If this number is not specified, control will be transferred to the location referenced by the program counter.

where 'bbbb' is the four-digit hexadecimal number which specifies the point where a breakpoint is desired. The contents of the specified location are set to a hexadecimal 'F7'. When execution reaches this point, control is returned to DEBUG.

When the breakpoint is set in the JUMP command, the contents of the location specified are set to a hexadecimal 'F7'. When execution reaches this location, control is returned to DEBUG and the breakpoint is removed.

Q (Quit)

Pressing <Q> will turn DEBUG off and return control to TRSDOS.

F (File Utility)

F is a special DEBUG command which enables you to load the contents of a disk file into memory and then change the contents.

When you press <F>, DEBUG will respond with the prompt:

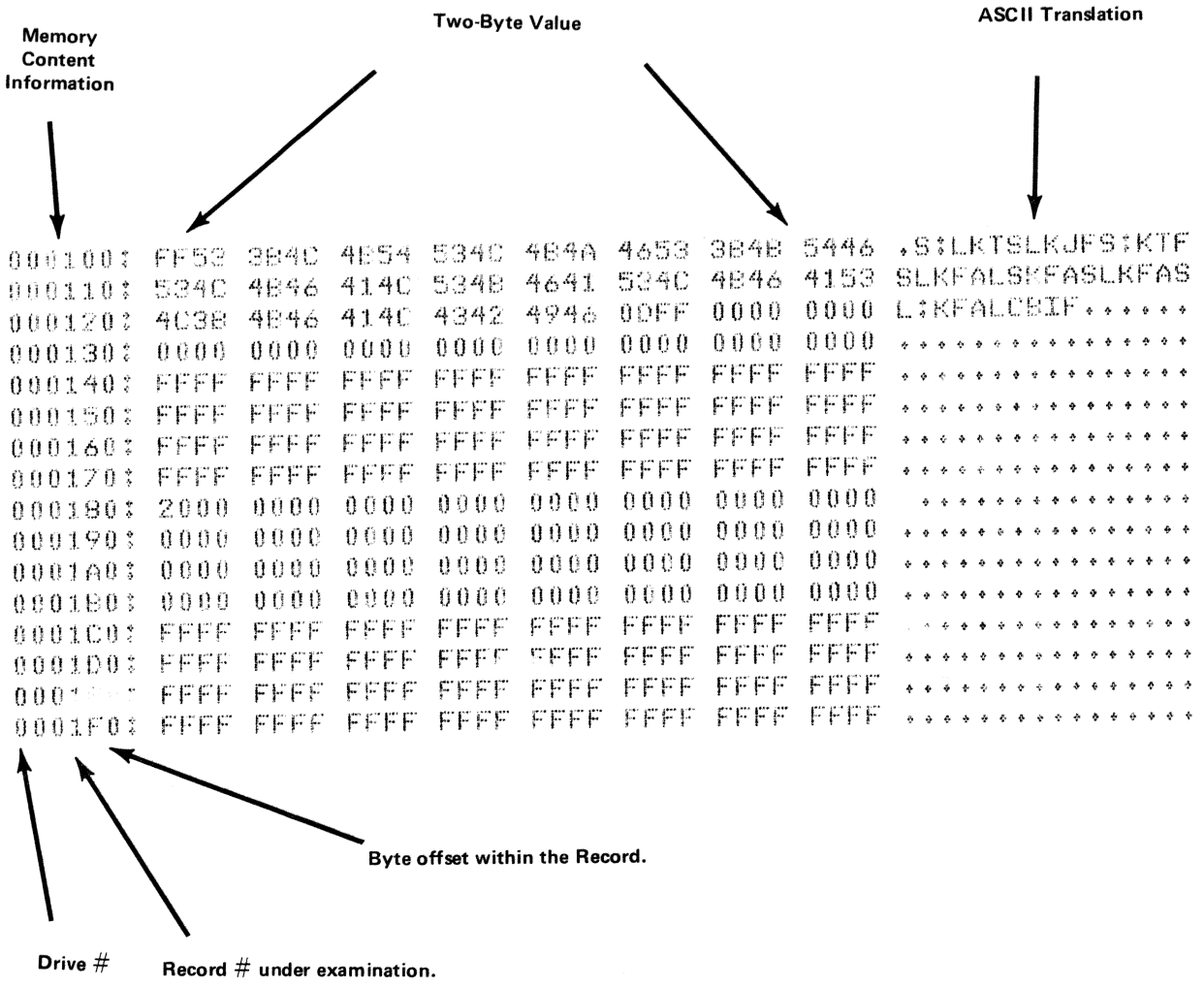
Filespec:

and you should enter the name of the file which needs to be displayed.

If the file is not found, an error message will be displayed and the prompt will reappear.

If the file is loaded, a full-screen display of the file contents will appear.

The following page gives a typical display.


Memory Content Information
Two-Byte Value
ASCII Translation
000100: FF53 3B4C 4E54 534C 4B4A 4653 3B4B 5446 .S;LKTSLKJFS;KTF
000110: 534C 4B46 414C 534B 4641 534C 4B46 4153 SLKFALSKFASLKFAS
000120: 4C3B 4B46 414C 4342 4946 00FF 0000 0000 L;KFALCBIF.......
000130: 0000 0000 0000 0000 0000 0000 0000 0000 ....
000140: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
000150: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
000160: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
000170: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
000180: 2000 0000 0000 0000 0000 0000 0000 0000 ....
000190: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0001A0: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0001B0: 0000 0000 0000 0000 0000 0000 0000 0000 ....
0001C0: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
0001D0: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
000[illegible]: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
0001F0: FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF ....
Byte offset within the Record.
Drive #
Record # under examination.

In this display mode, control values are not translated into periods, so all character values will appear in the right-hand column.

The display control commands are the same as in the normal display mode (i.e., <;> for forward, <-> for backward, etc.). To modify the contents of a file, press <M>.

The cursor will appear as a blinking character at the location of the first byte. The arrow keys (<↑>, <↓>, <→>, and <←>), will control the cursor's position.

To complete the operation, press <ENTER>. To abort the modification, press (BREAK>.

To load another file, press <BREAK> and the prompt will return. To exit, press <BREAK> when the prompt is displayed.

Once the utility has been started, control will not return to DEBUG. When you exit from the prompt, control returns to TRSDOS.

DIR
List the Diskette Directory

DIR :drive-specification (INV,SYS,PRT)

':drive-specification' 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

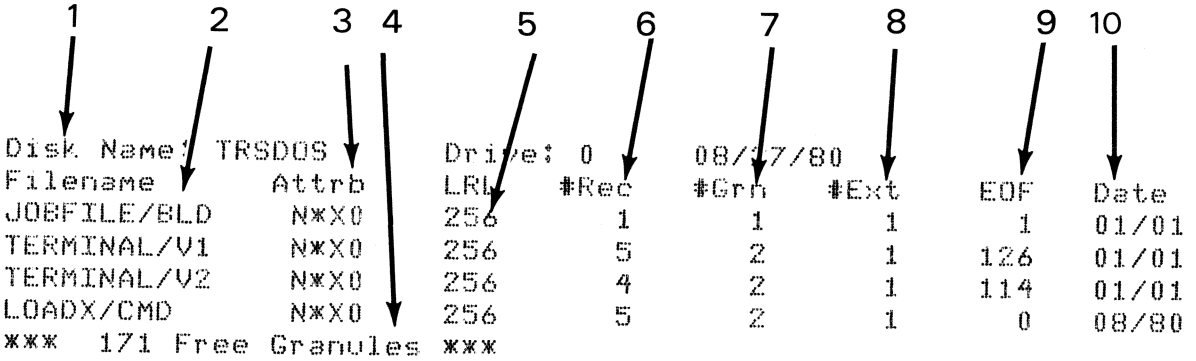
DIR

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



Definition of column headings

1. Disk Name--The name assigned to the disk when it was formatted.

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

3. Attributes--A four-character field.
   The first character is either I (Invisible file) or N (Non-invisible file).

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

   The third character gives the password protection status.

   - 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 Kill file and everything listed below.
   - 2 Rename file and everything listed below.
   - 3 Not used.
   - 4 Write and everything listed below.

5 Read and everything listed below.
6 Execute only.
7 No access.

4. Number of Free Granules--How many free granules remain on the diskette.

5. Logical Record Length--Assigned when the file was created.

6. Number of Records--How many logical records have been written. Asterisks (*) signify none have been written.

7. Number of Granules--How many granules have been used in that particular file.

8. Number of Extents--How many segments (contiguous blocks of up to 32 granules) of disk space are allocated to the file. Asterisks (*) signify none.

9. End of File (EOF)--Shows the last byte number of the file

10. Creation Date--When the file was created.

DO
Begin Auto Command Input from Disk File

DO command-line
'command-line' is the name of file created with BUILD (or its equivalent).

If no extension is included, /BLD is used.

The extension must be omitted in most /BLDs.

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 command cannot be included in an automatic command input file.

Running User Programs from a DO-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 Ouput to Video and Printer

```
DUAL (switch)
   'switch'       is one of two options, ON or OFF.
If option is omitted, TRSDOS uses ON.
```

This command enables all video output to be copied to the printer after <ENTER> has been pressed (and vice versa).

The DUAL (ON) command is displayed after being set to serve as an acknowledgement.

Video and printer output may be different because of intrinsic differences between output devices and output software.

Using the DUAL command will slow down the video output process.

The DUAL command cannot be used during ROUTE and vice versa.

Sample Use

For a printed copy of all system/operator dialog, type:

```
DUAL
```

To turn off the DUAL process, type:

```
DUAL (OFF)
```

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 6000H.

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, TRSDOS uses the START address instead.

Note: Addresses must be hexadecimal form, without the X' ' notation.

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=6000,END=6F00,TRA=8010,RELO=8000)
```

Creates a program file named PROG2/CMD containing the program in addresses X'6000' to X'6F00. When loaded, PROG2/CMD will reside from X'8000' to X'8F00'. Execution will start at X'8010'. The program is executable from TRSDOS READY.

ERROR
Display Error Message

ERROR number
'number' is a decimal number for a TRSDOS error code.

This command displays a descriptive error message.

* * ERROR 47 * *

You respond by typing:

ERROR 47

and the Video will display the full error message.

Example

ERROR 3

Gives you the message:

Lost Data During Disk I/O

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

FORMS
Set Printer Parameters

FORMS (WIDTH=aaa,LINES=bbb)

WIDTH='aaa' is the maximum number of characters per line. 'aaa' can be any number between one and 255. If omitted, whatever value currently in effect will be used. To turn the function off, 255 is used.

LINES='bbb' is the maximum number of lines to print before an automatic form feed. 'bbb' can be any number between one and the maximum number of lines of your page size. If omitted, 60 is used.

This command lets you set up the TRSDOS Printer software to suit the Printer you have attached. If the Printer was ready when you started TRSDOS, and the default parameters WIDTH=132 LINES=60, are appropriate, then you do not need to use this command.

If no specification is indicated, FORMS will skip to top of form.

Examples

FORMS

Resets all parameters to their default values (and skips to top of form).

FORMS (LINES=56)

Resets the maximum number of printed lines per page to 56, leaving 10 lines blank on each page.

FORMS (WIDTH=80)

Sets up the serial printer driver with 80-character lines and

all other parameters according to their default values.

Setting the Parameters

Lines per page. This number determines the number of blank lines at the bottom of each page. If you set lines equal to page size, then TRSDOS will print every line on the page. If you set lines equal to page size minus 6, then TRSDOS will leave 6 blank lines on each page. Lines per page cannot exceed page size.

Width. This number sets the maximum number of characters per line. If a print line exceeds this width, TRSDOS will automatically break the line at the maximum length and continue it at the beginning of the next Print line.

FREE
Display Disk Allocation Map

FREE :d (PRT)

':d' is the drive specification

(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.) This information is useful when you want to optimize file access time.

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
D	Directory Information
X	Allocated Granule
F	Granule Contains a Flawed Sector (Unusable)

The following is a typical free map display.

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

HELP
Explanation of TRSDOS Command

HELP command
'command' is the specific TRSDOS command which you need help understanding or using. It is an option.

If omitted, TRSDOS will display a list of all usable commands.

If a command is used which isn't on the list, TRSDOS will default the list of usable commands.

This command gives you "help" in using TRSDOS commands by displaying a specific definition as well as syntax format upon request.

Without a specific command, TRSDOS will list all available commands that are defined by and used with the HELP command.

Example

If you type in the following:

HELP BACKUP

TRSDOS will respond with the syntax format, a definition of the command, and an explanation of the abbreviation.

BACKUP [:d] [:d]
Duplicate a diskette. :d = Drive Number

KILL
Delete a File

KILL file/EXT:d
'file' is the file-specification. If file-specification is omitted, all unprotected files with the same extension will be KILLed.

This command deletes a file from the directory and frees the space allocated to that file. If no drive is specified, TRSDOS will search for the file, starting with drive 0.

Examples

KILL TESTPROG/BAS

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

KILL JOBFILE/IDY.FOGGY

Deletes the named file from the first drive that contains it. The file is protected with the password FOGGY.

KILL FORM/A:3

Deletes FORM/A from drive 3.

Sample Uses

When updating a file, it is a good practice to input from the old file and output updated information to a new file. That way, if the update is wrong, you still have the old file as a backup. When you have verified that the update file is correct, you can KILL the old file.

LIB
Display Library Commands

LIB

This command lists to the Display all the Library Commands.

Example

LIB

The following is an example of a LIB display.

```
DOS Ready
LIB
APPEND   ATTRIB   AUTO     BACKUP   BUILD    CLEAR    CLOCK    CLS
COPY     CREATE   DATE     DEBUG    DIR      DO       DUAL     DUMP
ERROR    FORMS    FORMAT   FREE     HELP     KILL     LIB      LIST
LOAD     MASTER   PATCH    PAUSE    PROT     PURGE    RELO     RENAME
ROUTE    SETCOM   TAPE     TIME     WP

DOS Ready
```

LIST
List Contents of a File

LIST file (PRT,SLOW,ASCII)
'file' is the file specification

PRT tells TRSDOS to list to the Printer. This is optional. If omitted, only the Video Display is used.

SLOW tells TRSDOS to pause briefly after each record. This is optional. If omitted, the listing is continuous.

ASCII tells TRSDOS to list the file in the ASCII format. This is optional.

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.

Only ASCII codes 0-7F' are sent to the Printer. All ASCII print characters have a bit-7 cutoff before being printed. Furthermore, all ASCII hexadecimal numbers of 80 and above will be forced to below 7F'.

TABs in the text files will be displayed.

Examples

LIST DATA/TXT

Lists the contents of DATA/TXT

LIST TEXTFILE/A (SLOW)

Lists the contents of TESTFILE/1, pausing after each record.

LIST PROGRAM/CMD (PRT)

Lists the file PROGRAM/CMD to the Printer.

LIST PROGRAM/CMD (ASCII)

Lists the ASCII code for the file PROGRAM/CMD.

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 DOS 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.

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, Master function is turned off for all drive.

This command allows you to assign a specified drive as the Master Read or Write drive in the system.

If a drive is specified as the Master drive, TRSDOS will begin searching at that drive, by-passing all previous drives.

If the file is not found on the specified drive, TRSDOS will continue searching on the next drive. If the file is not found on that drive, TRSDOS will indicate an error message for the file not found.

When no drive is specified, any drive defined as Master will be released as the Master drive.

Example

If you enter the command

MASTER (DRIVE=2)

drive 2 will become the Master drive and file searching will begin at that drive.

PATCH
Change the contents of a disk file

PATCH file (ADD=aaaa,FIND=bb,CHG=cc)
This is the form to use when you are patching a program. Files created with DUMP will fall into this category.

'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 two-digit hexadecimal number.

CHG='cc' specifies the string you wish to change 'cc' to.

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. You can also use it to make minor replacement changes in data files.

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.

Using PATCH on a Program File

Suppose you want to change seven bytes in a machine-language program file. First determine where the 7-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: VDREAD
Start address: X'5280'
Sequence of code to be changed: X'CD2D25E5'
Replacement code: X'0000009'

Then you could use the following command:

```
PATCH VDREAD (ADD=5280,FIND=CD2D25E5,CHG=X'0000009')
```

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
PRESS <ENTER> TO CONTINUE
```

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 the Master Password

```
PROT :d (PW,LOCK)
  ':d'    is the drive specification
  PW      is the prompt for password change
  LOCK    tells TRSDOS to assign the Master Password
          to the unprotected user file.

          If LOCK is omitted, user file protection is
          left unchanged.
```

PROT changes file protection on a large scale. If you know the diskette's Master Password, you can change it.

A diskette's Master Password is initially assigned during FORMAT or BACKUP. The TRSDOS diskette is supplied with the Master Password, PASSWORD.

Example

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

TRSDOS will prompt you with the message:

```
New Master Password?
```

which you can then enter.

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	System files only.
INV	Invisible files only.
ALL	All files on disk.

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.)

All System files are required for TRSDOS to function. Do not eliminate System files if you want to use the diskette in drive 0.

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.

RELO
Change Where Program Loads into Memory

RELO file (ADD=aaaa)

'file' is the file-specification

ADD='aaaa' is the memory relocation address where the program loads into memory. 'aaaa' is a four-digit hexadecimal number referring to an address in the user memory.

This command allows you to change the address at which the program loads into memory. RELO will change the loading address, not the program itself.

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 file file

'file' is the file specification.

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

If the new file name includes a drive specification or password, it will be ignored. The file will retain its former password, if any.

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.

Example

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= is the source I/O device. This is optional.

DESTIN= is the destination I/O device. This is optional.

'aa' and 'bb' may be any two of the following two-letter abbreviations:

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

If options are omitted, TRSDOS resets I/O Drivers to their original I/O route.

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

If no source or destination is specified, TRSDOS resets the routing to its original state.

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

Example

ROUTE (SOURCE=KB DESTIN=PR)

TRSDOS will route your keyboard input directly to the Printer.

ROUTE

I/O drivers are returned to their original state.

SETCOM
Set Up RS-232C Communcations

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

OFF turns RS-232C 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, WORD is set to its present value.

BAUD='bbbb' is the desired baud. 'bbbb' must be a decimal number between 50 and 9600. If omitted, BAUD is unaffected.

STOP='c' is the desired number of bits. 'c' must be either 1 or 2. If omitted, STOP is unaffected.

PARITY='d' determines whether the parity is odd, even, or none. 'd' must be 0 (none), 1 (odd), or 2 (even). If omitted, PARITY is set to its present value.

'mode' type either WAIT or NO WAIT

Options must be entered in the order shown.

If option is omitted, you must instead press <ENTER> to register the present value.

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

To change the setting on a currently active channel, you must first turn the channel off. If the channel is already off when you try to turn it off, you'll get an error message.

See the Model III Operation Manual for a description of RS-232C signals used. For hard-wired connection from one Model III to another, see the wiring diagram in Technical Information, RS-232C supervisor call.

These system routines are only available when the channel has been initialized, see Technical Information for details.

Examples

SETCOM (WORD=7,BAUD=300,STOP=1,PARITY=0,WAIT)

This would set the RS-232C to 7 bits, 300 baud, 1 stop bit, no parity, and place it in the WAIT mode.

SETCOM

The command without specifications will default to 300 baud, seven bits, even parity, and one stop bit.

The following program will allow you to use your Computer as a terminal. For further information, refer to the Operation section, page 8/2, of your Model III Operation Manual.

Note: This program executes at 300 Baud.

```
5 DEFINT A-Z              'INTEGER VARIABLE FOR SPEED
10 POKE 16896, 0          'DON'T WAIT FOR SERIAL I/O
15 POKE 16888, (5*16)+5   'TX/RCV AT BAUD RATE 300
20 DEFUSR0 = &H005A: REM SET UP CALL TO $RSINIT
40 X = USR0(0)
60 DEFUSR1 = &H0050
65 DEFUSR2 = &H0050
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
Execute Tape Transfer Operation

TAPE (S=a,D=b)
S= is the source device for beginning the operation.

D= is the destination device to complete the operation.

'a' and 'b' may be any of the following abbreviations:

T	(Tape)
D	(Disk)
R	(RAM/Memory)

TAPE allows you to transfer program files from one memory device to another.

The following combinations may be used.

Disk TO Tape
Tape TO Disk
Tape TO RAM

If you transfer information from Tape to Disk, TRSDOS will display the tape filename and prompt you with the following:

Cass?

You should press <H> (High speed), <L> (Low speed), or <ENTER> (High speed will be used), whichever is appropriate.

Press any key to continue.

If you are transfering disk information to tape, TRSDOS will ask you the file-specification before you can proceed.

Example

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

After the prompt, TRSDOS will transfer a program file from a diskette to a cassette tape.

TIME
Reset or Get the Time

TIME hh:mm:ss

'hh:mm:ss' is the specification for 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, TRSDOS displays the current time.

This command lets you reset or display the time.

It should be remembered that the built-in clock is a 24-hour clock. Therefore, any times displayed or reset after 12:00 noon will be greater than 12 and less than 24. (For example, 1:00 pm is displayed as 13:00 o'clock.)

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 pm.

Examples

TIME

Displays the current time.

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

Resets the time to 1:20:00 pm (13:20:00 will be displayed).

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 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 over-written by this command. It is a software write-protect rather than a hardware write-protect (such as a write-protect tab on the diskette).

Only one drive may be protected at a time.

To un-protect on a diskette, making it accessible to writing, simply enter the command WP and press <ENTER> and all drives will be un-protected. This will not, however, 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-drive :destination-drive

':source-drive' Tells TRSDOS the number of the drive containing the original disk

':destination-drive' Tells TRSDOS the number of the drive containing the copy disk

if source/destination numbers are omitted, TRSDOS requests the specific drive-numbers

Note: Spaces are necessary before colons.

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.

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 Program Conversion

CONVERT

Model I formatted diskettes containing program and data files, cannot be used with your Model III Computer unless the information is first "transported" to a Model III diskette.

This is necessary because of Model III's many special features, such as a difference in the number of sectors as well as the density of the disk.

CONVERT allows you to transport the information by reading a Model I formatted diskette, reformating the information, and then copying it unchanged onto a Model III formatted diskette. (Note: If you try to copy onto an unformatted diskette, TRSDOS will prompt you to FORMAT the diskette first. The destination diskette must be a Model III formatted diskette.)

The Model I diskette may still be used on a TRS-80 Model I since only the transported information was reformatted when it was copied, not the Model I diskette itself.

(Note: The process converts Model I formatted information to Model III formatted information only. Model III formatted information cannot be converted for Model I use.)

Only user files may be converted. CONVERT will disregard any Model I system file it encounters on a diskette.

Before you begin the conversion process, always be sure there is a Model III TRSDOS system disk in drive 0 of your TRS-80 Model III.

Once you enter the command, TRSDOS will prompt you for the source- and destination-drives, list the Model I diskette filenames, and then execute the conversion.

Information is then reformatted and copied from one drive to another (from drive 1 to drive 0 or from drive 2 to drive 1, for instance) as long as the Model III system disk remains in drive 0 and the source drive number is greater than the destination drive number.

Once the original files have been reformatted and the information copied, the diskette containing the reformatted information may be used just like any other Model III diskette.

For example, if you wish to COPY the converted programs or data files onto another Model III diskette, you may do so in the normal manner. If you don't need the information and wish to free the space, you may then KILL the files.

Example (for a typical two-drive system)

If you enter the command:

CONVERT

The following prompt will appear:

Source Drive?

You should type the drive-number of the Model I diskette --1.

You will then be prompted with:

Destination Drive?

You should type the drive-number of the Model III diskette--*0*.

If CONVERT encounters a Model I filename which is the same as a filename on the Model III diskette, the message:

File Exists. Use It?

will appear. If you want to copy over the file on the Model III diskette, simply type:

Y

for YES. Otherwise, you can type:

N

for NO.

Next, the filenames of the Model I diskette will be displayed as they are converted for Model III use.

CONVERT will end the operation with the message:

Conversion Complete

The information on the Model I diskette in drive 1 would have been copied and reformatted to a Model III format on the diskette in drive 0, enabling you to use the Model I programs and data files now on the diskette in drive 0 with your Model III system. You could, however, still use the Model I diskette with a Model I TRS-80 since the Model I diskette itself has not been changed.

The only change which occured was the re-formatting of the information as it was copied to the Model III diskette.

Note: If you are using either a three- or four-drive system (multi-drive), you may use, for example, drive 1 as the destination drive and drive 2 as the source drive. Drive 0 is reserved for the Model III system diskette.

Sample Use

You wish to convert a Model I diskette which contains a file called JOBFILE/BLD. (Multi-drive system.)

DOS Ready

```
.................................
```

Type:

```
CONVERT <ENTER>
```

The following message will be displayed:

```
Model 1 To Model 3 Conversion Utility . Ver v.r
```

('v.r.' is a pair of numbers specifying the version and release you have.)

```
Source Drive?
```

Type:

```
2 <ENTER>
```

```
Destination Drive?
```

Type:

```
1 <ENTER>
```

The filenames on the diskette will be displayed:

```
JOBFILE/BLD
```

The following message will appear when the command execution is complete:

```
Conversion Complete

DOS Ready
.........................
```

Note: Files with access password must be blank (see ATTRIB (ACC=,)) before conversion can take place. If access password is not blank, the file cannnot be converted and will be skipped over (no message will indicate this) and CONVERT will pass on to the next file.

If you have a file with an update password, TRSDOS will prompt you for the password. If the correct password is given, the conversion will take place.

If an incorrect password is givenm you will be re-prompted,

If the update password is unknown, you can press ENTER . The original password will then be copied along with the file and the file will then be assigned an unknown password.

In other words, if you're unable to update a file on Model I because you don't know the password, you will not be able to update it on Model III; you can, however, still copy and convert it.

Note for Machine-language programmers: Machine-language programs may be converted as long as the address calls are the same. If the address calls are not the same, the conversion will still occur but the program may not execute correctly. For changes in these calls, see the Technical Information section of this manual and the Model III Operation Manual.

FORMAT
Prepare a Data Diskette

FORMAT :d

':d' is the disk-drive which contains the diskette being formatted

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

Note: Data diskettes can 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 directory and bootstrap files.

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 re-format, 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, re-format the disk. If there are defective tracks, FORMAT will lock them out, and you'll be left with an otherwise usable diskette.

TRSDOS Technical Information

Contents of This Section

Disk Organization

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

The minimum system overhead amounts to one full track of directory information and a half track of TRSDOS bootstrap program and other information. This means that every TRSDOS diskette is self-loading, although it may or may not actually contain the TRSDOS system. This is done to prevent the Computer from attempting to bootstrap a diskette containing only user data files.

The utility command library is optionally available on the diskette. Since the utility command programs are not always required, it will often be advantageous for multi-drive users to format diskettes for use in drives 1 through 3. Such "data diskettes" contain a minimum of system code, leaving more space for user

files. Maximum file size is limited only by the physical size of the diskette, since a file must be wholly contained on one diskette.

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 only be initiated 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 de-blocking 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

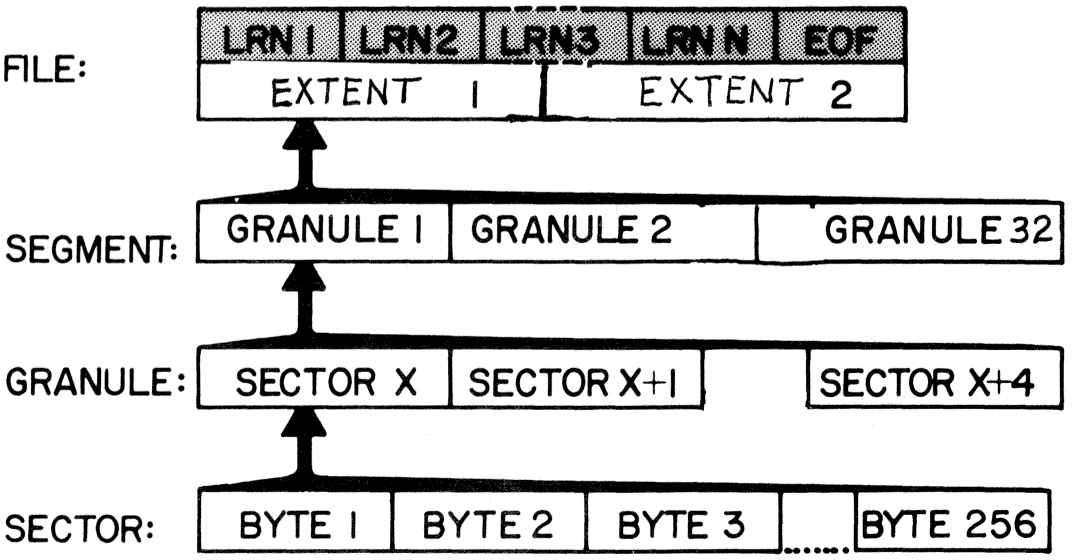
A TRSDOS file is composed of one or more segments of storage space. Each segment consists of from one to 32 physically contiguous granules of storage. A granule is the minimum allocatable unit of storage, and consists of **3** sectors (**768** bytes). (See Figure below).

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.

The effect is to decrease the amount of "thrashing" present in a file which has had frequent additions made. (A wholly sector-mapped system could not offer this benefit.)

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

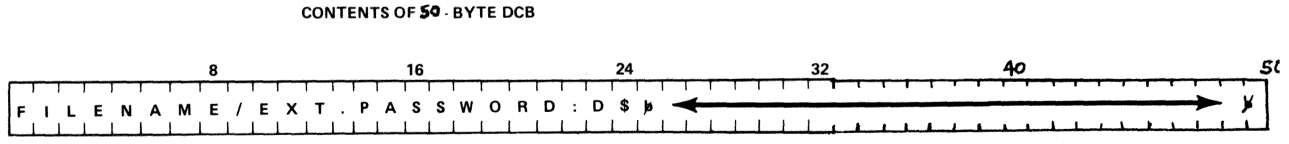
This information is provided for customers who wish to write their own assembly level I/O routines. An explanation of the calling sequence and parameters for each necessary I/O routine is given. A knowledge of Z-80 machine code is assumed.

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)
DE= > *xxxx*	Registers DE contain the address of (point to) *xxxx* in machine format. (If address of *xxxx*=5AF1H then the values in the registers are: D=5A ; E=F1)
B= *xx*	Register B contains the numeric value of *xx* in binary form. If *xx*=64 decimal, then B=40H.
A= *xx*	Register A contains the numeric value of *xx* in binary form. If *xx*=127 decimal, then A=7FH. 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.
Z=OK	Zero flag is set (OK) if successful return from the system routines.
X'*nnnn*'	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.

DCB before OPEN and after CLOSE:

The DCB 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:



Notes: /EXT, .PASSWORD, :D are optional.
$ stands for a carriage return (X'0D')
b stands for a blank (X'20')

Explanation of DCB while OPEN:

lsb/msb is least significant byte followed by most significant byte in Z80 RAM format (i.e. addr=7CC8 in RAM is C8 7C).

Addr.	Len.	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 no. – open sets=X'0000' – lsb/msb)
+12	2	ERN (ending record no. – 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 5-1/4" 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.

Fundamental TRSDOS I/O Calls

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

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

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 properly apply this data, 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 (jump vector = 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: HL= >BUFFER (see beginning of this section for notation)
DE= >DCB
B= LRL
CALL 4420H

exit: 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 (jump vector = 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: HL=> BUFFER
DE=> DCB
B= LRL
CALL 4424H
exit: Z=OK
Z=0 if file does not exist.
A=TRSDOS error code.

POSN (jump vector = 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 (see page 2).

entry: DE=> DCB (must have been opened previously)
BC= Logical record number to position for.
CALL 4442H
exit: Z=OK
A=TRSDOS error code.

READ (jump vector = X'4436')

If LRL>0, READ transfers the logical record whose number is in the DCB as NRN (see page 2) into the RAM area addressed as UREC for the length LRL as defined at open time. The record comes from the RAM 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 one in the DCB after the transfer is completed. INIT/OPEN sets NRN=X'0000' in order to read the first record with the first READ.

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

entry: HL= > UREC if LRL is not zero. Unused if LRL=0.
DE= > DCB
CALL 44364
exit: Z=OK
A=TRSDOS error code. (EOF=X'1C' or X'1D')
(see errors 28,29 for EOF or NRF)

WRITE (jump vector = X'4439')

IF LRL>0, 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 RAM 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 one.

IF LRL=0, WRITE transfers one physical record from the RAM 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: HL= > UREC if LRL is not zero. Unused if LRL= 0
DE= > DCB
CALL 4439H
exit: Z=OK
A=TRSDOS error code.

VERF (jump vector = 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: HL= > Same as WRITE above.
DE= > DCB
CALL 443CH
exit: Z=OK
A=TRSDOS error code.

PUTEXT ------ 17492/X'4454'

This routine will add an extension to a filename if an extension does not already exist. An extension to a filename can identify the type of data file being stored.

Entry Conditions

(DE) = Disk DCB before the OPEN
(HL) = The extension to be added to the file

Exit Conditions

None

BACKSPACE ----- 17480/X'4448'

This routine positions the file record pointer to the previous record. This is useful in record sequence-checking.

Entry Conditions

(DE) = DCB

Exit Conditions

Z = Invalid position in file
NZ = Invalid position in file (REC-1)-1

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 (especially payroll for checks and check registers).

Entry Conditions

(DE) = DCB

Exit Conditions

Z = Good file specifications
NZ = Bad file specifications

POSEOF ----- 17477/X'4445'

Point to end-of-file. This routine positions the file pointer to the last record in the file. This may be used to verify that the last record was written to the file.

Entry Conditions

(DE) = DCB

Exit Conditions

Z = Good file specifications
NZ = Bad file specifications

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

Exit Conditions

Z = Good file specification
NZ = Bad file specification

DIVIDE ----- 19295/X'4B5F'

The divide routine provides a 16-bit dividend and an 8-bit divisor. After division, the quotient replaces the 16-bit dividend and the remainder the 8-bit divisor.

Entry Conditions

(HL) = **Dividend**
(A) = **Divisor**

Exit Conditions

(HL) = Quotient
(A) = Remainder

DMULT ----- 19269/X'4B$%'

The multiply routine provides a 16-bit multiplicant and an 8-bit multiplier. After multiplication takes place the product replaces the 16-bit multiplicant.

Entry Conditions

(HL) = Multiplicant
(A) = Multiplier

Exit Conditions

(HL) = Product
A = Overflow, if any

RAMDIR ------- 16975/X'424F'

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

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).

HL = Buffer area where record is to be sent (user determined).

Exit Conditions

NZ = Error occurred.

Z = No error. HL = directory or free-space information.

Directory Format

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

Byte Number	Contents
0-14	Filename/EXT:d (left-justified followed by spaces)
15	Protection Level --Binary 0-6
16	Byte EOF (0-255)
17	Logical Records Length -- Binary 0-255
18-19	Last sector number in file -- Binary
20-21	Number of Grans file allocated (LSB,MSB)-Binary
22	"++" (Marks the End of Directory after entire directory.)

FILPTR -------- 16972/X'424C'

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

A = 424C

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

Exit Condtions

NZ = Error occurred.

Z = No error. The following registers are set up:

B = Which drive contains the file (binary 0,1,2, or 3).

C = Logical file number (1-96)

Note: This operates with User files only.

CLOSE (jump vector = 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" is complete for file management.

entry: DE=> DCB
CALL 4428H
exit: Z=OK
A=TRSDOS error code.

KILL (jump vector = X'442C')

KILL deletes the directory entry for an open file and then completes the close on the DCB. The disk space released by the old file is now re-useable for other purposes. Otherwise KILL is the same as CLOSE.

entry: DE=> DCB
CALL 442CH
exit: Z=OK
A=TRSDOS error code.

Supplementary Information

Other routines and addresses which may be of interest are defined here. Pay particular attention to the error routine. It does NOT perform error recovery. It displays TRSDOS error messages on the video display.

(1) CALL 402DH – Normal return to TRSDOS at program end.

(2) **X'4470'** address of the 64-byte buffer that contains the last TRSDOS command that was entered. Useful to decode special parameters entered when program was executed (run).

(3) If HL => 8-byte buffer, then:
CALL **35ADH** returns the time of day into the 8 bytes in the ASCII format – HH:MM:SS
CALL **35BBH** returns the date into the 8 bytes in the ASCII format – MM/DD/YY

Binary forms of the time and date are located in TRSDOS RAM at these locations:

X'4217' time – binary – 3 bytes – sec,min,hrs
X'421A' date – binary – 3 bytes – yr, day, mon

(4) Printing TRSDOS error codes on the video display.

```
CALL  4420H    Example of system I/O call. Any call
               is ok. Zero flag not set means an error
               has occurred during the I/O attempt.
JR    Z,OKGO   Ignore error message display if no
               error.
OR    80H      Optional for detailed error message.
               Register A already contains proper
               code for a single line message display.*
CALL  4409H    Display error message on video screen.

          Optional user error recovery code goes here

OKGO continue with program here - - -
```

*Note: If Bit-6 is ON, a detailed message is given. If Bit-6 is OFF, only the error number is displayed. If Bit-7 is ON, control is returned to the CALLing instruction. If Bit-7 is OFF, the return is to DOS Ready.

```
OR          40H          test 7th Bit
OR          COH          test Bits 6 and 7
```

Omitting the OR will cause the error to be displayed and a return to DOS Ready.

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 file
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 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

Part III Disk BASIC

DISK BASIC

LANGUAGES

Contents of This Section

Introduction

DISK BASIC is a set of enhancements to **Model III** , plus features to allow disk input/output of BASIC programs and data. It is a memory image file stored on the TRSDOS software diskette with the name BASIC and extension /CMD.

When DISK BASIC is loaded into RAM, it automatically takes control of the **Model III** ROM program, using almost all of its routines and adding others.

BASIC occupies memory beginning at hex address 5200 (decimal 20992).

To load and execute DISK BASIC, first power-up the Disk Operating System (see System Operation), so that

```
DOS Ready
```

is displayed. Now type:

```
BASIC ENTER
```

TRSDOS will load BASIC into RAM, and BASIC will begin the "initialization dialog". This is a series of questions and answers which tell BASIC how to organize memory according to your needs.

The first question is,

```
HOW MANY FILES?
```

You should respond with the maximum number of files that will be open (in use) at any one time (any number from 1-15) and whether or not the file is to be variable in length (see OPEN for more details). For example:

```
3V
```

(Every program or data set you store on the disk is referred to as a "file". In fact, everything on the disk, including system software, exists in the form of files.)
The number you enter tells BASIC how many disk I/O buffers and data control blocks to create and the V reserves adequate buffer space. If 'n' files are to be used, then 'n' buffers will be required.

If you simply press **ENTER** without entering a number, BASIC will use a default value of 3; so you'll be able to have 3 file buffers in use at once.

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.

Suppose you're going to be using 2 files: 1 for inputting data, 1 for outputting data. Then you might answer 2 to the FILES? question. However, if only 1 of these files will be open at once, then you really only need to reserve 1 file buffer/control block.

Examples:

```
HOW MANY FILES? 0 ENTER
```

causes BASIC to set aside zero buffers for I/O to disk files. You won't be able to open files, but you will have the maximum amount of RAM for use by your program.

```
HOW MANY FILES? 15 ENTER
```

tells BASIC to create 15 I/O buffers and control blocks; you will then be able to have 15 files open at once; however, this will reduce your available memory by 15*290 = 4350 bytes.

```
HOW MANY FILES? ENTER
```

tells BASIC to use a default of 3 for the number of files to be in use at once.

After you answer the FILES question, BASIC will ask:

```
MEMORY SIZE?
```

Simply press ENTER without typing a number.

```
MEMORY SIZE? ENTER
```

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.

You would then 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.

Here's how you might use your protected memory:
You can load machine-language programs or routines into high memory, and then access these routines from DISK BASIC via specially defined USR*n* functions, or via the SYSTEM command. These machine language routines may be loaded from tape using the SYSTEM command, LOADed in the DOS READY mode, or placed in memory one byte at a time using BASIC POKE commands. If you do not reserve memory, such routines will be destroyed during execution of BASIC statements.

Refer to the Memory Map for decimal addresses of the various TRS-80 memory configurations (16K, 32K, 48K).

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 files the diskette contains

To exit Disk BASIC and return to the DOS Ready mode, type:

```
CMD"S" ENTER
```

This results in a normal return to DOS – without re-initialization of the system. If you have a BASIC program in RAM, it will be lost, so be sure to save it on disk or tape before using CMD"S"

You can return to BASIC with program intact if you haven't changed user memory while in TRSDOS. Use BASIC *.

Note: The following technical information explains how to protect BASIC memory from machine language programs loaded through TRSDOS.

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

&H and &O (hex and octal constants)

Often it is convenient to use hex (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*	
where *dddd*	is a 1 to 4 digit sequence composed of hexadecimal numerals 0,1,...9,A,B,...,F.
&O*ddddd*	
where *ddddd*	is a sequence of octal numerals 0,1,...,7. and &O*ddddd*< =177777 decimal. **Note:** The O can be omitted from the prefix &O. Therefore &O*ddddd*=&*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.

```
100 FOR I=(&H3C00) TO (&H3FFF) STEP (&H40)
200 IF A=(&H37E8) THEN A=A+1
300 POKE A%, (X% AND &HFF)
```

Masks the most significant byte of X% and POKEs the result into location A%.

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
Lxx	List Program Line 'xx'
Exx	Edit Program Line 'xx'
Dxx	Delete Program Line 'xx'
Axxx,xxxx	AUTO Beginning at Line 'xxx', Incrementing by 'xxxx'.

BASIC *
Return to BASIC (Keeping BASIC Program Intact)

BASIC *

BASIC * is a **TRSDOS command** which enables you to leave BASIC, enter TRSDOS and return to BASIC without losing the resident program. Variables, however, will not be retained.

If you have typed in a BASIC program and exited Disk BASIC in the normal manner (see CMD"A" or CMD"S"), but wish to re-enter BASIC without losing the program, you must type in BASIC *.

After entering BASIC *, you immediately enter Disk BASIC, by-passing the usual "File" and "Memory Size" prompts.

Example

The Display contains the prompt:

```
DOS Ready
.....................
```

Enter Disk BASIC and enter a program. When you are ready to exit BASIC, type in:

```
CMD"S"
```

To re-enter Disk BASIC, yet retain the program, type in:

```
BASIC *
```

The normal BASIC prompt will immediately appear:

```
READY
>
```

You can then LIST or RUN the current program.

Sample Program

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

CMD"S"

DOS Ready
.........................

BASIC *

READY
>

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

CMD"A"
Return to TRSDOS

CMD"A"

This command allows you to return to TRSDOS from Disk BASIC. However, the resident BASIC program can be lost (see BASIC *).

After entering the command, the message:

Operation Aborted

will be displayed.

Sample Use

When you type in:

CMD"A"

the following will be displayed.

Operation Aborted

DOS Ready
.....

CMD"C"
Delete Remarks and Spaces from a Program Line (Compression)

CMD"C",R,S

R **indicates a** Remark (' or :REM) in a program line. This is an option.

S **indicates a** Space in a program line. This is an option.

If neither option is given, both will be used.

This command allows you to delete either spaces or remarks (or both) from a program without re-typing each individual line.

CMD"C",R,S is especially useful when saving programs but using as little disk space as possible in the process.

You can use either or both of the options to execute the operation. If neither option is given, both options will be used, deleting all spaces and remarks.

Disk BASIC remains in control after the command has been executed.

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 wanted to delete the Remarks (lines 850 and 860), type the command:

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

and the program would 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 in either:

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

or

```
CMD"C"
```

Sample Program

```
10 X = 1                    'COUNTING PROGRAM
20 FOR X = 1 TO 10          'COUNT THIS FAR
30 PRINT X
40 NEXT X

CMD"C",R,S

LIST
```

```
10 X=1
20 FORX=1TO10
30 PRINTX
40 NEXTX
```

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, non-invisible files will be displayed.

The drive specification is not optional and must be specified for all drives, including drive 0. If you do not specify which drive, Disk BASIC will not default to drive 0; instead, a Syntax Error message will be displayed.

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"

Disk BASIC will display the last TRSDOS error message displayed whenever you type in and enter CMD"E".

After the command has been executed, you will remain in Disk BASIC.

If no errors occurred prior to the command, the message:

No Error Found

will be displayed.

Example

If you type in:

CMD"E"

and the last TRSDOS error message displayed was an invalid drive number error, Disk BASIC would respond with the message:

Invalid Drive Number

followed by the Disk BASIC prompt,

READY
>

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

It's possible to execute TRSDOS commands directly from BASIC by using CMD"I".

This is similar to CMD"S", except it lets you enter a command (operator or library) for TRSDOS to execute without first entering TRSDOS.

It's also possible to enter Z-80 subroutines directly from BASIC using CMD"I".

As long as BASIC is not overwritten by the execution of a library command, control will return to BASIC; otherwise, control will return to TRSDOS.

Example

```
CMD"I","BASIC PGM"
```

returns you to TRSDOS and executes the command file BASIC PGM.

```
CMD"I","DIR:3"
```

returns you to TRSDOS and displays the Directory for drive 3.

CMD"J"
Calendar Date Conversion

CMD"J", source, destination
'source' is a string variable or expression containing the date which needs to be converted
'destination' is a string variable containing the converted date

conversion format can be one of the following

mm/dd/yy to yy/ddd

-yy/ddd to mm/dd/yy

where 'mm' is a two-digit number specifying the month
where 'dd' is a two-digit number specifying the day of month
where 'ddd' is a three-digit number specifying the day of the year
where 'yy' is a two-digit number specifying the year

This command enables you to convert a calendar date to one of two specified formats.

If you need to change from a format which lists the day in terms of the month and the year (mm/dd/yy) to a format which numbers that day in terms of the the year only (-yy/ddd), source would be the string dd/mm/yy and destination would be the string -yy/ddd.

If the date needs to be changed from a specific day in the year (-yy/ddd) to a day of a particular month (dd/mm/yy), then the contents of the source/destination strings would be reversed.

Example

If you need to know which day of the year October 23, 1980, is, type in the command:

```
CMD"J","10/23/80",D$
```

Next :

```
PRINT D$
```

and Disk BASIC will return with the number of the day in the year.

```
297
```

Conversely, if you need to know the specific date of the 313th day of 1980, type in the command:

```
CMD"J","-80/313",D$
```

Then:

```
PRINT D$
```

Disk BASIC will respond with the correct date:

```
11/08/80
```

Sample Use

```
READY
>CMD"J","08/12/80",D$
READY
>PRINT D$
225
READY
>


READY
>CMD"J","-64/201",D$
READY
>PRINT D$
07/19/64
READY
```

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. This 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).

Example

The command:

```
CMD"L","PROG"
```

will load a progam file named PROG into RAM.

```
CMD"L",P$
```

will load a file which has been specified as P$.

CMD"O"
Alphabetize (Sort) Array Contents

CMD"O"x,y$

'x' is an integer variable whose value is the number of items which is to be sorted

'y$' is the point where the sorting begins (i.e., the string name

The contents of simple **strings** (whose contents are stored in resident memory) can be alphabetized (sorted) by using the CMD"O" command.

'x', the number of items in the string, must be an integer **variable** 'y$' specifies the point at which the alphabetizing is to begin.

Disk BASIC remains in control after the operation has been executed.

Example

Suppose you had a list of six names which needed to be alphabetized. Include the command

```
CMD"O",N%,A$(1)
```

in the program and Disk BASIC will alphabetize the list.

Sample Program

```
10 DEFINT N:N=6
20 DATA BILL, ABE, DEBBIE, FRANK, CARL, MIKE
30 FOR I=1 TO N: READ A$(I):PRINT A$(I);" ";:NEXT I
40 PRINT
```

```
50 CMD"O",N,A$(1)
60 FOR I = 1 TO N: PRINT A$(I);" ";:NEXT I

RUN


BILL, ABE, DEBBIE, FRANK, CARL, MIKE
ABE, BILL, CARL, DEBBIE, FRANK, MIKE
```

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, off-line, 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 Use

```
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), whether or not you set the time when you turned the system on and whether or not the display is on or off.

Example

To turn on the clock display, type in:

CMD"R"

CMD"S"
Return to TRSDOS

```
CMD"S"
```

To exit Disk BASIC, returning control to TRSDOS, simply type in the command:

```
CMD"S"
```

No further message will appear except for the TRSDOS prompt. The resident BASIC program, however, can be lost (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.

```
DOS Ready
.................
```

Sample Program

```
READY
```

```
>
CMD"S"
DOS Ready
.............
```

CMD"T"
Turn Off Clock-Display

CMD"T"

This command stops the updating of the real-time clock display in the upper right corner of the Video Display.

The clock continues to run, however, regardless of whether or not the Display has been stopped with CMD"T".

Example

To stop the clock display update type:

CMD"T"

CMD"X"
Cross-referene of Program Lines

CMD"X",tag
'tag' is one of the following options:
reserved word
variable
string variable (must be enclosed in quotes)

CMD"X" allows Disk BASIC to cross-reference and list the line number of the program lines which contain specified reserved words, string variables, or strings.

Furthermore, the cross-reference will be automatically output to the printer, if a printer is available.

Quotation marks around the tag cannot be included if the tag is a reserved word (such as GOTO, AND, etc.) or if the tag is a variable which has been previously been assigned a value (such as A$ = "mailing list").

Quotation marks around the tag must be included if the variable is either string or literal or if a reserved word is used inside a print statement.

After the command has been typed in, the line number will be displayed as a five-digit number with leading zeros (if applicable).

After the command has been executed, control returns to Disk BASIC.

Example

Your program reads as follows:

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

If you request a cross-reference of all lines which contain the reserved word GOTO, type in:

```
CMD"X",GOTO
```

and Disk BASIC will respond:

```
00850        00870
```

If you request a list of the lines which contain the statement END, the display will be:

```
00880
```

If you want a list of the lines which contain the word GOTO used within quotes (i.e., not as a reserved word) you must type in:

```
CMD"X","GOTO"
```

and the Display will respond:

```
00890
```

Sample Program

```
10 X = 1
20 FOR X = 1 TO 10
30 PRINT X
40 NEXT X
```

```
CMD"X",PRINT
```

```
00030
```

CMD"Z"
Duplicate Output to Video and Printer

CMD"Z"

This command enables all video output to be copied to the printer.

To turn CMD"Z" off, re-type the command.

Video and printer output may be different because of intrinsic differences between output devices and output software.

Using the command will slow down the video output process.

Example

After typing in:

CMD"Z"

program lines such as the following will be simultaneously displayed on the Video and Printer as each line is entered.

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

DEF FN (define function)

DEF FN *var1*(*var2*[,*var*. . .]) = *exp*

where *var1* will be the name of the function, and is any valid variable name
var2 and subsequent var-items are used in defining what the function does
exp is an expression usually involving the variable(s) passed on the left of the equals sign

This statement lets you create your own implicit functions. That is, you only have to call it by name and the implicit function you defined will automatically be performed. Once a function has been defined with the DEF FN statement, you can call it simply by referencing the function name prefixed by FN. You can use it exactly as you'd use one of the intrinsic functions, e.g., SIN, ABS, STRING$.

The type of variable used to name the function determines what type of value the function will return. For example, if the function name has the single-precision attribute, then that function will return a single-precision value – regardless of the precision of the arguments.

Examples:

```
DEFFNA$(TITLE$,GRAPHICS%)=STRING$(LEN(TITLE$),GRAPHICS%)
```

The function A$ will require two arguments – one integer, one string; and it will return a string value.

```
DEFFNRC!(A)=1/(A*A)
```

The function RC! requires one argument, and returns a single-precision value, regardless of the precision of the argument.

The particular variable names you use as arguments in the DEF FN statement are not assigned to the function; when you call the function later, any valid 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.

The function must be defined with at least one argument – even if this argument is not actually used to pass a value to the function. For example:

```
DEF FNR(A)=RND(0)
```

Examples:

```
10 DEFFNMLT(A,B) = A * B
20 INPUT "ENTER ARGUMENTS"; X,Y
30 PRINT "PRODUCT IS"; FNMLT(X,Y)
```

Notice that FNMLT is defined with arguments A,B, but that when the function is called in line 30, variables X and Y are used. Any two valid variable names can be used to pass values to the function.

```
DEF FNR(A,B) = A+INT((B-(A-1))*RND(0))
```

Returns a random number between A and B.

```
DEF FNL8$(A$) = LEFT$(A$,8)
```

Returns first 8 characters of string argument

```
DEF FNX#(A#,B#) = (A#-B#)*(A#-B#)
```

Returns double-precision value of "the square of the difference"

```
100 '          PROGRAM:  STRING
110 ' EXAMPLE OF A STRING DEFFN FUNCTION
120 '
130 ' ******** FUNCTION TO CONCATENATE STRINGS *********
135 '
140 DEF FNADD$ (A$, B$) = A$ + " " + B$
150 CLS: PRINT TAB(15);  "STRING DEFFN EXAMPLE"
160 PRINT: F$ = "": INPUT "ENTER FIRST NAME"; F$
165 IF F$ = "" THEN END
170 INPUT "ENTER LAST NAME"; L$
180 '
190 ' ******* ADD F$ TO L$ WITH 1 BLANK IN BETWEEN *******
200 '
210 Z$ = FNADD$ (F$, L$)
220 PRINT TAB(6); "FULL NAME:    "; Z$
230 GOTO 160
```

```
RUN ENTER

                STRING DEFFN EXAMPLE

ENTER FIRST NAME? JOHN ENTER
ENTER LAST  NAME? DOE ENTER
      FULL NAME: JOHN DOE
```

```
100 '          PROGRAM: MINMAX
110 ' EXAMPLE OF DEFFN FEATURE
120 '
130 ' ******* DEFINE MIN AND MAX FUNCTIONS *******
135 '
140 DEF FNMIN (A, B) = (A + B - ABS (A - B)) / 2
150 DEF FNMAX (A, B) = (A + B + ABS (A - B)) / 2
160 '
170 ' ******* READ 1ST VALUE - CALL IT THE MIN AND MAX *****
180 '          MN IS CURRENT MINIMUM VALUE
190 '          MX IS CURRENT MAXIMUM VALUE
200 '
210 READ MN: MX = MN
220 '
230 ' ******* GET NEXT VALUE AND FIND NEW MIN/MAX ********
240 '
250 READ V: IF V = 99999 THEN 320   'V=99999 MEANS ALL DONE
260 MN = FNMIN (MN, V)   'GET NEW MINIMUM
270 MX = FNMAX (MX, V)   'GET NEW MAXIMUM
280 GOTO 250
290 '
300 ' ******** PRINT RESULTS *******
310 '
320 PRINT "MINIMUM VALUE =", MN
330 PRINT "MAXIMUM VALUE =", MX
340 '
350 ' ******* DATA FOLLOWS - LAST VALUE MUST BE 99999 *******
360 '
370 DATA 1.2, 2, 3, 4.7, 5.332, 0.314, 6, 7, 8.3, 9.57, 99999
```

```
>RUN ENTER
MINIMUM VALUE =  .314
MAXIMUM VALUE =  9.57
READY
>370 DATA -1, 9, 0, 8, 1, 7, 2, 6, 3, 5, 4, 99999 ENTER
>RUN ENTER
MINIMUM VALUE = -1
MAXIMUM VALUE =  9
READY
>_
```

DEFUSR
(define entry address for USR routine)

DEFUSR*n*=*nmexp*

where *n* equals one of the digits 0,1,...,9; if *n* is omitted, 0 is assumed
nmexp specifies the entry address to a machine-language routine.

This statement lets you define the entry points for up to 10 machine-language routines. (Where only one USR routine is available, the entry point address is POKEd into RAM.)

Example:

```
100 DEFUSR3=&H7D00
```

Assigns the entry point 7D00 hex, 32000 decimal, to the USR3 call. When your program calls USR3, control will branch to your sub-routine beginning at hex 7D00.

Here are three ways to get a machine language program into RAM so that it can be accessed via a USR*n* call:

1) Use the TRS-80 Editor Assembler to convert the source code into an object file on tape; then load the tape under the SYSTEM command (use MEMORY SIZE to protect the code from destruction by BASIC).
2) Use the TRSDOS DEBUG program to type in the machine-code routine (then DUMP it to disk for safe-keeping); call DISK BASIC and answer MEMORY SIZE to protect the routine.
3) Have your DISK BASIC routine POKE the routine (decimal values for each byte) into high RAM. MEMORY SIZE should be set during initialization to protect the area you will POKE into.

See USR*n*.

INSTR (string search function)

INSTR([*n*,]*exp1$*,*exp2$*)

where *n* specifies a position in exp1$ where the search is to begin; if *n* is not supplied, 1 is assumed. (Position 1 is defined as the first character in the string.)
exp1$ is the string to be searched
exp2$ 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, starting at the position you specify.

Examples (let A$="ABCDEFG"):

Expression	Result
INSTR(A$,"BCD")	2
INSTR(A$,"12")	0
INSTR(A$,"ABCDEFGH")	0
INSTR(3,"1232123","12")	5

See the EDIT program under MID$= for a sample use of INSTR.

LINE INPUT (input a line from keyboard)

LINE INPUT[*"prompt"*] ;*var$*

where *prompt* is a prompting message

var$ 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:

- When the statement is executed, and the Computer is waiting for keyboard input, no question mark is displayed
- Each LINE INPUT statement can assign a value to just one variable
- Commas and quotes will be accepted as part of the string input
- Leading blanks are not ignored – they become part of *var$*
- The only way to terminate the string input is to press **ENTER**

LINE INPUT is a convenient way to input string data without having to worry about accidental entry of delimiters – because only the **ENTER** key serves as a delimiter. If you want anyone to be able to input information to a program without special instructions, use LINE INPUT and then analyze the resultant string.

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.

Try the following program to get the idea of LINE INPUT.

```
100 '          PROGRAM: LNINPUT
110 ' EXAMPLE OF LINEINPUT STATEMENT
120 '
130 CLEAR 300: CLS
140 PRINT TAB(15); "LINE INPUT STATEMENT": PRINT
150 PRINT: PRINT "*** ENTER TEXT ***"
151 '
152 ' *** GET STRING, THEN PRINT IT ***
153 '
155 A$=""   'SET A$ TO NULL STRING
```

```
160 LINEINPUT "==> "; A$
165 IF A$="" THEN END   'IF STILL NULL STRING, STOP!
170 PRINT A$
180 GOTO 155
```

```
RUN ENTER

                LINE INPUT STATEMENT

*** ENTER TEXT ***
==>             EXAMPLE TEXT
THIS TEXT HAS EMBEDDED LINE FEEDS AND TABS
IN      IT.  LINEINPUT ALSO ALLOWS DELIMITER (, :;"' ETC). ENTER
            EXAMPLE TEXT
THIS TEXT HAS EMBEDDED LINE FEEDS AND TABS
IN      IT.  LINEINPUT ALSO ALLOWS DELIMITER (, :;"' ETC).
==> ENTER
READY
>_
```

MID$= (replace portion of string)

MID$(*var$*,*n1*[,*n2*]) = *exp$*

where *var$* names the string to be changed
n1 specifies the starting position for the replacement
n2 specifies how many characters are to be replaced; if n2 is omitted, LEN(*exp$*) or LEN(*var$*)–*n1*+1 is used, whichever is smaller.

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

Note that the length of the target string (*var$*) is never changed by the MID$= statement. If the replacement string, *exp$*, is too long to fit in the specified portion of var$, then the extra characters at the right of *exp$* are ignored.

However, if you specify the number of characters to be replaced, and this number is larger than the replacement string, then the length of the replacement string overrides the length you specified.

A$="ABCDEFG" at the beginning of each example below:

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

In example 2, the specified replacement length exceeds the length of the replacement string (which is zero); therefore the replacement-string length is used. In effect, no characters are replaced.

Sample program: EDIT

This program accepts an initial string, asks for a **target string** and a replacement string. Then it performs the MID$= replacement and prints the new string. Type in a position equal to zero to stop the program.

```
100 '          PROGRAM: EDIT
110 ' EXAMPLE OF INSTR FUNCTION FOR TEXT EDITTING
115 '
120 CLEAR 800: CLS
130 PRINT TAB(15); "  STRING-FUNCTION EDITOR"
135 '
140 ' ******* GET INITIAL TEXT *******
145 '
150 PRINT: PRINT "ENTER INITIAL TEXT STRING"
160 S$="": LINE INPUT S$: IF S$="" THEN END
165 '
170 ' ******* GET TARGET & REPLACEMENT STRINGS *******
175 '
180 T$="": PRINT: LINE INPUT"     TARGET STRING  "; T$
185 IF T$="" THEN END
190 LINE INPUT "REPLACEMENT STRING  "; R$
195 IF LEN(T$)<>LEN(R$)THEN PRINT"CAN'T CHANGE STRING LENGTH":
    GOTO 180
200 ' ******* MAKE REPLACEMENT(S) AND PRINT NEW STRING *****
210 I=1   'VARIABLE I POSITIONS TO BEGINNING POINT OF SEARCH
220 I=INSTR(I, S$, T$): IF I=0 THEN 150   'I=0 IF NOT FOUND
230 MID$(S$,I)=R$   'MAKE REPLACEMENT
240 PRINT "POSITION - "; I: PRINT S$
250 I=I+LEN(R$): GOTO 220   'ADVANCE POSITION
```

```
RUN ENTER
```

```
                STRING-FUNCTION EDITOR

ENTER INITIAL TEXT STRING
CHANGE "DISC" TO "DISK" EACH TIME IT OCCURS. . (DISC=>DISK)

     TARGET STRING  DISC ENTER
REPLACEMENT STRING  DISK ENTER
POSITION -  9
CHANGE "DISK" TO "DISK" EACH TIME IT OCCURS. . (DISC=>DISK)
POSITION -  48
CHANGE "DISK" TO "DISK" EACH TIME IT OCCURS. . (DISK=>DISK)

ENTER INITIAL TEXT STRING ENTER

READY
>_
```

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 in the range < -32768 +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 USR 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. A RET or JP 0A9A instruction in the routine returns control to the USR call in your BASIC program.

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 USR1 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 OA7FH	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!).

```
                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        ;SOURE 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, as follows.

```
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 DEMICAL CODE FOR HEX PROGRAM *******
320 '
330 DATA 33,0,60,54,255,17,1,60,1,255,3,237,176,201
```

RUN the program. An equivalent BASIC white out routine takes a long time by comparison!

Disk-Related Features

Programs and data are stored as "files" under TRSDOS. Each program or data-set on the disk has its own, distinct file specification=-which includes a name plus identifying information.

Before attempting any disk I/O--including loading and saving BASIC programs, refer to the TRSDOS section and the Notations described under Operation.

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.

Commands discussed under "File Manipulation":

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

Statement and functions discussed under "File Access":

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
GET	Read from disk, random access mode
PUT	Write to disk, random access mode
FIELD	Assign field sizes and names to random access file buffer
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
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$*

where *epx$* 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 wipes out any resident BASIC program, clears all variables, and closes all open files. LOAD with the R option deletes the resident program and 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.

If you attempt to LOAD a non-BASIC file, a DIRECT STATEMENT IN FILE or LOAD FORMAT ERROR will occur.

Examples:

LOAD"PROG1/BAS:2"	Clears resident BASIC program and loads PROG1/BAS from drive 2; returns to BASIC command mode.
10 REM . . . INSTRUCTIONS	Example of chaining two programs – the first may be used to give instructions and then to load the "working" part of the program (PROG2/BAS). Note that line 1000 is equivalent to: 1000 RUN"PROG2/BAS",R
1000 LOAD"PROG2/BAS",R	

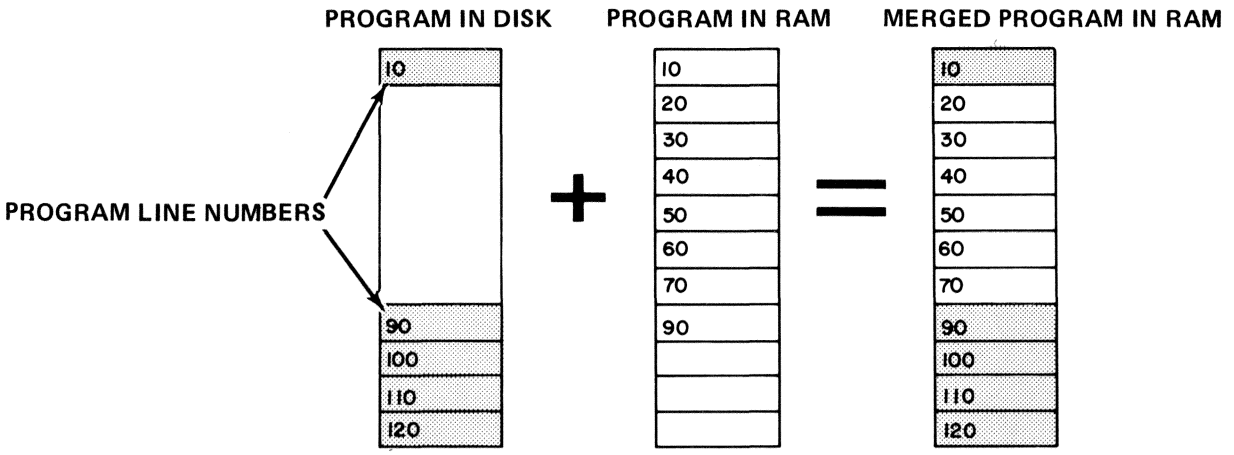
MERGE
(merge disk program with resident program)

MERGE *exp$*

where *exp$* defines a filespec 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 wiped out before the new program *exp$* is loaded. Instead, *exp$* 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$*.



MERGE provides a convenient means of putting modular programs together. For example, an often-used set of BASIC subroutines can be tacked onto a variety of programs with this command.

For example, suppose the following program is in RAM:

```
10 REM...MAIN PROGRAM
20 GOSUB 1000
30 REM...MORE PROGRAM LINES HERE
999 END
1000 REM...NEED TO ADD SUBROUTINES HERE
1010 REM...SO USE MERGE COMMAND
1020 PRINT"SUBROUTINE NOT AVAILABLE":RETURN
```

And suppose the following program is stored on disk in ASCII format:

```
1000 REM...BEGINNING OF SUBROUTINE
1010 PRINT"EXECUTING SUBROUTINE..."
1020 REM...MORE PROGRAM LINES HERE
1100 RETURN
```

Assuming the subroutine program is named SUB/TXT, then we could MERGE it with the statement:

```
MERGE"SUB/TXT"
```

and the resultant program in RAM would be:

```
10 REM...MAIN PROGRAM
20 GOSUB 1000
30 REM...MORE PROGRAM LINES HERE
999 END
1000 REM...BEGINNING OF SUBROUTINE
1010 PRINT"EXECUTING SUBROUTINE..."
1020 REM...MORE PROGRAM LINES HERE
1100 RETURN
```

Note that MERGE closes all files and clears all variables. Upon completion, BASIC returns to the command mode.

RUN"program"
(load and execute a program from disk)

RUN *exp$* [,R]

where *exp$* defines the filespec for a BASIC program stored on disk. R leaves open files open

If the R-option is not selected, all open files will be closed.

When the command is executed, any resident BASIC program will be replaced by the program contained in *exp$*.

Suppose you save the following program on disk with the name "PROG1/BAS":

```
10 PRINT"PROG1 EXECUTING..."
20 RUN"PROG2/BAS"
```

And save this program on disk with the name "PROG2/BAS":

```
10 PRINT"PROG2 EXECUTING..."
20 RUN"PROG1/BAS"
```

Now type:
RUN"PROG1/BAS ENTER
and you'll see a simple example of program chaining.
Hold down the BREAK key to interrupt the program chain.

SAVE (save program onto disk)

SAVE *exp$* [,A]

where *exp$* defines the file-name and optional extension, password, and drive to be used. If the file-name already exists, its previous contents will be lost as the file is re-created.

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. This is the way BASIC programs are stored in RAM.

Using the ASCII option makes it possible to do certain things that cannot be done with compressed-format BASIC files.

Examples:

- The MERGE command requires that the disk file be in ASCII form.
- You can use the TRSDOS commands LIST and PRINT with ASCII-format files.
- Programs which read in other programs as data will typically require that the data programs be stored in ASCII.

Useful conventions for placing extensions on BASIC programs:
For compressed-format programs, use the extension /BAS.
For ASCII format programs, use the extension /TXT.

Examples of SAVE command:

```
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 non write-protected drive.

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**

then 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
Assign Buffer to a File and Set Mode

OPEN "expl$",nmexp,"exp2$",n

'expl$' is a string expression or constant. Only the first character is significant; this character specifies the mode in which the file is to be opened:

expl$=	access mode
I	sequential input
O	sequential output
R	random I/O

Note: Must be enclosed in quotes.

'nmexp' has a value from 1-15, and tells BASIC which buffer to assign to the file specified by exp2$

'exp2$' defines a TRSDOS file specification (Must be enclosed in quotes.)

'n' **specifies the number of bytes in one logical record and is a decimal number between 1-256. If zero, 256 used. This number cannot exceed the number specified FIELD and is only valid for Random length files.**

This statement makes it possible to access a file. expl$ determines what kind of access you'll have via the specified buffer; nmexp determines which buffer will be assigned to the file; exp2$ names the file to be accessed; length determines the number of files.

Note: nmexp (buffer number) cannot exceed the number you entered for the FILES? question during initialization. If you entered:

HOW MANY FILES? 2,V <ENTER>

then nmexp can have the value of 1 or 2.

Examples

```
100 OPEN "O",1,"CLIENTLS/TXT"
```

Opens the file CLIENTLS/TXT for sequential output. Buffer 1 will be used. If the file does not exist, it will be created. If it already exists, then its previous contents are lost. (This is explained under "Sequential I/O Techniques".)

```
100 OPEN "I",1,"PROG1/TXT:1"
```

Opens the file PROG1/TXT on drive 1 for sequential input. Buffer 2 assigned to the file, If PROG1/TXT does not exist on drive 1, an error message is returned since you can't input from a non-existent file.

```
100 INPUT "MODE (I,O,R)";MODE$
110 INPUT "BUFFER NUMBER";BUFFER$
120 INPUT "FILE SPECIFICATION";FILESPEC$
130 OPEN MODE$, BUFFER%, FILESPEC$
```

This sequence of statements lets you provide the arguments for the OPEN statement during program execution. The first character of MODE$ sets the acess mode, BUFFER% determines which buffer will be used, and FILESPEC$ gives the file specification.

```
OPEN"R",2,"DATA/BAS.SPECIAL",2
```

OPENs the file DATA/BAS with password SPECIAL, in the Random I/O mode, using buffer 2 and indicating record lengths of 2 files. If DATA/BAS does not exist, it will be created on the first non write-protected drive.

While a file is open, it is referenced by the buffer-number which was assigned to it. Examples:

```
GET buffer-number
PUT buffer-number
PRINT # buffer-number
INPUT # buffer-number
```

All these statements will reference a file which was OPENed via buffer-number. The mode must be correct.

Once a buffer has been assigned to a file with the OPEN statement, that buffer cannot be used in another OPEN statement. You must close it first.

More on Buffer Assignment

Two or more buffers may be assigned to the same file for sequential input (I-mode). However, only one buffer at a time may be assigned to a file for sequential output (O-mode) or random access (R-mode).

For example:

```
10 OPEN "I",1,"TEST/TXT:1"
20 OPEN "I",2,"TEST/TXT:1"
```

Now TEST/TXT can be accessed via buffers 1 and 2 for sequential input.

Do not leave disk files OPEN longer than you have to. This is because the disk files are especially vulnerable to power failures and voltage transients, accidental removal of diskettes, etc.

For example, it is NOT good practice to OPEN a file at the beginning of a program, and leve it OPEN until the end of the program. Instead, you should OPEN the file when you are ready to read or write the data, and CLOSE the file when you've finished.

CLOSE (close access to the file)

CLOSE [*nmexp* [,*nmexp* . . .]]

where *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 of CLOSE statements:

```
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 an open file – 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.

The following actions and conditions cause all files to be automatically closed:

NEW ENTER
RUN ENTER
MERGE *filespec* ENTER
EDITing a file
Adding or deleting program lines
Execution of the CLEAR *n* statement
Disk Errors

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-buffer 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 <EN> (a carriage return) is preceded by <LF> (a line feed), the <EN> 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 <LF> character from the keyboard, press the down-arrow character. To enter the <EN> character, press <ENTER>.)

This exception applies to all cases noted below where <EN> is said to be a terminator.

Numeric Input

Suppose the data image on disk is

␣1.234␣-33␣␣27␣ <EN>

<EN> 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 <EN> 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 < EN > –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 < EN > .
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)
<EN>
<EN> <LF>
␣[␣ ...][<EN>]
␣[␣ ...][<EN><LF>]

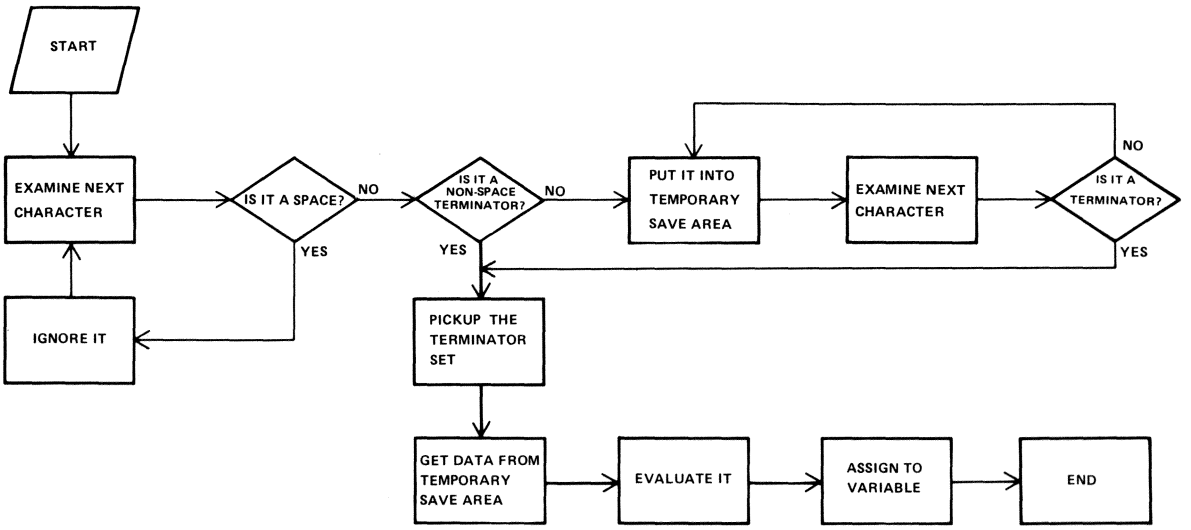
Quoted-string terminator sets

end of file encountered
255th data character encountered
" (double quote)
" [␣...][,]
" [␣...][<EN>]
" [␣...][<EN> <LF>]

Unquoted-string terminator sets

end of file encountered
255th data character encountered
,
<EN> [<LF>]

Here's a flow chart describing how INPUT# assigns data to a variable:



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␣ <EN><LF>␣8.2E4␣␣7000<EN>	A=123.45 B=82000 C=7000
2	␣␣␣3<LF><EN> 4 <EN>5 <EN> A12eof	A=34 B=5 C=0
3	1,,2,3,4 <EN>	A=1 B=0 C=2
4	1,3,end-of-file	A=1 B=2 C=0 end of file error

In Example 2 above, why does variable C get the value 0? When the input reaches the end of file, it terminates the 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 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. <EN>	A$=ROBERTS B$=J.
3	THE WORD "QUO",12345.789 <EN>	A$=THE WORD "QUO" B$=12345.789
4	BYTE<LF> <EN> UNIT OF MEMORY eof	A$=BYTE<LF> <EN> UNIT OF MEMORY B$=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 <EN> is preceded by an <LF>, therefore it does not terminate the first string; both <LF> and <EN> are included in A$.

Technical Note: The above discussion ignores the role of the input buffer in the sequential input process. Actually, DISK BASIC always reads in 256-byte data records into the buffer, and then sorts through what's in the buffer to "satisfy" the INPUT# variable list. That's why

```
100 INPUT#1, A%
200 INPUT#1, B%
```

do not necessarily require two disk accesses. The 256-byte record in the buffer can contain enough data for A%, B% and more.

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 <EN> character which is not preceded by <LF>
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, <LF> <EN> pairs – are included in the string.

For example, if the data looks like:

```
10 CLEAR 500   <EN>
20 OPEN "I",1,"PROG"   <EN>
.
.
.
```

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␣ <EN>

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␣ <EN>

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 <EN>

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 <EN>

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 <EN> –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$="DOE, JOHN Q." B$="100-01-001"

If you use

```
PRINT#1,A$;", ";B$
```

the disk image will be:

DOE, JOHN Q.,100-01-001 <EN>

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 <EN>

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$(10)	<LF> Line Feed
CHR$(13)	carriage return (<EN>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. You could create a report file this way, which then could be LISTed or PRINTed (TRSDOS commands).

Or you could use this option to control how many characters of a value are written to disk.

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   <EN>
```

(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.

Technical Note: The above discussion ignores the role of the output buffer in the sequential write process. Actually, the data is first placed into the buffer, and then, as 256-byte records are filled, the data is written to the disk file. That's why there isn't always a disk access during execution of each PRINT# statement.

Random Access Statements

FIELD
(organize a random file-buffer into fields)

FIELD *nmexp,nmexp1* AS *var1$* [,*nmexp2* AS *var2$* . . .]

where *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

Before FIELDing a buffer, you must use an OPEN statement to assign that buffer to a particular disk file (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 256 bytes which can store data for transfer from disk storage to BASIC or from BASIC to disk. 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).

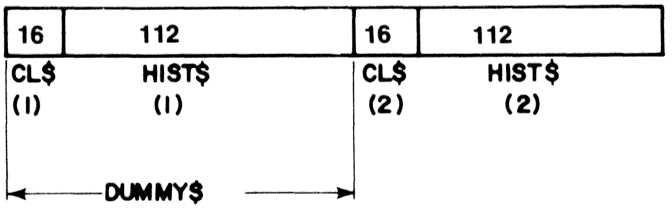
During random input, the GET statement places data into the 255-byte buffer, where it can be accessed using the field names assigned to that buffer. During random output, LSET and RSET place data into the buffer, so you can then PUT the buffer contents into a disk file.

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*]

where *nmmexp1* 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*

When BASIC encounters the GET statement, it looks at the buffer's control block, and obtains:

- the information needed to access the file
- the mode in which this buffer was set up (must be R)
- the current record number
- The EOF (end-of-file) record number, i.e., the highest numbered record in the file
- lots of other information for internal use

BASIC then reads record *nmexp2* from the file and places it into the buffer. If you omit the record number, it 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 GET 1	GET record 26 into buffer 1

If you attempt to GET a record whose number is higher than that of the end-of-file record, BASIC will fill the buffer with hex zeroes, 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*]

where *nmexp1* specifies a random access file buffer, *nmexp*=1,2,...,15

nmexp2 specifies the record number in the file, *nmexp2*=1,2,.., up to 335, depending on how much space is available on the disk; 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 PUTing 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.

This has an important implication. When you PUT a record whose number exceeds the EOF record number, space is allocated on the disk to accommodate the new highest record number plus all lower-numbered records.

Examples (assume a file named SAMPLE/BAS exists and that you have previously written 10 records to it, so that LOF=10):

Program statement	Effect
1000 OPEN"R",1,"SAMPLE/BAS"	Open SAMPLE/BAS for random address under buffer 1
1010 FIELD 1,.....	Prepare buffer
1020 LSET	Place data in buffer
1030 PUT 1	Copy buffer contents into current record (=#1)
1035 LSET	Place data in buffer
1040 PUT 1,30	Acquire disk space for records 2 through 30 and copy buffer contents into record 30; set LOF=30
1045 LSET	Place data in buffer
1050 PUT 1,25	Copy buffer contents into record 25
1055 LSET	Place data in buffer
1060 PUT 1	Copy buffer contents into current record (=#26)

LSET and RSET
(place data in a random buffer field)

LSET *var$* = *exp$* and RSET *var$* = *exp$*

where *var$* is a field name

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$*)

where *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$*)

where *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$*)

where *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, 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*)

where *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
```

LOF (get end-of-file record number)

LOF(*nmexp*)

where *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.

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 zeroes, 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*)

where *nmexp* is evaluated as a double-precision number

MKI$(*nmexp*)

where *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*)

where *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:

```
ASC(MKI$(I%)) equals the lsb of I%, i.e., (I% AND 255)
ASC(RIGHT$(MKI$(I),1))=the msb of I%, i.e., INT(I%/256)

LSET AVG$=MKS$(0.123)
```

AVG$ would typically reference a four-byte random buffer field. Now it contains a representation of the single-precision number 0.123.

```
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$, converted single-precision values; YR$ 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. (Direct access is also called random 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`

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.

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 sequentialy, 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 one. This matters if you want to Close the file and re-Open it for `Random access;` in such a case, you must specify a record length of 1.

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 X'0D'

For example, the stored data would start out:

IN−>CM,ϕ2.54001ϕ X'0D'

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.M, 4046.86
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.

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$(9), FACTR(9)          'allows for up to 10 data pairs
20 OPEN"I",1,"METRIC/TXT:1"
25 I%=0
30 IF EOF(1) THEN 70
40 INPUT#1, UNIT$(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$(ITEM%)
130 NEXT
140 PRINT @ (19,0), "Which conversion (0-6)";
150 INPUT CHOICE%
160 INPUT"Enter English quantity";V
170 PRINT"The Metric equivalent is" V*FACTR(CHOICE%)
180 INPUT"Press <ENTER> to continue";X
190 PRINT @ (19,0), CHR$(24)     '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 entires into the English-Metric conversion file. You can't simply re-Open the file for sequential output and PRINT# the extra data – that would immediately set the EOF to the beginning of the file, effectively destroying the file's previous contents. Do this instead:

1) Open the file for sequential input
2) Open another new data file for sequential output
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.

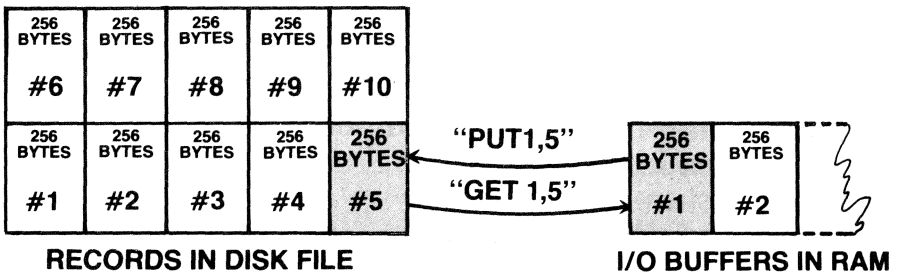
The last advantage listed above is also the "hard part" of direct access. It takes a little extra thought.

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 the following illustration, we assume a record length of 256.



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 righ
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
```

Lines 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 above example shows the necessary sequences to read and write using direct 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 direct access for file maintenance. But before looking at the program, study this general procedure for creating and maintaining files via direct 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 <EN> OR <EN> IF OK"; W$
190 D$ = "" : PRINT "TYPE NEW DEF'N <EN> OR <EN> IF OK?" :
    LINE INPUT D$
200 INPUT "TYPE NEW SEQUENCE NUMBER OR <EN> 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 <EN> TO READ NEXT ALPHA. ENTRY,":
    PRINT" OR RECORD # <EN> FOR SPECIFIC ENTRY,":
    INPUT " OR 0 <EN> 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

100	Field overflow
102	Internal error
104	Bad file number
106	File not found
108	Bad file mode
114	Disk I/O error
122	Disk full
124	Input past end
126	Bad record number
128	Bad file name
132	Direct statement in file
134	Too many files
136	Disk write-protect
138	File access

INDEX

Subject Page

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.

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.

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 TC **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

8749167-980-SP

PRINTED IN U.S.A.