

**TANDY**

TRS-80

**COMPUTER  
PRODUCTS**

MODEL 2000

MS™-DOS 2.0

DRIVE A

CAT. NO. 26-5103

# MS™-DOS/BASIC

MS™-DOS 2.0, © 1983, MICROSOFT CORP., ALL RIGHTS RESERVED, LICENSED TO TANDY CORP.  
GW™ BASIC, © 1983, MICROSOFT CORP., ALL RIGHTS RESERVED, LICENSED TO TANDY CORP.  
MODEL 2000 BIOS SOFTWARE, © 1983, TANDY CORP., ALL RIGHTS RESERVED

02.11.01

**MODEL 2000**

Custom manufactured in USA  
by Tandy Corporation.



Cat. No. 26-5103

MS<sup>TM</sup>-DOS



TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK AND TANDY  
COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM A RADIO SHACK  
COMPANY OWNED COMPUTER CENTER, RETAIL STORE OR FROM A RADIO SHACK  
FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

## LIMITED WARRANTY

### I. CUSTOMER OBLIGATIONS

- A CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment") and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.
- B CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

### II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

- A For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. This warranty is only applicable to purchases of Radio Shack and Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores and from Radio Shack franchisees and dealers at its authorized location. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy, in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.
- B RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.
- C Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.
- D Except as provided herein, Radio Shack makes no express warranties, and any implied warranty of merchantability or fitness for a particular purpose is limited in its duration to the duration of the written limited warranties set forth herein.
- E Some states do not allow limitations on how long an implied warranty lasts, so the above limitations may not apply to CUSTOMER.

### III. LIMITATION OF LIABILITY

- A Except as provided herein, 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 "Equipment" or "Software" sold, leased, licensed or furnished 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 the "Equipment" or "Software". In no event shall Radio Shack be liable for loss of profits, or any indirect, special, or consequential damages arising out of any breach of this warranty or in any manner arising out of or connected with the sale, lease, license, use or anticipated use of the "Equipment" or "Software".
- Notwithstanding the above limitations and warranties, Radio Shack's liability hereunder for damages incurred by customer or others shall not exceed the amount paid by customer for the particular "Equipment" or "Software" involved.
- B RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.
- C No action arising out of any claimed breach of this Warranty or Transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.
- D Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitations or exclusions may not apply to CUSTOMER.

### IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on one computer, subject to the following provisions:


- A Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
- B Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
- C CUSTOMER may use Software on one host computer, and access that Software through one or more terminals, if the Software permits this function.
- D CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on one computer, and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
- E CUSTOMER is permitted to make additional copies of the Software, only for backup or archival purposes, or if additional copies are required in the operation of one computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.
- F CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.
- G All copyright notices shall be retained on all copies of the Software.

### V. APPLICABILITY OF WARRANTY

- A The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party, for lease to CUSTOMER.
- B The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

### VI. STATE LAW RIGHTS

The warranties granted herein give the original CUSTOMER specific legal rights, and the original CUSTOMER may have other rights which vary from state to state.





MS™-DOS Software:  
Copyright 1984 Microsoft Corporation.  
Licensed to Tandy Corporation.  
All Rights Reserved.

Model 2000 BIOS Software:  
Copyright 1984 Tandy Corporation.  
All Rights Reserved.


MS™-DOS Commands Reference Manual:  
Copyright 1984 Microsoft Corporation and Tandy Corporation.  
Licensed to Tandy Corporation.  
All Rights Reserved.

Reproduction or use without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information contained herein.




# About MS-DOS


---



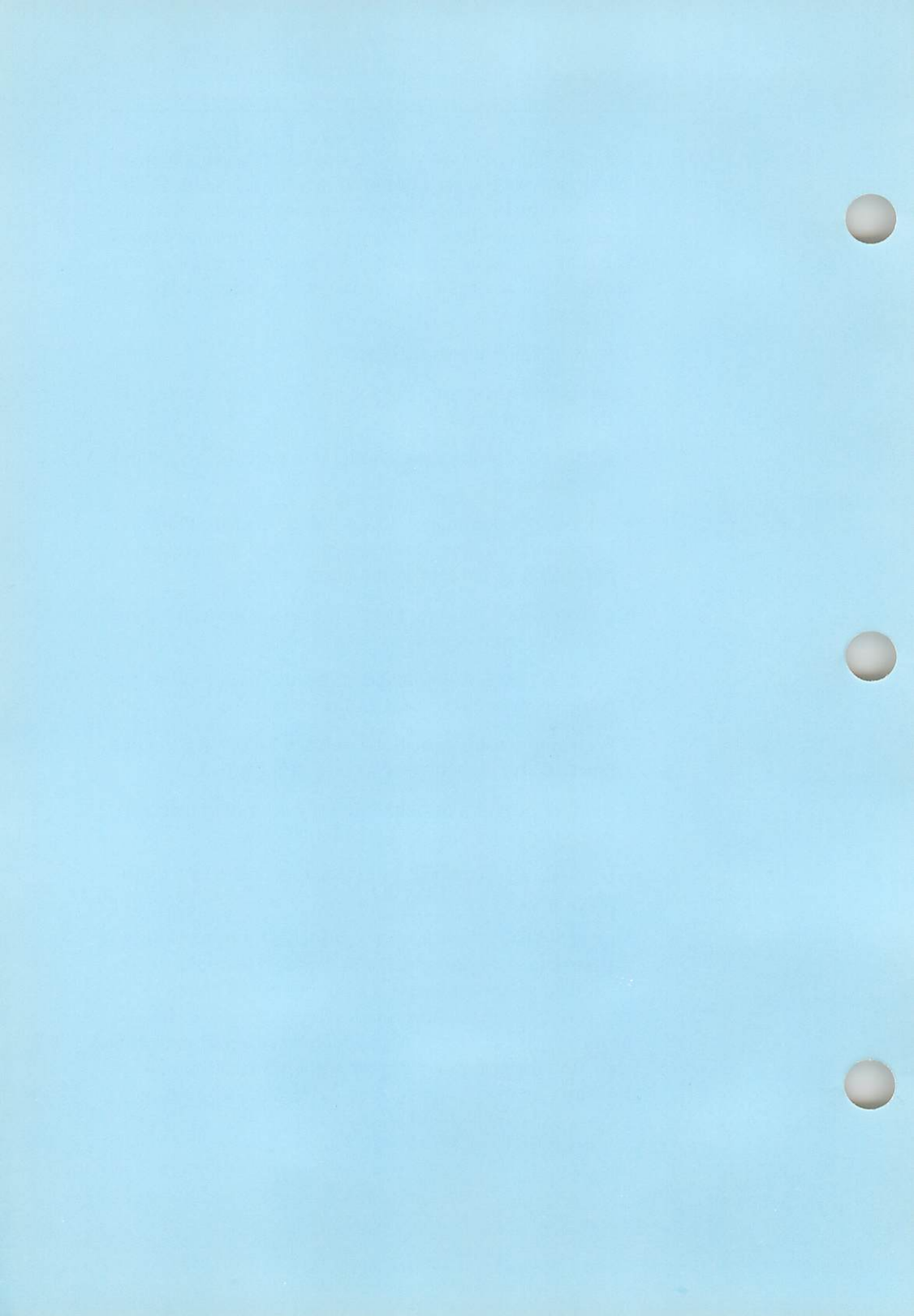
MS-DOS™ (pronounced em-es-dos) is a popular, sophisticated, state-of-the-art operating system for your Model 2000. For the beginner, who uses the system mainly to run applications, MS-DOS's "everyday" commands are easy to use. For the advanced user, who wants an efficient and versatile system, MS-DOS offers a wide variety of special features.

The MS-DOS features include:

- 
- An editor program, EDLIN, to help you create and change your files.
  - A command storage area, to let you re-enter or edit the last command.
  - "Piping" of commands to let you give more than one command at a time to the system. One command's output becomes another command's input.
  - A "batch file" process, to let you store a series of commands to execute whenever you want.
  - "Redirection" of command input or output, so it can come from or go to files and devices.
  - "Filtering" (transformation) of input before it is output (such as the alphabetical sorting of information).
  - A linker program to help you write your own machine-language programs.
  - DEBUG, a program to help you test and correct your programs.




One of MS-DOS's greatest strengths is its multi-level directory structure. Suppose that you work on several projects — or that other people use your computer. With MS-DOS, you can keep related information in small, easy-to-handle groups. You can see, at a glance, how material is organized. And the computer can pinpoint information quickly for rapid access.





# About This Manual

---




This manual shows how you can use MS-DOS to store, retrieve, and manipulate information on disk. It is divided into five sections: "Introduction to MS-DOS," "Commands Reference," "EDLIN," "The Linker," and "DEBUG."

**Before reading this manual or using any commands, be sure to read your *Introduction to Model 2000* manual.** It explains everything you need to know to start up MS-DOS and to use it to run application programs. It also shows how to use the most commonly used commands.

## Section I, "Introduction to MS-DOS"

This section includes four chapters designed to introduce you to the system, its commands, and the many features that help you use commands efficiently.



We strongly recommend that you read all of Section I before attempting to use MS-DOS. Be sure, at least, to read Chapter 1, "Organization of Information in MS-DOS." It explains the MS-DOS directory structure. After you are familiar with this structure, you should be able to quickly understand the sample uses and the examples of the commands in Section II, "Commands Reference."


## Section II, "Commands Reference"

This section contains an alphabetical listing of all MS-DOS commands and shows how to use each.

## Section III, "EDLIN"

This section shows how to use EDLIN to create, update, and save files. Using this special program, you can delete, insert, edit, or display lines in your source program or text files.

## Section IV, "The Linker"



This section is for assembly-language programmers only. It shows how to use the linker program to combine several programs into one relocatable load module (a program that you can run).

---

## Section V, “DEBUG”

This section, also, is for assembly-language programmers only. It shows how to use the DEBUG program to test and edit your executable object files.



### Terms

Below is a list of terms that we use frequently in this manual. Throughout, as in the list, *italicized words* represent variable information that you must supply. All terms are explained in more detail in the Chapter 1.

disk file	A disk area in which you store information.
directory	A disk area that keeps track of your files. Each disk may have several directories, each of which may contain several subdirectories or files or both.
<i>pathname</i>	The name that directs the operating system to a file's location on disk. The MS-DOS pathname has the general form <i>drive:\path\filename.ext</i>
<i>drive</i>	Specifies the disk that contains the file.
<i>path</i>	Specifies the directory or subdirectory that contains the file.
<i>filename</i>	The name you give a file (1-8 characters).
<i>ext</i>	An optional extension (1-3 characters). you can use to further identify your file.
parameter	A variable item of information that customizes a command.
current drive	The drive you are “in” (the drive that contains the disk you are using).
current directory	The directory you are in.
root directory	The first directory level on any disk. All other directories are subdirectories of the root directory.



# Contents

---

## SECTION I / INTRODUCTION TO MS-DOS ..... 5

### Chapter 1 / Organization of Information in MS-DOS..... 7

Entering a Command .....	7
Executing a Program.....	7
Organization of the File System .....	8
Names.....	10
Pathnames .....	12
Device Names .....	13
Directories.....	13
Using Directories.....	13
Creating Directories .....	15
Deleting Directories .....	17
The Current Drive.....	17
The Current Directory .....	18
Home Directories .....	20
Anonymous Directory Names.....	21

### Chapter 2 / Introduction to Commands ..... 23

Types of MS-DOS Commands .....	23
Internal Commands.....	24
External Commands .....	24
Command Parameters.....	25
Information Common to all MS-DOS Commands... ..	26
Input and Output .....	27
Redirecting Input.....	27
Redirecting Output .....	27
Filters .....	28
Command Piping .....	29

**Chapter 3 / Batch Files: Executing  
Several Commands..... 31**

The Batch File.....	31
Creating a Batch File.....	31
REM and PAUSE.....	32
Executing a Batch File.....	32
A Word of Advice about Batch Files.....	33
Summary of the Batch File Process.....	34
The Autoexec.bat File.....	34
Creating an Autoexec.bat File.....	34
Batch Files with Replaceable Parameters.....	35
Creating a Batch File with Replaceable Parameters.....	36
Executing a Batch File with Replaceable Parameters.....	36
Sample Use of Replaceable Parameters.....	37
Reminders about Batch Files.....	38

**Chapter 4 / Correcting Errors and  
Editing Commands..... 41**

The Template.....	41
The Editing Keys.....	42
Sample Uses of Editing Keys.....	43
The Control Character Keys.....	45

**SECTION II / COMMANDS REFERENCE..... 47**

Entry Organization.....	47
Syntax Notation.....	48
Synonymous Keywords.....	48
A Special Note to Hard Disk Users.....	48
Command Entries.....	50

## **SECTION III / THE LINE EDITOR (EDLIN).....153**

Starting the EDLIN Program. ....	153
Editing an Existing File. ....	154
Creating a File and Editing It. ....	154
Saving Files . . . . .	155
Special Editing Keys . . . . .	156
Sample Uses for the Editing Keys. ....	157
Creating a File . . . . .	157
Entering Information in the Sample File . . . . .	157
EDLIN Commands. ....	165
Error Messages . . . . .	195

## **SECTION IV / THE LINKER .....199**

### **Chapter 1 / Basic Information .....201**

Overview of the Linker. ....	201
Linking Object Modules and Producing a Run File . . . . .	201
Resolving External References . . . . .	201
Producing a List File . . . . .	201
The VM.TMP (Temporary File) . . . . .	202
Definitions . . . . .	203
Segment . . . . .	205
Group . . . . .	205
Class . . . . .	206
Command Prompts. ....	206
Command Characters . . . . .	208
Linker Switches . . . . .	209
Starting the Linker . . . . .	211
Method 1: Keyboard Responses. ....	211
Method 2: Responses on Command Line . . . . .	212
Method 3: Response File . . . . .	213
Sample Link Session . . . . .	214

**Chapter 2 / Technical Information.....217**

Definitions ..... 217

- Alignment ..... 217
- Combine Type ..... 218
- Canonical Frame ..... 218

How the Linker Combines and Arranges Segments . 218

Segment Addresses ..... 221

How the Linker Assigns Addresses ..... 222

Relocation Fixups ..... 222

- Short References ..... 223
- Near Self-Relative References..... 223
- Near Segment-Relative References..... 223
- Long References ..... 224



**Chapter 3 / Linker Error Messages.....225**

**SECTION V / DEBUG .....229**

How to Start DEBUG..... 229

Commands ..... 230

Parameters ..... 232

Command Entries ..... 235



**APPENDIX A / PROBLEMS AND ERROR  
MESSAGES .....265**

**APPENDIX B / COMMAND QUICK  
REFERENCE .....276**

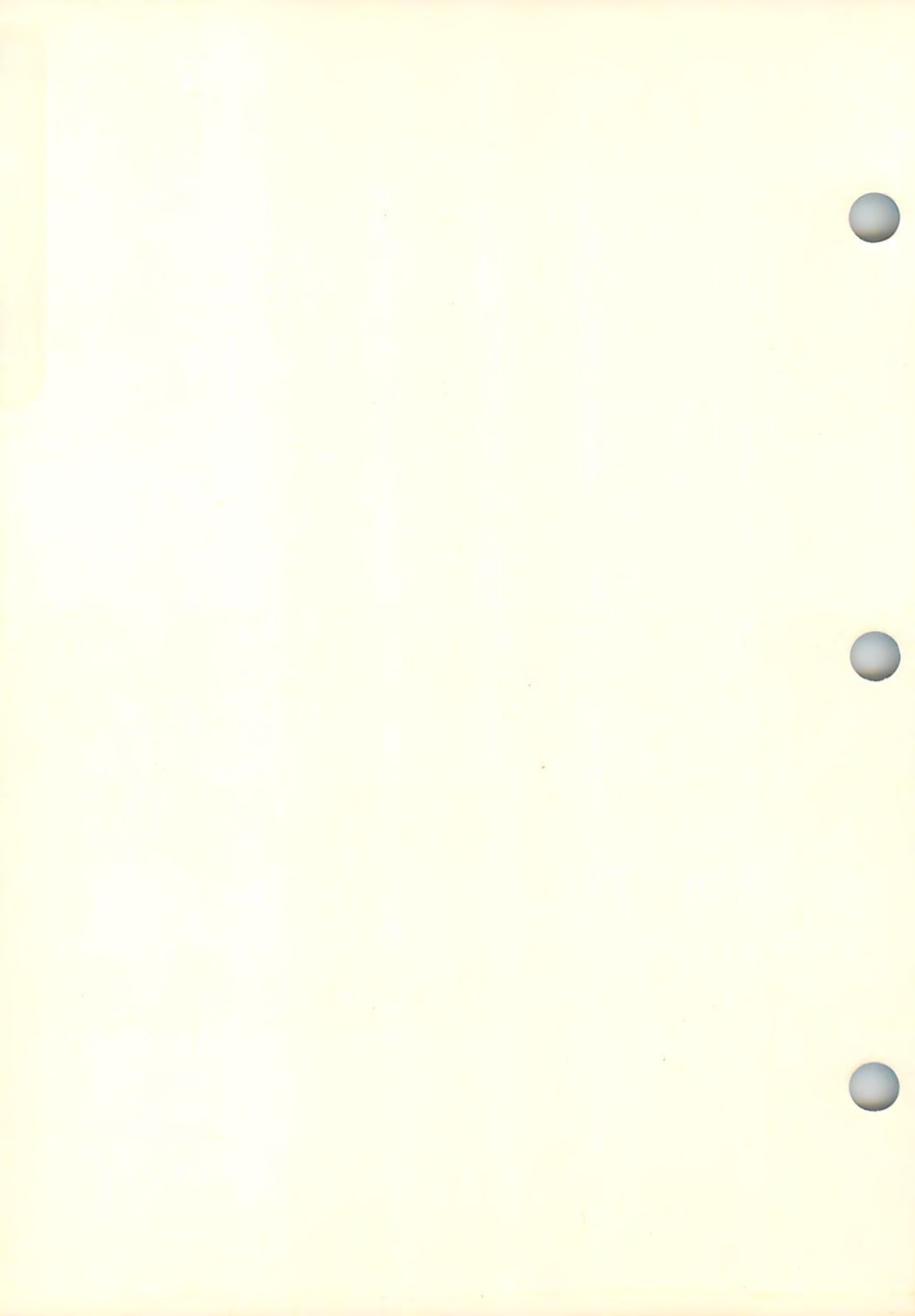
**APPENDIX C / HOW TO CONFIGURE YOUR  
SYSTEM .....279**



**APPENDIX D / ASCII AND SCAN CODES.....281**

**APPENDIX E/ASCII CHARACTER  
CODES ..... 285**








# Section I

---


## Introduction to MS-DOS




This section provides the background information you need to make the best use of Section II, "Commands Reference." It also introduces you to MS-DOS's special features.

Chapter 1, "Organization of Information in MS-DOS" is essential to understanding Section II. **Be sure to read it before using the system.**

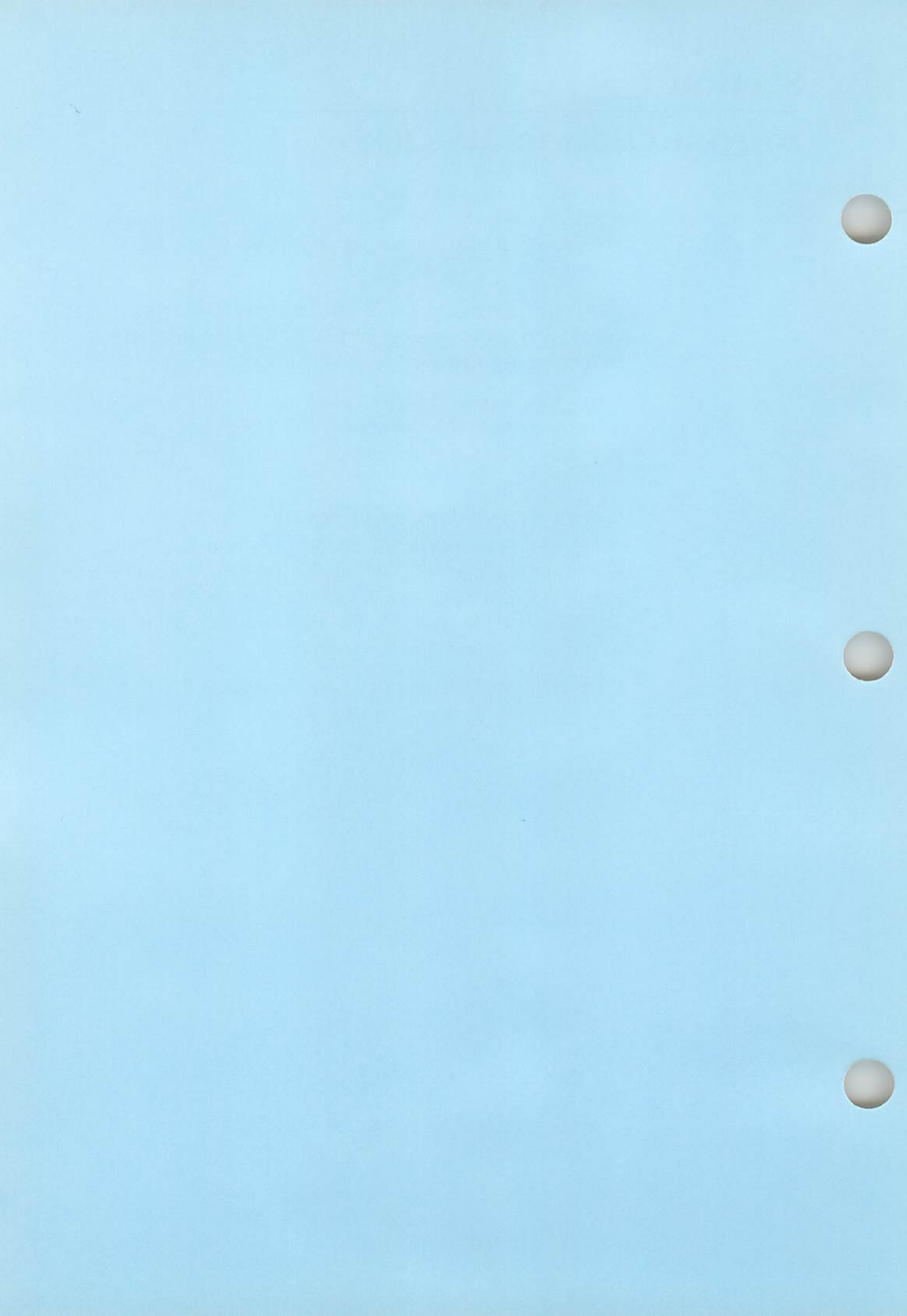
Chapter 2, "Introduction to Commands," familiarizes you with the many ways you can use MS-DOS commands. It also explains the difference between the types of commands.



Chapter 3, "Introduction to Batch Files," shows how to create and save a series of commands in a batch file. Whenever you want, you can run the file, instead of entering all the commands again. Chapter 3 also explains the Autoexec.bat file, a batch file that runs automatically whenever you start up your computer.



Chapter 4, "MS-DOS Editing Keys," shows how to correct any errors you make when typing commands.



# Chapter 1

---

## Organization of Information in MS-DOS

After you turn on your computer and start up MS-DOS as instructed in your *Introduction to Model 2000* manual, MS-DOS prompts you to enter the date and time. After you do this, the screen displays the system prompt:

A>

This means that you are at the MS-DOS command level. At this level, you can execute a program or a command.

**Note:** To perform any other operation, your system must be under the control of an application program.

If an error occurs while your computer is under the control of MS-DOS, the screen displays one of the error messages listed in Section II or Appendix A.

If you get an error not listed, it came from an application program. See the application program manual for an explanation of the error message.

## Entering a Command

You can enter a command whenever the screen displays the system prompt. The command may have up to 125 characters, including any combination of upper- or lower-case letters. End each command by pressing **ENTER**.

For example, type

CLS **ENTER**

and MS-DOS executes the CLS command, which clears the screen and displays the system prompt.

## Executing a Program

You can also execute a program (such as Inventory Control or Accounts Payable) at the system prompt. If what you enter is not a recognized command, MS-DOS checks to see if it is the name of a program. If MS-DOS finds a matching program file, it loads and executes the file. Otherwise, the screen displays an error message.

## Organization of the File System

**Hard Disk Users:** The rest of Chapter 1 is written as if you have a floppy disk system. Thus, when reading, you should substitute your Drive C (assumed to be formatted and initialized) when we refer to Drive A and your Drive A when we refer to Drive B.

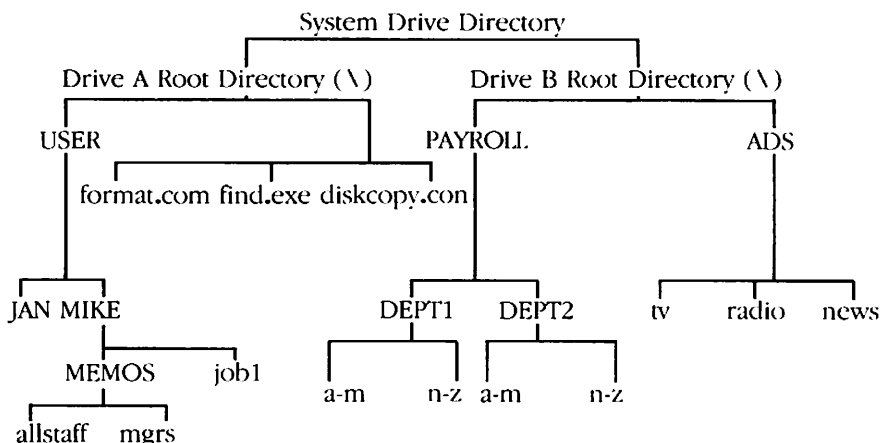
Disks are devices that store information, both text and programs, in separate “files.” MS-DOS handles files in a number of ways designed to help you organize information easily and well.

For instance, with MS-DOS, you can collect groups of files into “directories,” much as you would use a file cabinet drawer to collect all file folders pertaining to a particular subject. Directories, in turn, can be collected into larger directories.

When you work with MS-DOS files and directories, it's important to remember this multi-level organization. It lets you build downward, branching out as you go — in effect, creating an upside-down tree of files and directories.

Each user on your system can organize material into separate, easy-to-handle groups without affecting anyone else's material. Both you and MS-DOS can easily locate stored material.

Here is a simplified diagram of a typical MS-DOS disk.



**Note:** In the commands in this manual, directory names are in upper-case and filenames are in lower-case. This is only to help you distinguish between directories and files. With MS-DOS, you can type all names in upper- or lower-case or in any combination of the two.

When you receive your MS-DOS system disk, it has only one directory, the “root” directory. The root contains all external command files. “Internal commands” are built into the computer. “External commands” are those that reside on disk. Format.com, Find.exe, and Diskcopy.com are only a few of MS-DOS’s external commands.

The Drive A root directory is the root from which the rest of the disk’s file system “grows.” The shorthand notation for a root directory is a backwards slash (\).

In the system diagrammed here, the user has added the USER directory to Drive A. This system also contains a second disk, with its own root directory. (Root directories are automatically created when you initialize a disk using the FORMAT command.)

On the Drive B disk, the user has created directories called PAYROLL and ADS. PAYROLL, in turn, contains two other directories, each of which contains two files. The ADS directory contains three files.

## Names

Each file has a name. Names can include from 1 to 8 characters. Usually, you'll use letters and numbers:

- Upper-case letters (A-Z)
- Lower-case letters (a-z)
- Decimal digits (0-9)

The following symbols also are allowed in filenames:

\$ & # % ' ( ) -  
@ ^ { } ~ ` !

**Warning:** Do not include more than eight characters in a filename. If you do, MS-DOS truncates the filename to the first eight characters and accepts it. This can cause problems. For example, MS-DOS truncates both Johnfile1 and Johnfile2 to Johnfile. Because two files (that are in the same directory) cannot have the same name, MS-DOS overwrites the old file with the new file.

**Extensions.** You can use an "extension" to provide additional information on a file. Extensions are always preceded by a period (.) and can be from 1-3 characters long.

Using extensions such as .new, .irs, and .pay, you can distinguish files that have the same name or can divide files into categories.

You can also use an extension to indicate the file type. For example, you may wish to use some of the following:

.bas	for	BASIC programs
.txt	for	ASCII text
.dat	for	Data files
.obj	for	Object code
.rel	for	Relocatable code
.src	for	Source code

If you add the extension `.dat` to the inventory name, the filespec becomes

`invntory.dat`

**Once you include an extension in a filename, you must use it whenever you specify the file.**

**Warning:** Again, do not include more than three characters. If you do, MS-DOS truncates the extension to the first three characters and accepts it.

**Examples of Filenames.** Some legal names are:

`rawdata2`  
`REPORTS`  
`X.x`  
`project%.txt`  
`PROG1.bas`  
`SAMFILE`  
`2AR.dat`

Some illegal filenames are:

<code>max*min</code>	(because * isn't a legal character for names)
<code>.DATA</code>	(because the period can be used only to separate the filename and extension)
<code>open orders</code>	(because a name can't contain a space)

**Wild Cards.** MS-DOS lets you use these shorthand notations in filenames and extensions:

- ? The question mark indicates that any character can occupy that position.
- \* The asterisk indicates that any character can occupy that position or the remaining positions in the filename or extension.

Suppose you specify this filename:

`test?run.exe`

MS-DOS finds all files (in one directory) that begin with "test," have any character next, followed by "run," with the extension .exe. Here are some files MS-DOS might find:

**Test1run.exe**  
**Test3run.exe**  
**Test4run.exe**

If you specify this filename:

**test\*.exe**

MS-DOS finds all files (in one directory) that begin with "test" and have the extension .exe. In addition to the files listed above, MS-DOS might find others, including

**Test.exe**  
**Testall.exe**  
**Test1.exe**

If you specify this filename:

**oldfile.\***

MS-DOS finds all files named Oldfile (in one directory), regardless of their extensions. Here are examples of some files it might find:

**Oldfile.bas**  
**Oldfile.exe**  
**Oldfile.txt**

## **Pathnames**

Whenever you want to access a file, you must tell MS-DOS where to find it. You provide the information in the form of a "pathname," a list of names from the root directory down to the file you want to access. Each pathname can have up to 63 characters.

Suppose, for example, that you want to access the Radio file on the disk in Drive B. Tell MS-DOS the path to follow from the root directory to the file, separating the "stops along the way" with backslashes, as follows:

**B:\ADS\radio**



MS-DOS reads the pathname, from left to right, to determine that the file you want is on Drive B, in the ADS directory, and that its name is Radio.

MS-DOS's multi-level organization and pathname convention help you access files quickly. It also lets you give two files the same filename, as long as you store them in separate directories so that they have different pathnames.

**Note:** Under some circumstances, you can take a "short-cut" to a file or directory. See "Current Directory."

## Device Names

Each input/output device supported by the system has a unique name. The device names are as follows:

- AUX (auxiliary) refers to whatever device you set up as the auxiliary device. Normally, it is the RS-232C serial port.
- CON (console) refers to the screen or keyboard.
- PRN (print) refers to the printer.
- LST (list) refers to the printer.

Never use a device name as a filename.

## Directories

On MS-DOS, directories, which are collections of files, are nonetheless files themselves. They are processed by the same input/output functions used with regular files.

## Using Directories

To understand how directories work, assume that the disk in Drive B is freshly formatted so that it has only a root directory. You can use the editor, EDLIN, to create the test file File1.tst Drive B. To do so, type

```
EDLIN B:\file1.tst ENTER
```

EDLIN displays

New file

\*

The asterisk indicates that EDLIN is ready for you to enter a command. To enter the insert mode, so that you can enter text lines into the file, type

I (ENTER)

EDLIN displays the line number followed by a colon and asterisk. It places each line into the text file until you type (F6) (ENTER) or (CTRL) (Z) (ENTER) to end the file.

Create this file:

1:\*This is my test file. (ENTER)

2:\*I'm using it to show the use of directories. (ENTER)

3:\*(F6) (ENTER)

After you press (F6) (ENTER), EDLIN displays the asterisk to indicate it's ready for another command. To exit EDLIN and return to MS-DOS, type

E (ENTER)

MS-DOS displays the system prompt again.

If you use the DIR command, which lists the files in a directory, it now indicates the existence of the new file:

DIR B:\ (ENTER)

Volume in drive B has no label

Directory of B:\

FILE1	TST	70	8-24-83	9:07a
-------	-----	----	---------	-------

1 File(s)	nnnnnn bytes free
-----------	-------------------

You can use the TYPE command to display the text stored in the file:

TYPE B:\file1.tst (ENTER)

This is my test file.

I'm using it to show the use of directories.

Suppose you use EDLIN to create two more text files.

EDLIN B:\file2.tst (ENTER)

New file

\*I (ENTER)

1:\*This is my second file (ENTER)

2:\*I'm using it to show the use of directories

3:\*(F6) (ENTER)

\*E (ENTER)

EDLIN B:\file3.tst (ENTER)

New file

\*I (ENTER)

1:\*This is my third file (ENTER)

2:\*I'm using it to show the use of directories

3:\*(F6) (ENTER)

\*E (ENTER)

Now if you use DIR, it shows three file names:

DIR B:\ (ENTER)

Volume in drive B has no label

Directory of B:\

FILE1	TST	70	8-24-83	9:07a
FILE2	TST	70	8-24-83	9:09a
FILE3	TST	69	8-24-83	9:09a

3 File(s)

nnnnnn bytes free

## Creating Directories

To create a directory on the system, use the MKDIR command. Suppose that you want to create in the Drive B root directory a new directory called MYDIR. Type

MKDIR B:\MYDIR (ENTER)

MS-DOS automatically makes MYDIR a subdirectory of the Drive B root directory. You can check it by using DIR:

DIR B:\ (ENTER)

Volume in drive B has no label

Directory of B:\

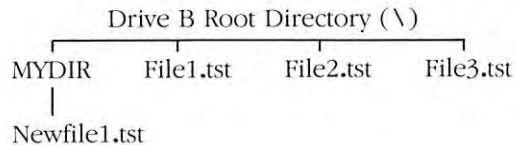
FILE1	TST	70	8-24-83	9:07a
FILE2	TST	70	8-24-83	9:09a
FILE3	TST	69	8-24-83	9:09a
MYDIR	<DIR>		8-24-83	9:09a

4 File(s) nnnnnn bytes free

Now suppose you want to copy File1.tst to the new directory. Use the COPY command, as follows:

COPY B:\file1.tst B:\MYDIR\newfile1.tst (ENTER)

Here's what the structure looks like now:



You can use DIR to see the name of the file in the new directory:

DIR B:\MYDIR (ENTER)

Volume in drive B has no label

Directory of B:\

.	<DIR>		8-24-83	9:09a
..	<DIR>		8-24-83	9:09a
NEWFILE1	TST	70	8-24-83	9:07a

3 File(s) nnnnnn bytes free

It's possible to use MKDIR to create a subdirectory of MYDIR, a subdirectory of the new directory, and so on. Your only limit is disk space availability.

**Note:** See “Anonymous Directory Names” for an explanation of the . and .. symbols.



## Deleting Directories

When deleting a directory, you must first remove all the files it contains. If you delete a directory while it still contains files, MS-DOS has no way — no path — to access those files, or to return their storage to the storage pool.

To delete a directory, follow these steps:

1. Use the DIR command to see what files are in the directory.
2. Use the COPY command to copy any files you may need to another directory.
3. Use the ERASE command to remove all files from the directory.
4. Use the RMDIR command to remove (delete) the directory.



## The Current Drive

With MS-DOS, the disk you are using at any given time is your “current disk” or “current drive.” Immediately after startup, MS-DOS places you in the root directory of the Drive A disk. Thus, Drive A is your current drive.

**Using the Current Drive.** Knowing about the current drive enables you to take “shortcuts” when specifying pathnames. It also lets MS-DOS find what you want more quickly.

- When specifying a file or directory that is not in the current drive, you must specify the entire pathname.
- When specifying a file or directory that **is** in the current drive, however, you need not include the drive specification in the pathname.



For example, suppose you are in A:\. If you want to access B:\PAYROLL, specify the drive, as well as the directory name:

B:\PAYROLL

If you want to access A:\USER, specify only the directory name:

\USER

**Changing the Current Drive.** MS-DOS lets you quickly change the current drive. To do so, enter the drive specification at the system prompt. For example, at A>, type

B: (ENTER)

MS-DOS displays a new system prompt, B>, to indicate you are now in Drive B.

## The Current Directory

The directory you are using at any given time is your “current directory.” Therefore, immediately after startup, the root directory of Drive A is your current directory.

**Using the Current Directory.** Knowing about the current directory lets you take even more shortcuts when specifying pathnames.

- When specifying a file or directory that is higher than the current directory, and on the same disk, you must give a complete pathname (minus the drive specification).
- When specifying a file or directory that is within or lower than the current directory, and on the same disk, you may begin the pathname immediately below your current directory. The rest of the pathname is implied.

For example, suppose you are in the directory A:\USER\MIKE. If you want to access the \USER directory on the same disk, give the complete pathname (minus the drive specification):

\USER

If you want to access the \USER\MIKE\MEMOS directory on the same disk, begin the pathname below the current directory, as follows:

### **MEMOS**

Notice that you should **not** precede this pathname with a backslash. That is because the pathname \MEMOS specifies a directory that does not exist — one that would be an immediate subdirectory of the root directory.

If you are still in \USER\MIKE, you can specify a file in that directory by giving only the filename. For example, if you type

```
job1
```

the complete pathname \USER\MIKE\job1 is implied.

Again, if you are still in \USER\MIKE and you type

### **MEMOS\mgrs**

the complete pathname \USER\MIKE\MEMOS\mgrs is implied.

When you enter commands, MS-DOS automatically expands pathnames as needed. For example, if you type

```
COPY job1 MEMOS\newjob1 (ENTER)
```

MS-DOS interprets this as

```
COPY \USER\MIKE\job1  
  \USER\MIKE\MEMOS\newjob1 (ENTER)
```

**Changing the Current Directory.** Using the CHDIR command, you can make any directory on the current drive your current directory. To do this, enter the CHDIR command followed by the pathname of the new current directory.

For example, to change the current directory from A:\ to A:\USER\MIKE type

```
CHDIR \USER\MIKE (ENTER)
```

To change to a directory on another drive, simply change drives before using the CHDIR command. For example, to change your current directory to B:\PAYROLL, type

B: (ENTER)

MS-DOS automatically puts you in the root directory of Drive B. To change to \PAYROLL, type either of the following:

CHDIR PAYROLL (ENTER)

CHDIR \PAYROLL (ENTER)

## Home Directories

Suppose your current directory is B:\PAYROLL and then you change it back to A:\USER\MIKE. MS-DOS “remembers” that you were using B:\PAYROLL. For your convenience, it makes \PAYROLL the “home directory” of Drive B, until you change to another Drive B directory.

**Using Home Directories.** This means that whenever you specify a file or directory within or below B:\PAYROLL, you need not include the directory name PAYROLL.

For example, suppose you are in Drive A. If you type

TYPE B:allstaff (ENTER)

MS-DOS assumes you mean B:\PAYROLL\allstaff.

Similarly, if you type

DIR B: (ENTER)

MS-DOS assumes you mean DIR B:\PAYROLL; it shows a directory listing of all the files in the PAYROLL directory. It does **not** show a directory listing of the entire disk. For this, you must type

DIR B:\ (ENTER)



Knowing about home directories is particularly helpful when you are copying files. Suppose you must copy several files from A:\USER\MIKE to B:\PAYROLL. Make B:\PAYROLL the home directory of Drive B; then make A:\USER\MIKE your current directory. Now you can copy each file, specifying only the filenames.

If you type

```
COPY job1 newjob1 (ENTER)
```

MS-DOS interprets this as

```
COPY A:\USER\MIKE\job1  
B:\PAYROLL\newjob1 (ENTER)
```

## **Anonymous Directory Names**

Sometimes you may need to refer to your current directory, or a higher-level directory, although you may not know the full pathname. Or you may want merely to save typing time.

In either case, MS-DOS makes special “name substitutes” available:

- The name “.” refers to the current directory
- The name “..” refers to the “parent” of the current directory (the next highest-level directory in the path)
- The name “. \ .” refers to the directory two levels up

You can use the names in place of pathnames and/or as the first names in pathnames. Some examples:

```
DIR . (ENTER)
```

lists filenames in the current directory.

```
DIR .. (ENTER)
```

lists names in the current directory’s parent directory.

```
ERASE . \ taxes84 (ENTER)
```

deletes the Taxes84 file from the current directory’s parent directory.

The substitute names may refer to either the current directory or the home directory of another drive, depending upon whether you include a drive specification. For example,

**ERASE B:.\taxes84**

erases the Taxes84 file from the parent directory of the home directory of Drive B.



# Chapter 2

---

## Introduction to Commands

Commands are a way of instructing the computer to perform useful tasks. Here are a few of the things MS-DOS commands let you do:

- Format disks to accept MS-DOS files (get them ready for information storage)
- Copy the MS-DOS system (all system files) to another disk
- Copy all information (system and data files) from one disk to another and (optionally) compare the disks
- Create, copy, display, rename, and remove files
- Execute system programs, such as EDLIN and DEBUG, as well as your own programs
- Create, list, and remove directories
- Enter the date, time, and remarks
- Set various printer and screen options
- Request MS-DOS to pause

## Types of MS-DOS Commands

The two types of MS-DOS commands are:

- Internal
- External

### Internal Commands

These are the simplest, most commonly used commands. When you do a directory listing on your MS-DOS disk, you cannot see these commands. When you enter them, they execute immediately. The following internal commands are described in Section II:

BREAK	EXIT	REN
CHDIR (CD)	FOR	RMDIR (RD)
CLS	GOTO	SET
COPY	IF	SHIFT
CTTY	MKDIR (MD)	TIME
DATE	PATH	TYPE
DIR	PAUSE	VER
ECHO	PROMPT	VERIFY
ERASE (DEL)	REM	VOL

## External Commands

These commands reside on disks as program files. Therefore, MS-DOS must read them from disk before it can execute them. If the disk containing the command is not in the drive, the system cannot find and execute the command.

**Note:** MS-DOS must also know in which directory or drive to search for external commands, or it cannot execute them. (See the PATH command.)

Any filename that has an extension of .com, .exe, or .bat is considered an external command. For example, programs such as FORMAT.COM and DISKCOPY.COM are external commands. You may create external commands and add them to the system. Programs that you create with most languages (including assembly language) are .exe (executable) files.

When you enter an external command, do not include its filename extension. The following external commands are described in Section II:

BACKUP	HFORMAT
CHKDSK	MORE
COMPDUPE	PRINT
DISKCOPY	RECOVER
EXE2BIN	RESTORE
FIND	SORT
FORMAT	SYS

## Command Parameters

In Chapter 1, you learned about the *pathname*. This is only one of several “parameters” you may use to customize your commands.

Some parameters are required; others are optional. If you omit an optional parameter, the system may automatically provide a “default” value. For example, as you should recall from Chapter 1, the system defaults to the current drive whenever you omit the *drive* part of a *pathname*.

Required and optional parameters and default values — other than those discussed in Chapter 1 — vary according to the particular command. (See the particular command in Section II for more information.)

Here are examples of other parameters:

- filespec* specifies the file in the format *drive:filename.ext*, such as B:Text.txt. All parts of the *filespec* are optional.
- ON OFF are “arguments,” parameters from between which you choose. Arguments provide more information to a command. For example, BREAK ON turns on the **CTRL** **C** check.
- /W is an example of a “switch,” a parameter that controls a command. For example when you use /W with the DIR command, MS-DOS shows the “wide display” version of the directory. Precede all switches with a slash.

## Information Common to All MS-DOS Commands

The following information applies to all MS-DOS commands:

- The system prompt is the current drive designation followed by a greater-than sign. For example, **A>** tells you that Drive A is your current drive and that MS-DOS is ready to accept a command.
- Commands are usually followed by one or more parameters.
- Parameters, like commands, may be entered in upper-case or lower-case, or a combination.
- Commands and parameters must be separated by delimiters. The space and comma are the easiest to use. Examples:

```
DEL MYFILE.OLD NEWFILE.TXT  
rename,afile bfile
```

You may also use the semicolon (;), equal sign (=), or **(TAB)** as delimiters. This manual uses a space.

- Wild cards and device names (such as PRN and CON) are not allowed in command names.
- *source pathname* specifies the disk from which you transfer information. *target pathname* specifies the disk to which you transfer information. Some error messages refer to the “destination.” This is the same as the target.
- When typing commands, you may use the MS-DOS editing and function keys. (See Chapter 4.)
- Commands execute only after you press **(ENTER)**.
- By pressing **(CTRL) (C)**, you can abort a command when it is running.

- When a command produces much screen output, the display automatically scrolls to the next screen. To suspend the display, press **CTRL S** or **HOLD**. To resume, press **CTRL Q** or **HOLD**.

## Input and Output

MS-DOS assumes that command “input” comes from the keyboard and “output” goes to the screen. However, you can redirect input so it comes from a file and output so it goes to a file or a line printer.

In addition, you can create “pipes” that let one command’s output become another’s input.

### Redirecting Input

**SORT ENTER**

sorts the information you enter from the keyboard and displays the sorted output. To end keyboard input, press **CTRL Z ENTER**.

To redirect input so it comes from a file, use a less-than sign (<) in your command. For example,

**SORT <names ENTER**

sorts the information in the file Names and displays the sorted output.

### Redirecting Output

**DIR ENTER**

displays a list of all the directories that are on the current disk.

To redirect output so it goes to a file, use a greater-than sign (>) in your command. For example,

**DIR >myfiles ENTER**

sends the same listing to the file Myfiles in the current directory. If Myfiles does not already exist, MS-DOS creates it and stores the listing in it. If Myfiles does exist,

MS-DOS overwrites the old information with the new information.

Rather than replace an entire file, you may want to “append” your output to the end of it. To do this, use two greater-than signs. For example,

**DIR >>myfiles **(ENTER)****

appends your directory listing to the file Myfiles in the current directory. If Myfiles does not exist, MS-DOS creates it.

Here is an example in which both the input and output are redirected:

**SORT <names >list 1 **(ENTER)****

sorts the information in the file Names and sends the sorted output to the file List1 in the current directory.

## **Filters**

A “filter” is a command that transforms input in some way (filters it) before outputting it — usually to the screen or a file.


The MS-DOS filters are FIND, MORE, and SORT. Their functions are as follows:

- FIND — Searches for a constant string of text in a file
- MORE — Takes standard screen output and displays it, one screen at a time
- SORT — Sorts text from A-Z or from Z-A (reverse sort)

By combining commands and filters, you can replace several commands with a few filters. The section below tells you how.



## Command Piping



“Piping” lets you give more than one command at a time to the system. It does this by making one command’s output another command’s input.


To pipe commands, simply divide them with the “pipe separator,” the vertical bar (|). All output generated by the command to the left of the bar becomes input for the command to the right.

For example, suppose you have a program that produces output in columns. You may want to sort this output. This command:

**DIR | SORT**

displays an alphabetically sorted listing of the current directory. This command:


**DIR | SORT >direc.fil**

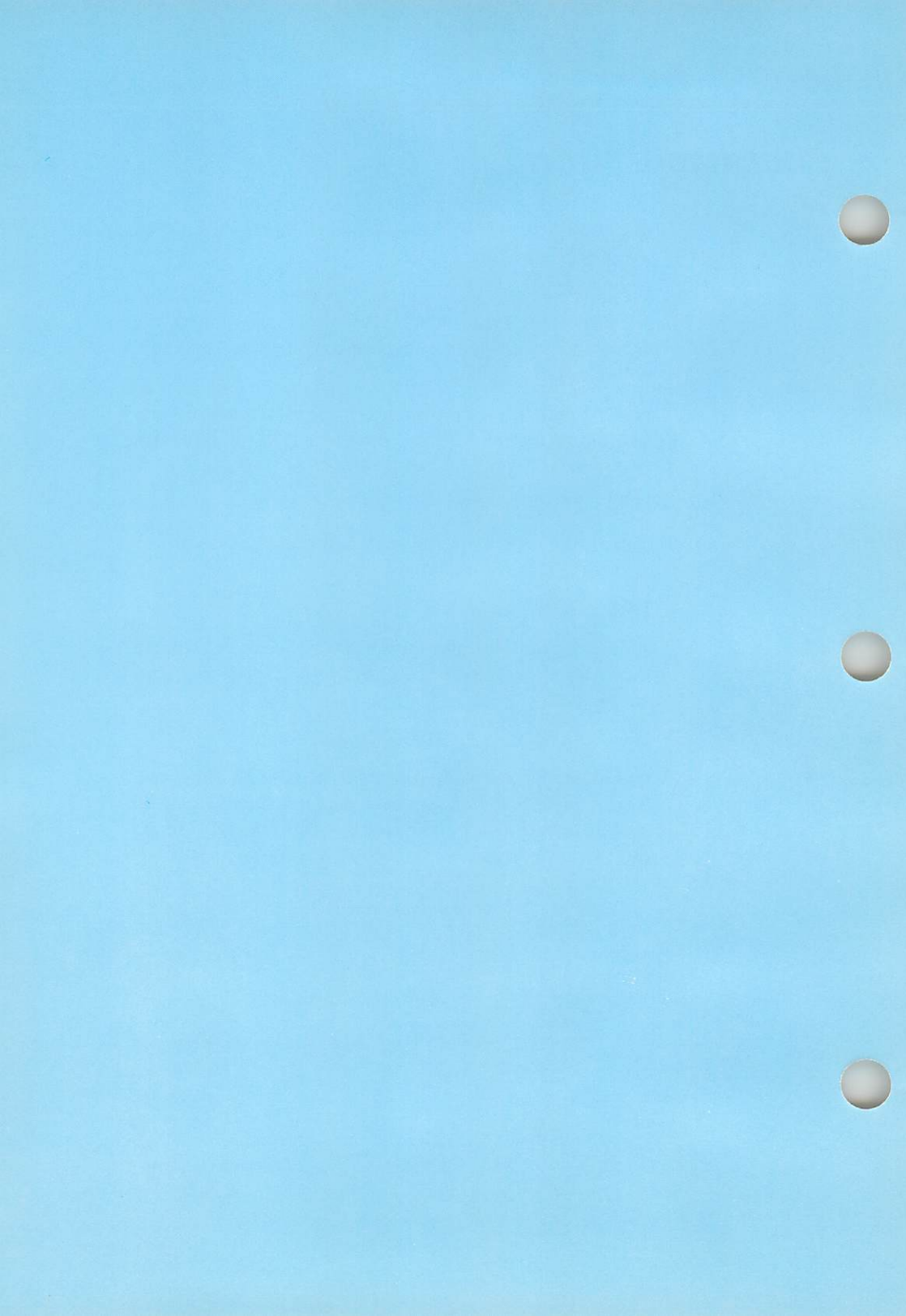


sends the same listing to the file Direc.fil in the current directory. If the file doesn’t exist, MS-DOS creates it. This command:

**DIR B: | SORT >B:direc.fil**

makes an alphabetically sorted listing of the home directory of Drive B. It then sends the listing to the file Direc.fil in the home directory on Drive B. Again, MS-DOS creates the file, if necessary.





# Chapter 3

---

## Batch Files: Executing Several Commands

Some tasks require you to enter two or more commands. For example, when preparing a disk for information storage, you must format the disk so you can write to it (FORMAT command). In addition, it is a good habit to immediately check the directory of the new disk for errors (CHKDSK command).

### The Batch File

With MS-DOS, you can put such a command sequence into a special file called a "batch file." Then you can execute the entire sequence simply by entering the name of the batch file.

When you create a batch file, you may specify the complete pathname, a filespec, or a filename. In any case, you **must give the file the extension .bat**. When you execute the file, however, enter the filename without the extension.

Below is a description of how to use the COPY command to create a batch file. (You may also use EDLIN.)

### Creating a Batch File

To create a file to perform FORMAT and CHKDSK as discussed above, type this at the system prompt:

```
COPY con prepdisk.bat (ENTER)
```

This command tells MS-DOS to copy the information entered from the keyboard (console) into a batch file called Prepdisk.bat. Because you do not specify otherwise, MS-DOS creates the file in the current directory.

MS-DOS again displays the system prompt. Now you can type the commands to be included in the file. Type

```
REM This is a file to prepare and check new disks  
(ENTER)
```

```
REM It is called Prepdisk.bat (ENTER)
```

```
FORMAT B: (ENTER)
PAUSE (ENTER)
CHKDSK B: (ENTER)
```

After entering the last command (CHKDSK B:), you are ready to save the commands in the file. To do this, simply press **(F6) (ENTER)** or **(CTRL) (Z) (ENTER)** at the next system prompt. MS-DOS displays ^Z to indicate you used the control character. Then it displays

1 File(s) copied

At this time, you may want to type DIR **(ENTER)** to verify that the file is created.

## REM and PAUSE

Two of the commands above — REM and PAUSE — are available only for use in batch files.

The REM command lets you include remarks in your batch file, without those remarks being executed as commands. The remarks in the Prepdisk.bat file are to remind you what the file does. Should you forget, you can type

```
TYPE prepdisk.bat (ENTER)
```

and MS-DOS displays the contents of the file, including the remarks.

The PAUSE command lets you control how much of the file you want to execute. When it reaches a PAUSE command, MS-DOS pauses and displays the message:

Strike a key when ready . . .

You may continue execution by pressing **(SPACEBAR)** or abort it by pressing **(CTRL) (C)**.

(The REM and PAUSE commands are explained further in Section II.)

## Executing a Batch File

To execute your batch file, you must be in the directory that contains the file. (If you are not in it, execute a CHDIR

command as described in Section II.) Then enter the filename, without the extension. For example, to execute `Prepdisk.bat`, type this at the system prompt:

```
prepdisk (ENTER)
```

Execution proceeds just as if you entered each command line at the keyboard.

## A Word of Advice about Batch Files

As you are getting acquainted with MS-DOS, we strongly recommend that you create all batch files in the root directory of your MS-DOS disk.

Later, when you better understand MS-DOS, you may want to create batch files elsewhere. You can do this by specifying a complete pathname for the file. If you do so, however, keep in mind the following:

- When you execute a batch file, you must be in the directory that contains the file. (That directory becomes your current directory and that drive becomes your current drive.)
- MS-DOS searches only the current directory for external commands.

Therefore, if you use any external commands in your batch file, you may need to do one of the following:

- Use the `PATH` command to tell MS-DOS to search for external commands in the directory that contains the commands.
- Use the `COPY` command to move your external commands to the directory that contains your batch file.
- Use the `CHDIR` command within your batch file to move from one directory to another as necessary.

## Summary of the Batch File Process

1. Create the file by typing  
`COPY con pathname.bat` **(ENTER)**
2. Enter the command lines.
3. Save the file by typing either of the following:  
**(F6)** **(ENTER)**  
**(CTRL)** **(Z)** **(ENTER)**
4. When you are in the directory that contains the file, execute the file by typing  
`filename` **(ENTER)**

## The Autoexec.bat File

An Autoexec.bat file lets you execute programs or commands automatically when you start MS-DOS. This is useful when you want to run a specific package under MS-DOS or when you want MS-DOS to execute a batch file automatically each time you start the system. Using an Autoexec.bat file, you can avoid loading two disks to do either of these tasks.

When you start MS-DOS, the command processor searches the MS-DOS disk for a file called Autoexec.bat. If MS-DOS finds the file, it immediately executes it, bypassing the date and time prompts.

If MS-DOS does not find an Autoexec.bat file, it displays the date and time prompts as usual.

## Creating an Autoexec.bat File

The Autoexec.bat file must be created in the root directory of your MS-DOS disk. (You must be in that directory when you create the file.) You create it the same way as any other batch file, except that you give it the name Autoexec.bat.

Remember, if you use an Autoexec.bat file, MS-DOS prompts you for a current date and time **only** if you

include the DATE and TIME commands in the file. Because MS-DOS uses this information to keep the directory current, you should always include these commands.

Suppose, for example, you want to load BASIC and run a BASIC program called MENU each time you start MS-DOS. To create and save a file that does this, type

```
COPY con Autoexec.bat (ENTER)
DATE (ENTER)
TIME (ENTER)
BASIC MENU (ENTER)
(F6) (ENTER)
```

The Autoexec.bat file is a great time-saver. You can use it to execute many programs and any series of commands.

## Batch Files with Replaceable Parameters

When creating a batch file, you may want to include replaceable (dummy) parameters. In this way, you can use different sets of data when you run the file.

For example, the Prepdisk.bat file shown earlier is changed here to include the dummy parameter %1.

```
COPY con prepdisk.bat (ENTER)
REM This is a file to prepare and check new disks in
  a specified drive (ENTER)
REM It is called Prepdisk.bat (ENTER)
FORMAT %1 (ENTER)
CHKDSK %1 (ENTER)
(F6) (ENTER)
```

To execute this file to format the disk in Drive B, enter the batch file's name, followed by the drive specification to replace %1, as shown here:

```
prepdisk B: (ENTER)
```

The next two sections tell more about creating and executing batch files that have dummy parameters.

## **Creating a Batch File with Replaceable Parameters**

The dummy parameters are %0 through %9. MS-DOS always replaces %0 with the filename (without extension) of the batch file, unless you use the SHIFT command. It replaces the other dummy parameters, sequentially, with the parameters you specify when you execute the batch file. (See the SHIFT command in Section II if you wish to specify more than 10 dummy parameters.)

These parameters may be pathnames, drive specifications, numeric values, or almost anything else you can think of.

For example, if you type

```
COPY con myfile.bat (ENTER)
COPY %1.mac %2.mac (ENTER)
TYPE %2.mac (ENTER)
TYPE %0.bat (ENTER)
(F6) (ENTER)
```

MS-DOS creates the batch file Myfile.bat in the current directory and copies the next three lines into that file.

When you execute the file, the parameters %1 and %2 are replaced sequentially by the pathnames you supply.

## **Executing a Batch File with Replaceable Parameters**

To execute a batch file that has replaceable parameters, enter the batch filespec (without its extension), followed by the parameters to replace the dummy parameters.

For example, to execute Myfile.bat, type

```
myfile A:prog1 B:prog2 (ENTER)
```

Myfile is substituted for %0, A:Prog1 for %1, and B:Prog2 for %2.

The result is the same as if you had entered each of the commands with its parameters, as follows:

```
COPY A:prog1.mac B:prog2.mac (ENTER)
```



TYPE B:prog2.mac (ENTER)

TYPE myfile.bat (ENTER)

The COPY command copies the contents of one file to another file. The TYPE command displays the contents of a file.

The following table illustrates how MS-DOS replaces each of the above dummy parameters:

Dummy Parameter	%0	%1	%2
Parameter	Myfile	Prog1	Prog2
Parameter & Ext.	Myfile.bat	Prog1.mac	Prog2.mac

## Sample Use of Replaceable Parameters

Replaceable parameters can be very handy in day-to-day use with application programs. For example, suppose you have a program that creates a client mailing list and saves the list to a file called Mail.dat in the root directory of your system disk.

The information in Mail.dat might look like this:

Tom	Cleo	2	8th St.	Lincoln	NE	68502
Ann	King	1	9th Av.	Rapid City	SD	57001
Sam	Beck	4	6th St.	Ft. Worth	TX	76133

As you can see, the information is in random order. Using a batch file with replaceable parameters, however, you can organize the information in any way that is most convenient at any given time. One time, you might organize it according to zip code; another time, according to last name; and so on. To create such a batch file, type the following:

COPY con mailsort.bat (ENTER)

COPY mail.dat %1.dat (ENTER)

TYPE %1.dat (ENTER)

SORT %2 <%1.dat >%3.dat (ENTER)

TYPE %3.dat (ENTER)

(F6) (ENTER)

Suppose you execute the batch file by typing

mailsort newmail / + 15 sort (ENTER)

The result is the same as if you had entered each command with its parameters, as follows:

```
COPY mail.dat newmail.dat (ENTER)
TYPE newmail.dat (ENTER)
SORT /+ 15 <newmail.dat >sort.dat (ENTER)
TYPE sort.dat (ENTER)
```

MS-DOS copies the information from Mail.dat into Newmail.dat and then displays the contents of Newmail.dat. It then sorts the contents, starting at Column 15 (it sorts alphabetically, according to last name). Finally, it copies the sorted data to the file Sort.dat and displays it.

**Note:** By piping commands, you can reduce your batch file to two command lines.

```
COPY mail.dat %1.dat | TYPE %1.dat | SORT /+ 15
  >%3.dat (ENTER)
TYPE %3.dat (ENTER)
```

## Reminders about Batch Files

The following list summarizes information you should know before you execute a batch process with MS-DOS:

- Do not enter the filename Batch (unless the name of the file you want to execute is Batch.bat).
- To execute a batch file, enter the filename without the extension.
- The commands in the file named *filename*.bat are executed.
- If you press (CTRL) (C) while in batch mode, this prompt appears:

Terminate batch job (Y/N)?

If you press (Y), MS-DOS ignores the rest of the commands in the batch file. The system prompt is displayed.

If you press (N), the current command ends. Batch processing continues with the next command in the file.

- If you remove the disk that contains the batch file being executed, MS-DOS prompts you to insert it again so that the next command can be read.
- Immediately upon executing one batch file, you may call another. To do so, simply put the second file's name (without its extension) as the last command in the first file.
- In a batch file, you may want to refer to a file whose name contains a percent sign. For example, you may want to include a command that copies the file `Abc%.exe`. As you know, the batch file normally interprets the percent sign as a replaceable parameter. To indicate that this is **not** the case, you must include a second percent sign. The batch file command, then, might be this: `COPY A:\USER\abc%%.exe B:\USER.`



## Chapter 4

---

# Correcting Errors and Editing Commands

If you make a mistake in typing a command line and have not pressed **(ENTER)**, you can use one of the following keys to correct the mistake.

- **(BACKSPACE)** — backs up the cursor, erasing the last character
- **(F8)** — voids the command line and lets you start over

Suppose, however, that you **have** pressed **(ENTER)**. What then? With many operating systems, you must retype the entire command — but not with MS-DOS.

## The Template

When you press **(ENTER)** after typing a command, MS-DOS sends the command to the command processor (COMMAND.COM) for execution. In this way, it is like other operating systems.

Unlike other systems, however, MS-DOS also sends the command to a special storage area called the “template.” The template stores only one command at a time — the last command entered.

You can then recall the command from the template and can do any of the following:

- Repeat the command instantly by pressing **(F3) (ENTER)**.
- Edit the command and retry it, without retyping the entire line.
- Edit the command line into a similar command so that you don't have to type the entire new command.

## The Editing Keys

The table below summarizes the MS-DOS editing keys. The next section gives an example of how to use the keys. At the end of the chapter is a summary of control character keys.

**Note:** You can also use the editing keys to edit your text files. How to do so is described in Section III, "EDLIN."

Function	Key(s)	Description
Copy <i>char</i>	<b>(→)</b>	Copies one character (from the template to the command line) and displays it.
Delete <i>char</i>	<b>(DELETE)</b>	Deletes a character from the template. Therefore, the character is skipped (is not copied to the command line).
Copy to <i>char</i>	<b>(F2)char</b>	Copies all characters up to the specified character and displays them.
Delete to <i>char</i>	<b>(F4)char</b>	Deletes all characters up to the specified character from the template. Therefore, the characters are skipped (are not copied to the command line).
Copy all	<b>(F3)</b>	Copies all remaining characters and displays the entire command line.
Insert	<b>(INSERT)</b>	Enters the insert mode. ( <b>(F3)</b> ends the insert mode.)
Replace template	<b>(F5)</b>	Makes the new line the new template, but does not send it to the requesting program. (Accepts the line for more editing.)
Void line	<b>(F8)</b> or <b>(CTRL) (X)</b>	voids the current input. Leaves the template unchanged. Type and enter a new line, or press <b>(ENTER)</b> to display the system prompt.

Function	Key(s)	Description
Enter line	<b>ENTER</b>	Makes the new line the new template and sends it to the requesting program.
End-of-file	<b>F6</b> or <b>CTRL Z</b>	Puts an end-of-file character in the new template.

## Sample Uses of Editing Keys

Suppose you have two files, Prog.com and Prog.asm, in the root directory of your MS-DOS disk. If you type the following command:

**DIR prog.com** **ENTER**

MS-DOS displays information about the file Prog.com. At the same time, it saves the command line (DIR prog.com) in the template.

Press **F3** to display the command line. (The underscore represents the blinking cursor.)

**F3** DIR prog.com\_

Execute the command again by pressing **ENTER**.

To display information about the file Prog.asm, edit the command line DIR Prog.com. Type:

**F2**C

MS-DOS displays all characters of the command line up to but not including the letter c, as shown here.

DIR prog.\_

Now type

asm\_

The letters asm replace the letters com, resulting in

DIR prog.asm\_

This new command line is now in the template. To execute it, press **ENTER**

Replace DIR with the TYPE command by typing

TYPE

The characters TYPE automatically replace the characters DIR and the space that followed.

Now type

**(INSERT)** **(SPACEBAR)** **(F3)**

Pressing **(INSERT)** **(SPACEBAR)** lets you insert a space. **(F3)** copies the rest of the template. The result is

TYPE prog.asm.

To execute the new command, press **(ENTER)**.

Suppose you had misspelled TYPE as BYTE, but had **not** entered the command. (Set up this situation quickly by typing BYTE and pressing **(F3)** again.) Your screen displays

BYTE prog.asm.

You can still use some of what you have typed. To do so, press **(F5)**. The symbol @ appears at the end of the line. This indicates that MS-DOS has put the new line in the template, although it has not sent the line to be executed.

You can now edit the line BYTE prog.asm so it becomes TYPE program.asm To do so, type

T $\rightarrow$ P**(F3)**

The  $\rightarrow$  key copies a single character (Y) from the template to the command line. The resulting command line is the command you want:

TYPE prog.asm.

Here is another way to change BYTE prog.asm to TYPE prog.asm. Press **(F5)** again. Then press

**(DELETE)** **(DELETE)**  $\rightarrow$  **(INSERT)**YP**(F3)**

Pressing **(DELETE)** twice deletes the first two characters (B and Y).  $\rightarrow$  copies the third character (T). **(INSERT)** enters the insert mode, so you can insert the letters Y and P. Then



**(F3)** copies the rest of the template.

Notice that **(DELETE)** does not affect the command line. It affects the template by deleting the first character. Similarly, **(F4)** deletes characters in the template, up to but not including a given character.

## The Control Character Keys

A control character key is one that affects a command line. You have already learned about **(CTRL) (C)**. Others are described below.

Remember that when you type a control character you must press the second key while holding down the first.

Key(s)	Function
<b>(CTRL) (C)</b>	Stops execution of a command.
<b>(SHIFT) (PRINT)</b>	Sends to the line printer everything currently displayed on the screen.
<b>(PRINT)</b> or <b>(CTRL) (P)</b>	Sends all output to the line printer, as well as to the screen. Press again to stop the function.
<b>(CTRL) (N)</b>	Toggles echoing of output to the line printer.
<b>(CTRL) (H)</b> or <b>(BACKSPACE)</b>	Removes the last character from the command line and erases the character from the display.
<b>(CTRL) (J)</b>	Inserts a physical end-of-line, but does not empty the command line. Use <b>(CTRL) (J)</b> to extend the current logical line beyond the limits of the screen.
<b>(HOLD)</b> or <b>(CTRL) (S)</b>	Suspends the screen. Press <b>(HOLD)</b> or <b>(CTRL) (Q)</b> to resume scrolling.

<b>Key(s)</b>	<b>Function</b>
<b>F8</b> or <b>CTRL X</b>	<p>Voids the current line and empties the command line. It then outputs a backslash (\), carriage return, and line feed. The template used by the special editing commands is not affected. Although the system prompt is not displayed, the system is ready for a command.</p>







## Section II

---

### Commands Reference

This section contains an alphabetical listing of all MS-DOS commands, including regular and batch commands.

The commands are:

BACKUP	EXIT	RECOVER
BREAK	FIND	REM
CHDIR	FOR	RENAME
CHKDSK	FORMAT	RESTORE
CLS	GOTO	RMDIR
COMPDUPE	HFORMAT	SET
COPY	IF	SHIFT
CTTY	MKDIR	SORT
DATE	MORE	SYS
DEL (ERASE)	PATH	TIME
DIR	PAUSE	TYPE
DISKCOPY	PRINT	VER
ECHO	PROMPT	VERIFY
		VOLUME

The listing also includes a discussion of the MS-DOS file comparison utility, **FC**.

### Entry Organization

Each entry begins with the command name and type (internal or external).

Next is the command "syntax." Use this as your guide to type in the command. (See the "Syntax Notation" section below.)

A brief description of the command's function follows the syntax. This includes any precautions you should take when using the command.

Next is the "parameters" section. This provides additional information about how to customize the command for your own purposes.

Sample uses and examples follow the parameters. For your convenience, some entries include additional remarks and error information.

## Syntax Notation

A command's syntax tells you what format to use when you type the command. For your convenience, the following notations are used in the command syntax and text referring to the commands:

### UPPER-CASE

indicates keywords (material that you must type). You may type the keywords in any combination of upper- and lower-case letters. MS-DOS interprets them as upper-case.

### **KEYBOARD CHARACTER**

indicates a key you press.

### *lower-case italics*

represent words, letters, characters, or values that you supply.

### [ ] (square brackets)

indicate optional parameters. Do not include the brackets when typing the command.

### . . . (ellipsis)

indicates that you may repeat a parameter as many times as you want.

Type all other punctuation exactly as shown in the syntax line.

## Synonymous Keywords

Some commands give you a choice of keywords. For example, you may type either DEL or ERASE when deleting files. Similarly, you may shorten MKDIR to MD. The command does the same thing, regardless of the keyword form.

## A Special Note to Hard Disk Users

Even though your system has only one **physical** floppy disk drive (Drive A), you can enter most commands as you would on a multi-floppy system. This is because your system includes a **logical** floppy drive (Drive B).

When you enter commands, think of Drive A and Drive B as referring to disks, rather than to drives.

Suppose you want to copy the file Myfile.dat from the current directory of one floppy disk to the home directory of another floppy disk. Think of the source disk as the Drive A disk and the target disk as the Drive B disk. At the system prompt, type

**COPY Myfile.dat B:Myfile.dat (ENTER)**

Because you specify Drive B (B:Myfile.dat) immediately after MS-DOS uses the Drive A disk (Myfile.dat), MS-DOS prompts you to insert the disk for Drive B. When it does so, remove the Drive A disk from Drive A and insert the Drive B disk. Press **(SPACEBAR)** to continue.

Remember, the system prompt represents the current drive. It does **not** represent the last disk used.

---

**BACKUP** [*pathname*] *drive* [/S] [/M] [/A]  
[/D:*mm/dd/yy*]

(HARD DISK ONLY) Copies (“backs up”) one or more files from a hard disk to floppy disks (“diskettes”).

To use files that are on backup diskettes, you must use the RESTORE command to copy them back to hard disk. Do not try to use them otherwise.

If, for some reason, you want to stop in the middle of backing up, press **CTRL C**.

**Note:** BACKUP works best if BUFFERS = 5 (or greater) in the CONFIG.SYS file. (See Appendix C.)

## Parameters:

*pathname* specifies the file you want to back up.

*drive* specifies the disk to receive the files.

/S causes BACKUP to copy all files in the specified directory and all directories below it.

/M causes BACKUP to copy only those files that have been modified since the last backup.

/A tells BACKUP to add the files (to be backed up) to the diskette already in the specified drive, rather than prompting you to insert a new diskette.

If you omit the /A parameter, BACKUP asks you to insert a diskette. Then, before backing up any hard disk files, it erases any existing files on the diskette. When the diskette is filled, BACKUP asks you to insert another. Be sure to label each diskette so you know the proper order of the backups.

/D:*mm-dd-yy* causes BACKUP to copy only those files created on or after the specified date.

## Sample Use:

Loss of information stored on hard disk, although not likely, can be disastrous — simply because of the amount



of information. Therefore, you should always keep and update floppy diskette copies of all hard disk information.

Suppose you want to back up all the files in the current directory. First, use the FORMAT command to prepare several diskettes for information storage. Then insert the first formatted diskette into Drive A and type

```
BACKUP *.* A: (ENTER)
```

Whenever it fills a diskette, BACKUP prompts you for another. Suppose that later you want to add the hard disk file Inventory.dat to the backup diskette. Assume the file is in the current directory; type:

```
BACKUP inventory.dat A: /A (ENTER)
```

**Note:** If you have several files to back up, you may want to create a batch file consisting of the necessary backup commands. Then you can do all the backups simply by running the batch file.

## Examples:

```
BACKUP *.bat A: (ENTER)
```

backs up all batch (.bat) files from the current directory to Drive A. This command erases any existing files on the diskette before copying files to it.

```
BACKUP C:STORE1\sales.dat A:/A (ENTER)
```

backs up the file Sales.dat that is in the STORE1 directory on Drive C to the diskette in Drive A, without erasing the diskette's existing files.

```
BACKUP *.* A:/M (ENTER)
```

backs up (to the diskette in Drive A) all files in the current directory that have been modified since the last backup.

```
BACKUP C:\ A:/S (ENTER)
```

backs up (to the diskette in Drive A) all files in all directories on Drive C.

## BREAK [ON|OFF]

Turns the **CTRL C** check on or off.

BREAK applies only when an application program is executed; it does not apply at the MS-DOS command level. It also does not apply to BASIC programs.

### Parameters:

Specifying BREAK ON tells MS-DOS to check for **CTRL C** from the keyboard whenever an application program makes **any** type of MS-DOS function call. Specifying BREAK OFF tells MS-DOS to check for a **CTRL C** only when a screen, keyboard, printer, or serial port function call is made.

### Sample Use:

Suppose you are about to run an application program that uses the **CTRL C** function key. Type

**BREAK OFF ENTER**

at the MS-DOS prompt. This turns off the MS-DOS **CTRL C** check. Now, whenever you press **CTRL C**, you affect your program instead of the operating system.

When you finish running your application program, and you are using MS-DOS, type

**BREAK ON ENTER**

### Example:

**BREAK ENTER**

displays the current setting of **CTRL C**.

# CHDIR (Change Directory)

Internal

CHDIR [*pathname*]  
CD [*pathname*]

Changes the current directory, or the home directory of the specified drive, to the directory specified by *pathname*. CHDIR also verifies the current directory.

## Parameters:

If you are changing the current directory, *pathname* must be another directory on the current disk.

If you are changing the home directory of a disk other than the current disk, you must specify the *drive* that contains the directory. *pathname* must be another directory on that disk.

If you omit the *pathname*, MS-DOS displays the *pathname* of your current directory. This lets you verify a directory change. It also helps in case you forget the name of the directory.

## Sample Uses:

**Entering Commands.** By changing directories, you can save much time when entering commands.

Suppose (1) the root directory on Drive A is your current directory, and (2) the root directory on Drive B is that drive's home directory. Now, suppose you want to copy several files from the B:\USER\MEMOS to the A:\USER\LETTERS. If you do not change directories, you must type the following to copy each file:

```
COPY B:\USER\MEMOS\filename  
A:\USER\LETTERS\filename (ENTER)
```

Instead, you can change your current directory to \USER\LETTERS by typing

```
CHDIR \USER\LETTERS (ENTER)
```

Then you can change your Drive B home directory to \USER\MEMOS by typing

**CHDIR B:\USER\MEMOS (ENTER)**

Now you can copy each file by typing only

**COPY B:filename filename (ENTER)**

**Executing Application Programs.** When executing an application program, you are likely to store your information in several data files in the same directory. Therefore, it may be convenient to make that directory your current directory.

Suppose you use a mailing list program to keep track of magazine subscriptions. You might create three data files: one sorted by your customers' last names, another by zip code, and another by subscription expiration date.

You can store all three files in a directory called \MAGMAIL. When running the program, make \MAGMAIL your current directory. Then you can quickly access and transfer data between files.

**Hard Disk Users:** Because you are likely to store several application programs on your hard disk, you may want to put each in a separate directory (See MKDIR). For example, you may want to store an Accounts Receivable program in a directory called \AR. When you are ready to use the program, use CHDIR to make \AR your current directory.

**Writing Application Programs.** By including a CHDIR command when you write an application program, you can avoid having to manually change directories when you run the program.

Suppose you write the mailing list program described above. You can have the program create the \MAGMAIL directory the first time you run the program (see MKDIR). Then you can have it make \MAGMAIL the current directory each time you start the program.

### Changing Your Current Directory to Another Disk.

You can use the CHDIR command to do this. First, however, you must make that disk your current disk.

Suppose you are in Drive A and your Drive B home directory is the root directory. You want your current directory to be B:\USER. At the A> prompt, type

B: (ENTER)

MS-DOS puts you in the root directory on Drive B and displays the new command ready prompt,

B>

Now you can change directories by typing

CHDIR \USER (ENTER)

If you type CHDIR (ENTER), MS-DOS verifies that \USER is now your current directory.

### Examples:

CHDIR .. (ENTER)

puts you in (changes your current directory to) the parent directory of your current directory.

CHDIR \ (ENTER)

puts you in the root directory of the current disk.

CHDIR \BIN\USER (ENTER)

puts you in the directory \BIN\USER on the current disk.

CHDIR (ENTER)

displays the pathname of your current directory. If, for example, you are in the directory \BIN\USER on the disk in Drive B, MS-DOS displays B:\BIN\USER.

CHDIR B:\USER (ENTER)

changes the home directory of Drive B to \USER.

**CHKDSK** [*drive*] [/F] [/V] [>*pathname*]

Checks the directory of the Tandy MS-DOS disk in the current or specified *drive* for errors. You should run CHKDSK occasionally on each disk.

CHKDSK doesn't prompt you to insert the disk; it assumes the disk is in the drive.

After checking the directory, CHKDSK displays the proper error messages, if any, and then a status report.

Here is a sample report. The numbers vary.

```
737280 bytes total disk space
 40960 bytes in 2 hidden files
   512 bytes in 2 directories
 34560 bytes in 8 user files
661248 bytes available on disk
```

```
131072 bytes total memory
105038 bytes free
```

**Note:** The report includes two hidden files. These are the system files, which are IO.SYS and MSDOS.SYS. You cannot see these files when you use the DIR command.

If you wish, you may redirect the output from CHKDSK to a file.

## Parameters:

*pathname* specifies the file to which CHKDSK is to redirect its output. Do not use this parameter if you use the /F parameter.

**/F** tells CHKDSK to correct (fix) any errors it can and update the disk. Do not use this parameter if you include a *pathname*.

**/V** causes CHKDSK to display messages while it is running and gives detailed information about any errors it finds.

## Sample Use:

Be sure to run CHKDSK before storing information on an old or frequently used disk.

## Remarks:

The /F parameter corrects the following errors automatically:

- Invalid sub-directory entry
- Cannot CHDIR to *filename*
- Tree past this point not processed
- First cluster number is invalid
- entry truncated
- Allocation error, size adjusted
- Has invalid cluster, file truncated

The /F parameter does **not** correct the following errors. **You** must correct them.

- Invalid drive specification  
Use a valid drive specification.
- Invalid parameter  
Use valid parameters only.
- Disk error reading FAT  
Use the COPY command to copy all files to another disk.
- Disk error writing FAT  
Use the COPY command to copy all files to another disk.
- Incorrect DOS version
- Insufficient memory  
Processing cannot continue

There is not enough memory to run CHKDSK for this disk. You must obtain more memory.

Errors found, F parameter not specified  
Corrections will not be written to disk

Specify the /F switch to correct the errors.

Invalid current directory  
Processing cannot continue

Restart the system and re-run CHKDSK.

Cannot CHDIR to root  
Processing cannot continue

The disk you are checking is bad. Restart the system and try using RECOVER command.

*filename* is cross linked on cluster

Make a copy of the file you want to keep; then delete both cross-linked files.

*x* lost clusters found in *y* chains  
Convert lost chains to files (Y/N)?

If you have specified the /F switch and you type **Y (ENTER)**, CHKDSK creates a directory entry and a file in which you can resolve this problem (files created by CHKDSK are named FILEnnnn.CHK). CHKDSK displays: *X bytes disk space freed*. If you have not specified the /F switch, CHKDSK frees the clusters and displays: *X bytes disk space would be freed*; it does not create a directory entry and file.

Probable non-DOS disk  
Continue (Y/N)?

You are not using an MS-DOS disk. Type **Y (ENTER)** to continue processing or **N (ENTER)** to stop processing.

Insufficient room in root directory  
Erase files in root and repeat CHKDSK

CHKDSK cannot process until you delete files in the root directory.

Unrecoverable error in directory  
Convert directory to file (Y/N)?



If you type **Y** (**ENTER**), CHKDSK converts the bad directory into a file. You can then fix the directory or delete it. If you type **N** (**ENTER**), the directory becomes unusable, and you can neither fix nor delete it. You should always type **Y** (**ENTER**).

## Examples:

**CHKDSK /F** (**ENTER**)

checks the directory of the current disk, displays the errors, asks if you want to fix them, and acts accordingly.

**CHKDSK B: /V** (**ENTER**)

checks the directory of the disk in Drive B, displaying the name of every directory and file on the disk, and reports on its progress. MS-DOS asks if you want to fix any errors. It cannot do so, however, until you specify the /F switch.

**CHKDSK B: >\USER\TOM\errors** (**ENTER**)

checks the directory of the disk in Drive B and outputs any errors to the file Errors in the \USER\TOM directory that is on the current disk. MS-DOS asks if you want to fix any errors. It cannot do so, however, until you specify the /F switch.

## CLS

Clears the screen.

The CLS command causes MS-DOS to send the ANSI escape sequence ESC[2J to your screen, thus clearing it.



## Sample Use:

After you enter two or three commands, your screen becomes cluttered and, therefore, hard to read. Whenever this happens, use the CLS command.

## Example:

CLS **ENTER**



# COMPDUPE (Compare/Duplicate)

External

## COMPDUPE [/D] [/S]

(FLOPPY DISK ONLY) Makes a mirror-image copy of the floppy disk in Drive A onto the disk in Drive B and compares the disks (duplication mode), or only compares the disks (comparison mode). Unlike the COPY command, COMPDUPE copies all information on a disk.

You can choose whether you want MS-DOS to format the Drive B disk during the duplication.

Immediately after you enter the COMPDUPE command, MS-DOS displays a screen similar to the following (the message varies slightly, depending on the mode chosen):

When ready press the SPACE bar, or to abort press CONTROL-C

After you press **(SPACEBAR)**, a dashed line is displayed:

Each dash in the dashed line represents a track. During the comparison or duplication, MS-DOS replaces each dash with one of the following symbols:

- \* = source disk track read with no errors
- . = track duplicated or compared with no errors
- S = track error reading the source disk (Drive A)
- F = track error formatting the destination disk (Drive B)
- D = track error writing to the destination disk
- C = track error (source does not compare to destination)

Note that COMPDUPE prompts you to insert the disks. If the disks already are in the drives, simply press **(SPACEBAR)**.

When the comparison or duplication is complete, COMPDUPE displays a message, asking if you want to do another.

## Parameters:

**/D** enters the command's duplication mode.

**/S** causes MS-DOS to skip formatting while duplicating. This makes the duplication faster. If you omit the **/S** option, MS-DOS formats the Drive B disk while duplicating. Do not include the **/S** parameter if you don't include the **/D** parameter.

Omitting both parameters enters the command's comparison mode.

## Sample Uses:

By using **COMPDUPE** to duplicate your MS-DOS system disk, you can create another operating system disk. To do so, insert your MS-DOS disk into Drive A and a blank disk into Drive B. Then, at the **A >** prompt, type

```
COMPDUPE /D (ENTER)
```

We suggest that you make copies of all your disks — especially your system disk and application program disks — and store your originals in a safe place. This reduces the possibility of losing information should something happen to the disk you are using.

If your destination disk already is formatted under MS-DOS, you can save time by skipping formatting during duplication. Before entering the **COMPDUPE** command, however, use the **DIR** command to make sure you don't need any information on the disk. Then type

```
COMPDUPE /D /S (ENTER)
```

## Examples:

```
COMPDUPE (ENTER)
```

compares the Drive A and B disks, track by track.

```
COMPDUPE /D (ENTER)
```

formats the Drive B disk and makes a mirror-image duplicate of the Drive A disk onto it.

### COMPDUPE /D /S **(ENTER)**

makes a mirror-image duplicate of the Drive A disk onto the Drive B disk, which was formatted before the COMPDUPE command was entered.

## Errors:

If no error occurs, MS-DOS displays a message to that effect.

If any error occurs, in either COMPDUPE mode, MS-DOS displays this message:

### Disks do not compare

The following errors cause MS-DOS to abort COMPDUPE and to return to the command ready prompt:

#### Bad switch syntax

You used an invalid switch syntax (such as /S only) or a switch other than /D or /S.

#### Destination disk format error

An error occurred during formatting. The disk may be “bad” or you may have inserted it in the drive incorrectly.

If you are duplicating, the following errors cause MS-DOS to abort COMPDUPE and to return to the command ready prompt.

If you are comparing, they change the track status indicator to the appropriate character (S, F, D, or C) and do not display an error message.

#### Destination disk read error

#### Destination disk write error

#### Source disk read error

**COPY** *source pathname* [*target pathname*]  
[**/A**] [**/B**] [**/V**]

Copies one or more files (1) to the same directory (as the source), giving the new file a different filename, or (2) to another directory, giving the new file the same or a different filename. In the second case, the target directory may be on any disk.

By varying the syntax of COPY slightly, you can also use it to:

- “Append” one or more files to the end of an existing file, keeping the target file’s filename.
- “Combine” files into a new file that has a unique filename.

(See COPY/APPEND and COPY/COMBINE for more information on these uses of COPY.)

## Parameters:

If you omit the *filename* from the *target pathname*, MS-DOS assumes you want to give the new file the same filename as the source file.

**/A, when used with a source file**, tells MS-DOS to treat the file as an ASCII file (also called a “text” or “data” file). MS-DOS copies only the information up to the first end-of-file (EOF) character.

**/A, when used with the target file**, tells MS-DOS to add an EOF character to the end of the file.

**/B, when used with a source file**, tells MS-DOS to treat the file as a binary file, such as a program file. Therefore, MS-DOS copies the entire file.

**/B, when used with the target file**, tells MS-DOS to not add an EOF character to the end of the file.

Each of these switches (/A and /B) affects the file immediately preceding it in the command line and all files following (up until the file preceding the next switch).

For example, if you type this:

```
COPY thisfile /A thatfile (ENTER)
```

the /A parameter affects both files. However, if you type this:

```
COPY thisfile /A thatfile /B (ENTER)
```

the /A parameter affects only the file Thisfile.

**If you omit the /A and /B switches, MS-DOS uses /B.** Notice that this default is the opposite of that for the COPY/APPEND and COPY/COMBINE commands.

**/V** tells MS-DOS to verify that the sectors written to the disk are recorded properly. This parameter causes the copy to run more slowly because MS-DOS must check each entry recorded on the disk.

## Sample Uses:

**Copying a File to Another Disk.** There are several reasons you may want to copy a file to another disk. Here are a few:

- To give the file to someone else.
- To create a "backup" copy of the file, without using COMPDUPE to duplicate the entire disk.
- To reduce file "fragmentation" (to make files contiguous again). Fragmentation, which occurs when you do much creating and deleting, wastes disk space. Copying the files to another disk is the only way to reduce fragmentation.

Remember, **the target disk must be formatted** and the target directory must exist.

Suppose you are in A:\USER on Drive A and the file Personel.dat is in that directory's parent directory (the root directory on Drive A). Suppose also that \USER is your home directory on Drive B. You can copy the file by typing

```
COPY .\personel.dat /A B: . . (ENTER)
```

MS-DOS copies the file from A:\ to B:\.

Suppose you have a disk containing several fragmented data files. To reduce fragmentation, copy all files to another disk. Use this command:

**COPY \*.\* /A B: (ENTER)**

MS-DOS copies all files from A:\ to B:\.

**Transferring External Commands to Another Directory.** When you receive your system disk, all external commands are in the root directory. As you become more familiar with MS-DOS, you may want to tailor your directory structure to your own needs. Therefore, you may want to move some of these commands to their own directory.

Assume that you always use your system disk in Drive A and that you are currently in the root directory on that disk. Use the MKDIR command, as follows, to create the directory \BIN on your system disk:

**MKDIR \BIN (ENTER)**

Then copy all command files **except COMMAND.COM** into \BIN. For example, to copy the FORMAT.COM file, type

**COPY format.com \BIN (ENTER)**

MS-DOS copies the file from the current directory to the \BIN directory, keeping the filename the same.

Now you can use the PATH command to tell MS-DOS to search A:\BIN, as well as A:\, for your external commands. To do so, type

**PATH A:\BIN (ENTER)**

You may now erase the copied commands from the root directory. Since you are still in that directory, you need specify only the *filename* (not the entire *pathname*). For example, to erase FORMAT.COM, type

**ERASE format.com (ENTER)**



MS-DOS erases the command from the current directory (the root directory) only. The file still exists in the \BIN directory.

## **Examples:**

**COPY memos.txt /A B:corr.txt (ENTER)**

copies the file Memos.txt from the current directory to the home directory of the disk in Drive B, naming the new file Corr.txt. MS-DOS copies information only up to the first EOF character and adds an EOF character to the end of the new file.

**COPY B:taxes83.dat /A taxes84.dat (ENTER)**

copies the file Taxes83.dat from the home directory in Drive B to the current directory, naming the new file Taxes84.dat. MS-DOS copies information only up to the first EOF character and adds an EOF character to the end of the new file.

**COPY prog.exe B:prog1.exe (ENTER)**

copies the file Prog.exe that is in the current directory to the home directory of Drive B. MS-DOS does not add an EOF character to the end of the new file, Prog1.exe.

**COPY \*.lst /A combin.prn (ENTER)**

finds all files that have the extension .lst (in the current directory) and copies them to the new file Combin.prn in the current directory. MS-DOS copies information only up to the first EOF character and adds an EOF character to the end of the new file.

## **Error:**

You cannot copy a file onto itself. (You cannot copy it to the same directory, giving it the same name.)

For example, this command causes MS-DOS to abort the copy:

**COPY B:memos B:memos (ENTER)**

This one also causes the COPY to abort. You are trying to copy the file Memos that is in the current directory to another file Memos, also in the current directory.

COPY memos **(ENTER)**

MS-DOS displays the error message

File cannot be copied onto itself

0 File(s) copied

## **COPY *target pathname* + *source pathname1* [+ *source pathname2 . . .*] [/A] [/B] [/V]**

Adds one or more files to the end of another existing file. The files are added in the order in which you list them.

Suppose you have these files in your current directory:

<u>Biglist</u>	<u>List1</u>	<u>List2</u>
Bob Anthony	Anne Barnes	Jerry Day
Carol Apple	Ted Erickson	Karen Ellis
Curtis Kelly	Mike McAdam	Jennifer Peters
Susan Leonard	Dave Shultz	Larry Thomas

If you type this command:

**COPY biglist + list1 + list2 **ENTER****

the new contents of the file Biglist are

Biglist

Bob Anthony  
Carol Apple  
Curtis Kelly  
Susan Leonard  
Anne Barnes  
Ted Erickson  
Mike McAdam  
Dave Shultz  
Jerry Day  
Karen Ellis  
Jennifer Peters  
Larry Thomas

When you append files, the original *source files* still exist separately. The information from them is **copied**, not moved.

### **Parameters:**

**/A, when used with a source file**, tells MS-DOS to treat the file as an ASCII file (also called a "text" or "data" file). MS-DOS copies only the information up to the first end-of-file (EOF) character.

**/A, when used with the target file,** tells MS-DOS to add an EOF character to the end of the file.

**/B, when used with a source file,** tells MS-DOS to treat the file as a binary file, such as a program file. Therefore, MS-DOS copies the entire file.

**/B, when used with the target file,** tells MS-DOS to not add an EOF character to the end of the file.

Each of these switches (/A and /B) affects the file immediately preceding it in the command line and all files following (up until the file preceding the next switch).

For example, if you type this:

```
COPY onefile.txt /A + myfile.txt + samfile.txt ENTER
```

the /A parameter affects all three files. However, if you type this:

```
COPY onefile.txt /A + myfile.txt /B + samfile.txt  
ENTER
```

the /A parameter affects only the target file, Onefile.txt.

**If you omit the /A and /B switches, MS-DOS uses /A.** Notice that this default is the opposite of that for the regular COPY command.

**/V** parameter tells MS-DOS to verify that the sectors written to the disk are recorded properly. This parameter causes the copy to run more slowly because MS-DOS must check each entry recorded on the disk.

## **Sample Use:**

Use this command to append information to existing data files. Suppose, for example, that you have these files in the current directory: Gizmo.dat, Widget.dat, and Inventory.dat. Each of the first two files contains information on a new product. The last contains information on all your products. Type

```
COPY inventory.dat + gizmo.dat + widget.dat ENTER
```

to append Gizmo.dat and Widget.dat to Inventory.dat.

## **Examples:**

**COPY B:read.dat + write.dat + print.dat (ENTER)**

appends the files Write.dat and Print.dat that are in the current directory, to the file Read.dat that is in the home directory of Drive B. MS-DOS adds an EOF character to the end of Read.dat.

**COPY big.lst + \*.lst (ENTER)**

appends all files that have the extension .lst (except Big.lst) and that are in the current directory to the file Big.lst that also is in the current directory. MS-DOS adds an EOF character to the end of Big.lst.

**COPY prog1.exe /B + prog2.exe (ENTER)**

appends the file Prog2.exe that is in the current directory to the file Prog1.exe that also is in the current directory. MS-DOS copies only information up to the first EOF character in each file. MS-DOS does not add an EOF character to the end of Prog2.exe.

**COPY** *source pathname1*  
[+ *source pathname2 . . .*]  
*target pathname* [/A] [/B] [/V]

Combines any number of *source* files into a *target* file that has a unique filespec. (The *target* file is new.) The files are added in the order in which you list them.

Suppose you have these files in your current directory:

<u>Oldlist1</u>	<u>Oldlist2</u>
Anne Barnes	Jerry Day
Ted Erickson	Karen Ellis
Mike McAdam	Jennifer Peters
Dave Shultz	Larry Thomas

If you type this command:

COPY oldlist1 + oldlist2 newlist **(ENTER)**

MS-DOS creates the file Newlist in the current directory. The file contains the names shown here:

Newlist  
Anne Barnes  
Ted Erickson  
Mike McAdam  
Dave Shultz  
Jerry Day  
Karen Ellis  
Jennifer Peters  
Larry Thomas

When you combine files, the original *source files* still exist separately. The information from them is **copied**, not moved.

## Parameters:

**/A, when used with a source file**, tells MS-DOS to treat the file as an ASCII file (also called a “text” or “data” file). MS-DOS copies only the information up to the first end-of-file (EOF) character.

**/A, when used with the target file,** tells MS-DOS to add an EOF character to the end of the file.

**/B, when the used with a source file,** tells MS-DOS to treat the file as a binary file, such as a program file. Therefore, MS-DOS copies the entire file.

**/B, when used with the target file,** tells MS-DOS to not add an EOF character to the end of the file.

Each of these switches (/A and /B) affects the file immediately preceding it in the command line and all files following (up until the file preceding the next switch).

For example, if you type this:

```
COPY myfile.txt /A + salfile.txt onefile.txt (ENTER)
```

the /A parameter affects all three files. If, however, you type this:

```
COPY myfile.txt /A + salfile.txt /B onefile.txt (ENTER)
```

the /A parameter affects only Myfile.txt.

**If you omit the /A and /B switches, MS-DOS uses /A.** Notice that this default is the opposite of that for the regular COPY command.

**/V** tells MS-DOS to verify that the sectors written to the disk are recorded properly. This parameter causes the copy to run more slowly because MS-DOS must check each entry recorded on the disk.

**Warning:** Never combine files into a *target file* that has the same name as an existing file. Instead, append other files to the existing file.

For example, suppose your current directory contains the files A.lst, B.lst, and All.lst. Do not enter this command:

```
COPY *.lst /B all.lst (ENTER)
```

If you do, COPY combines A.lst and B.lst into the All.lst target file, which replaces (destroys) the All.lst source file. Then COPY displays the message “Content of destination lost before copy.”

### Sample Use:

Suppose you have two files, Phone4.dat and Phone5.dat, each of which contains a list of phone extensions for personnel on a particular floor. As your company grows, however, you must move all these people to the ninth floor. Rather than type in all the information again, you can combine the two files into a new file. To do so, type

```
COPY phone4.dat + phone5.dat phone9.dat (ENTER)
```

Now you can sort your new file alphabetically, using the SORT command (see SORT).

### Examples:

```
COPY B:memos.txt + B:letters.txt B:corr.txt (ENTER)
```

combines the files Memos.txt and Letters.txt that are in the home directory on Drive B. COPY copies information only up to the first EOF character in each source file. Then it places the information in the new file Corr.txt in the same directory and adds an EOF character to the end of that file.

```
COPY A:\stats1.dat + A:\stats2.dat B:\allstats.dat (ENTER)
```

combines all information from the files Stats1.dat and Stats2.dat that are in the root directory of Drive A. Then COPY places the information in the new file Allstats.dat in the root directory of Drive B. MS-DOS adds an EOF character to the end of Allstats.dat.

```
COPY a.com + b.com + B:c.com oneprog.com /B (ENTER)
```

combines the files A.com and B.com that are in the current directory, with the file C.com that is in the home directory of Drive B. COPY copies information only up to the first EOF character in each source file. Then it places the information in the new file Oneprog.com in the current directory. It does not add an EOF character to the end of that file.



## CTTY *device*

Changes the input/output (I/O) device to the *device* specified.

Normally, input comes from the keyboard and output goes to the screen.

### Parameters:

The *device* can be either of the following:

- **AUX** specifies an auxiliary device, normally the RS-232C serial port.
- **CON** specifies the console (input from the keyboard and output to the screen).

It cannot be LST or PRN (list or print), however, because the printer is not capable of input.

### Examples:

CTTY AUX **(ENTER)**

moves all command input/output (I/O) from the current device to the auxiliary device.

CTTY CON **(ENTER)**

changes the input device to the keyboard and the output device to the screen.

## DATE [*mm/dd/yyyy*]

Enters or changes the date known to the system. This date is recorded in the directory for any files you create or change. You can also use this command to display the current date.

You can change the date from the keyboard or from a batch file. (Normally, MS-DOS displays a date prompt each time you start up your system. It does not, however, if you use an Autoexec.bat file. Therefore, you may want to include a DATE command in that file.)

**Note:** MS-DOS is programmed to change the months and years correctly, taking into account leap years and the number of days in the months.

### Parameters:

*mm-dd-yyyy* specifies the date in numerical form.

- *mm* (month) is a 1- or 2-digit number from 1-12
- *dd* (day of month) is a 1- or 2-digit number from 1-31
- *yyyy* (year) is a 2-digit number from 80-99 (1900 is assumed) or a 4-digit number from 1980-2099

**Note:** If the month or day is less than 10, you may include a leading zero (for example, 09/09/84). However, this is not necessary. When MS-DOS stores and displays the date, it includes a leading zero in a 1-digit day and excludes it from a 1-digit month (for example, it stores 9/09/1984).

You may separate the date, month, and year with either slashes or hyphens.

If you omit the *mm-dd-yyyy* parameter, DATE displays the current date and asks you to enter the new date. If you don't want to change the date, press **(ENTER)**. If you do want to change it, enter it in the *mm-dd-yyyy* format.

### Sample Uses:

When you change the date known to the system, you also change the date in any application program you use. This can be very handy.

Suppose that you have a program that keeps track of purchase orders according to the date received. For some reason, you get behind and can't enter the information on that date. Simply enter it later, after "turning back the calendar" to the necessary date.

You can also use DATE and TIME to include the date and time on a printout. For example, press **(PRINT)** or **(CTRL) (P)** to send all output to the line printer, as well as to the screen. Then type

```
DATE (ENTER) (ENTER)  
TIME (ENTER) (ENTER)
```

Press **(PRINT)** again to turn off the "print output" function. Now use **(SHIFT) (PRINT)**, **(PRINT)**, **(CTRL) P**, or the PRINT command as necessary to make your printout.

## Examples:

```
DATE (ENTER)
```

displays the current date and prompts you for the new date. For example, if the date is July 7, 1984, MS-DOS displays:

```
Current date is Sat 7-07-1984  
Enter new date:
```

You may now change the date or press **(ENTER)** to bypass the prompt.

```
DATE 11/15/1984 (ENTER)
```

enters the current date as Thursday, November 15, 1984.

## Error:

If the options or separators aren't valid, DATE displays the message

```
Invalid date  
Enter new date:
```

It then waits for you to enter a valid date.

See the ERASE command.



# DIR (Directory)

Internal

## DIR [*pathname*][/P][/W]

Displays (1) information about all files or the specified files that are in the current directory or in the directory specified by *pathname*, or (2) information about the one file specified by *pathname*.

First, MS-DOS displays the disk's volume label, which was assigned during formatting. If you did not assign a label, MS-DOS displays a message to that effect.

Then MS-DOS lists each file with its size (in bytes) and the time and date of the last modification. It then gives the number of files listed and the number of bytes remaining on the disk.

Here is a sample listing for all files that are in B:\USER:

Volume in drive B has no label

Directory of B:\user

.			<b>DIR</b>	6-24-83	12.36a
..			<b>DIR</b>	6-24-83	12.36a
MAILSRT	BAT	252		6-28-83	12.42a
TEST2	TST	1512		6-28-83	12.56a
INVNTORY	DAT	6759		6-30-83	8.45a
PERSONEL	DAT	5094		6-30-83	5.08p
0 File(s)		691794 bytes free			

Here is a sample listing for the file Mailsrt.bat that is in B:\USER:

Volume in drive B has no label

Directory of B:\user

MAILSRT	BAT	1512	6-28-83	12.42a
1 File(s)		691794 bytes free		

## Parameters:

If you omit the *pathname*, MS-DOS lists the directory entries for all files that are in the current directory. (If the disk contains only one directory, MS-DOS lists all files on the disk.) You may use the MS-DOS wild cards in the *pathname*.

**/P** selects the “page” mode. With **/P**, display of the directory pauses when the screen is filled. It displays this message:

Strike a key when ready . . .

To resume display of output, press **(SPACEBAR)**.

**/W** selects a wide display. With **/W**, MS-DOS displays file-names only; it does not display the files’ sizes or modification dates.

## Sample Uses:

You can use the DIR command to see how much space one file takes and how much free space remains on a disk.

Suppose you want to see how much space B:\USER\mailsr.bat takes and how much space remains on Drive B. Type

```
DIR B:\USER\mailsr.bat (ENTER)
```

Use DIR when you forget the name of a file. For example, if you are in Drive A and need to know the name of a file in Drive B, type

```
DIR B:\ (ENTER)
```

The backslash indicates the root directory. If you do not use it, MS-DOS lists the files in the home directory. If the file you want is not in the root directory, use the DIR command as needed to check the subdirectories.

You may want to index your disk by printing the directory listing. To do so, press **(PRINT)**. Then type

```
DIR (ENTER)
```

Pressing **(PRINT)** causes MS-DOS to send all output to the printer, as well as to the screen, until you again press **(PRINT)**.

## **Examples:**

**DIR B: (ENTER)**

displays a list of all files in the home directory of Drive B.

**DIR /W (ENTER)**

displays, in the wide mode, the filenames of all files in the current directory.

**DIR \USER\\*.bat /P**

displays a list of all batch files in the \USER directory on the current drive. MS-DOS halts ("pauses") the display when the screen is full. Press **(SPACEBAR)** to continue.

**DIR A:\USER\ACCT.\***

displays a list of files that have the name Acct — regardless of their extensions — and that are in the A:\USER directory.

## DISKCOPY [*source drive*] [*target drive*]

Copies the contents of the disk in the *source drive* to the disk in the *target drive*.

DISKCOPY is similar to COMPDUPE. Unlike COMPDUPE, however, DISKCOPY **does not** do the following:

- Format the target disk. You must use the FORMAT command before using DISKCOPY.
- Compare the disks.
- Give a track-by-track analysis of the copy's progress.

After copying, DISKCOPY prompts:

```
Copy complete  
Copy another (Y/N)?_
```

If you press **(Y)**, MS-DOS prompts you to insert the disks. Then it performs the next copy, using the drives you specified earlier.

If you press **(N)**, MS-DOS doesn't perform another copy.

### Parameters:

If the *source* and *target* drives are the same, MS-DOS performs a single-drive copy. It prompts you to insert the disks at the appropriate times. Press **(SPACEBAR)** to continue.

If you omit either *drive* specification, MS-DOS uses the current drive.

Therefore, if you omit both *drive* specifications, MS-DOS does a single-drive copy on the current drive. (This assumes that the DISKCOPY.COM file is in the current directory, so that MS-DOS can find it, or that you use PATH beforehand to tell MS-DOS where to look for the file.)



## Sample Use:

By using DISKCOPY to duplicate your MS-DOS system disk, you can create another operating system disk. We suggest that you make copies of all your disks — especially your system disk and application program disks — and store your originals in a safe place. This reduces the possibility of losing information should something happen to the disk you are using.

If you have a hard disk system (only one floppy drive), type  
**DISKCOPY (ENTER)**

MS-DOS prompts you to insert the “Drive A” disk. Insert the *source* disk into Drive A. It then prompts you for the “Drive B” disk. Remove the *source* from Drive A and insert the *target*. Press **(SPACEBAR)**. MS-DOS continues to prompt you, as necessary.

If you have a floppy disk system (two floppy drives), type  
**DISKCOPY A: B: (ENTER)**

MS-DOS copies all information from the disk in Drive A to the disk in Drive B.

Floppy disk users: DISKCOPY is also the command to use to copy an entire data disk. To do this, type

**DISKCOPY A: B: (ENTER)**

MS-DOS displays:

Insert source diskette in drive A:  
Insert target diskette in drive B:  
Strike any key when ready

When the drive light goes out, remove the system diskette from Drive A and insert the *source* data disk. Insert the *target* data disk in Drive B. Then press **(SPACEBAR)** to continue.

## **Examples:**

**DISKCOPY** **(ENTER)**

copies all information from a disk in the current drive to another disk in the current drive. MS-DOS prompts you to insert disks, as necessary. (This example assumes that the DISKCOPY.COM file is in the current directory, so that MS-DOS can find it, or that you use PATH beforehand to tell MS-DOS where to look for the file.)

**DISKCOPY A: B: (ENTER)**

copies all information from the disk in Drive A to the disk in Drive B. Before starting the diskcopy, MS-DOS prompts you to insert disks.

## **Remarks:**

The disks must have the same number of physical sectors, and those sectors must be the same size.

Files become fragmented (not contiguous) when much creating and deleting is done on a disk. A fragmented disk can cause poor performance because of delays involved in finding, reading, or writing a file. If this is the case, you must use the COPY command to reduce fragmentation. (See the COPY command.)

## **Error:**

If MS-DOS finds disk errors during DISKCOPY, it displays:

**DISK error while reading drive A**  
**Abort, Ignore, Retry?**

Refer to Appendix A, "Problems and Error Messages," for information on this error message.

## ECHO [ON|OFF|*message*]

Turns the batch ECHO feature on or off.

Normally, commands in a batch file are displayed (echoed) as you run the file. ECHO OFF turns off this feature. ECHO ON turns it on again. (Note: ECHO is automatically turned on again after execution of the batch file.)

ECHO *message* displays the specified message, regardless of whether ECHO is ON or OFF.

### Parameters:

If you don't specify ON or OFF, and you omit a *message*, MS-DOS displays the current setting of ECHO.

### Sample Use:

Whenever you want to protect information in a batch file, you can turn ECHO OFF. You can also use ECHO OFF when you don't want to clutter the screen with unnecessary information.

### Example:

The following batch file formats a disk in Drive B and then checks the disk. (The file uses some external commands, so it must be created in a directory that contains the external commands.)

```
COPY con prepdisk.bat (ENTER)
ECHO OFF (ENTER)
REM This file formats and checks disks in Drive B
  (ENTER)
REM It is called Prepdisk.bat (ENTER)
FORMAT B: (ENTER)
ECHO    Do you want to check Drive B?
PAUSE (ENTER)
CHKDSK B: (ENTER)
ECHO ON (ENTER)
(F6) (ENTER)
```

When you run the file, MS-DOS displays

```
COPY con prepdisk.bat
ECHO OFF
```

Insert new diskette for drive B:  
and strike any key when ready . . .

Formatting tracks

---

*nnnnnn* bytes total disk space  
*nnnnnn* bytes available on disk

Format another (Y/N)?n Do you want to check Drive B?  
Strike a key when ready . . .

*nnnnnn* bytes total disk space  
*nnnnnn* bytes available on disk

*nnnnnn* bytes total memory  
*nnnnnn* bytes free

As you can see, the commands are not echoed. The prompt “Do you want to see the directory and check Drive B?” is displayed, however. This is because the ECHO command used here includes the *message* parameter.

**ERASE** [*pathname*]  
**DEL** [*pathname*]

Erases (deletes) one or more files from current directory or from the directory specified by *pathname*.

## Parameters:

**Warning:** If you omit the *filename* from the *pathname*, MS-DOS assumes you want to erase **all** files in the directory. (Using the wild card \*.\* as the *filename* is the same as omitting the *filename*.)

If you omit the entire *pathname*, MS-DOS assumes you want to erase **all** files in the current directory. (Using the wild card \*.\* as the *pathname* is the same as omitting the *pathname*.) If the disk contains only one directory, MS-DOS erases all files on the disk (except . and ..).

To avoid erasing important files, use the DIR command, with the appropriate wild card, beforehand to check which files you need.

## Sample Uses:

By erasing unnecessary files, you can create more free space in a directory. Suppose you combine the files A.lst and B.lst, which are in the current directory, into a new file, All.lst, on a different directory.

Assuming the current directory contains no other files that have the extension .lst, you may erase A.lst and B.lst by typing

```
ERASE *.lst (ENTER)
```

**Warning:** If you include an extension, such as the .lst used here, MS-DOS does **not** prompt you before erasing the files, regardless of whether or not you use a wild card. It prompts only when you use the full wild card, \*.\*.

You must erase all files in a directory (except the files . and ..) before you can use RMDIR to remove the directory.

## Examples:

ERASE \BIN\USER\MARY\text.txt **(ENTER)**

erases the file Text.txt from the directory \BIN\USER\MARY on the current disk.

ERASE B:\USER\JOHN\\*.\* **(ENTER)**

tells MS-DOS to delete all the files in the directory \USER\JOHN on Drive B. MS-DOS asks: *Are you sure?* If you type **Y (ENTER)**, MS-DOS erases the files. If you type **N (ENTER)**, MS-DOS does not erase the files.

ERASE B:\BIN\USER\MARY **(ENTER)**

tells MS-DOS to erase all files in the \BIN\USER\MARY directory on Drive B. MS-DOS asks if you are sure you want to do this and then acts accordingly.

ERASE file2 **(ENTER)**

erases the file File2 from the current directory.

# EXE2BIN (Executable to Binary)

External

**EXE2BIN** *source pathname* [*target pathname*]

Converts the .exe (executable) file, specified by *source pathname*, to .com file format (binary format).

**EXE2BIN is an advanced command, recommended for advanced users only.**

The source file must be in valid .exe format produced by the linker. The resident, or actual code and data part of the file, must be less than 64K. There must be no STACK segment.

## Parameters:

If you don't include an extension in the ***source pathname***, the extension defaults to .exe.

***target pathname*** specifies a file to receive the converted program file. If you omit the *drive*, MS-DOS uses the *drive* specified in the *source pathname*. If you omit the *extension*, it gives the new file the extension .bin. If you omit the entire *target pathname*, MS-DOS uses the *source pathname*.

### Kinds of Conversion:

Two kinds of conversions are possible, depending on whether the initial CS:IP (Code Segment:Instruction Pointer) is specified in the .exe file.

If CS:IP is not specified, EXE2BIN assumes a pure binary conversion. If segment fixups are necessary (the program contains instructions requiring segment relocation), EXE2BIN prompts for the fixup value. This value is the absolute segment at which the program is to be loaded.

In a pure binary conversion, the resulting program is usable only when loaded at the absolute memory address specified by a user application. The command processor cannot properly load the program.

If CS:IP is specified as 0000:100H, EXE2BIN assumes the file is to be run as a .com file with the location pointer set at 100H by the assembler statement ORG. It deletes the first 100H bytes of the file. It allows no segment fixups, as .com files must be segment relocatable.

When this kind of conversion is complete, you may rename the resulting file, giving it a .com extension. Then the command processor can load and execute the program in the same way as the .com programs supplied on your MS-DOS disk.

### Sample Use:

Converting executable files to binary format saves space and speeds program loading.

### Examples:

EXE2BIN testfile.exe B: **(ENTER)**

Converts the file Testfile.exe that is in the current directory to binary format and places the new file, Testfile.bin, in the home directory of Drive B.

EXE2BIN A:\USER\oldfile.exe A:newfile **(ENTER)**

Converts the file Oldfile.exe that is in A:\USER to binary format and places the new file, Newfile.bin, in the home directory of Drive A.

### Errors:

If CS:IP does not meet either criterion (discussed in “Kinds of Conversions”) — or if it meets the .com file criterion but has segment fixups — EXE2BIN displays the following message:

**File cannot be converted**

EXE2BIN also displays this message if the file is not a valid executable file.

EXE2BIN may also display these error messages:

**File not found**

The file is not on the disk specified.

**Insufficient memory**

There is not enough memory to run EXE2BIN.



**File creation error**

EXE2BIN cannot create the output file. Run CHKDSK to determine if the directory is full or if some other condition caused the error.

**Insufficient disk space**

There is not enough disk space to create a new file.

**Fixups needed - base segment (hex):**

The source (.exe) file contains information that indicates that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be located.

**File cannot be converted**

The source file is not in the correct format.

**WARNING -Read error on EXE file.  
Amount read less than size in header**

This is a warning message only.

## EXIT

Exits the program COMMAND.COM (the command processor) and returns to a previous level, if one exists.

If you have exited an application program to use the command processor, use EXIT to return to the application.

### Sample Use:

While running an application program, you may need to use an MS-DOS command. To do so, you must exit your program.

For example, if you want to see your directory, you must use the MS-DOS DIR command. Type

**COMMAND** (ENTER)

to exit your application program and start the command processor. Then type

**DIR B:** (ENTER)

to display the directory of the disk in Drive B. Finally, type

**EXIT** (ENTER)

to return to your application program.

### Example:

**EXIT** (ENTER)

returns you to the previous level.

**FC** [/number] [/B] [/W] [/C] *pathname1*  
*pathname2* [>*target pathname*]

Compares the contents of two files, *pathname1* and *pathname2*, and sends the output to the screen or to the file specified by *target pathname*.

The two kinds of comparisons are:

- Line-by-line
- Byte-by-byte

In a line-by-line comparison, FC marks off blocks of lines, and then compares the lines within each block. The “Sample Use” section explains how FC determines where each block begins and ends.

In a byte-by-byte comparison, FC simply displays the bytes that are different.

## Parameters:

*pathname 1* and *pathname2* may be either “source files” (files that contain source statements of a programming language) or binary files (files output by the assembler, the linker, or a Microsoft high-level language compiler).

/B forces a binary comparison of both files. The two files are compared byte-by-byte, with no attempt to re-synchronize after a mismatch. FC displays the mismatches as follows:

```
--ADDRS----F1----F2-  
xxxxxxxx yy zz
```

where *xxxxxxxx* is the relative address of the pair of bytes from the beginning of the file. Addresses start at 00000000.

*yy* and *zz* are the mismatched bytes of *pathname1* and *pathname2*, respectively. If one file contains less data than the other, FC displays a message to that effect.

For example, if *pathname1* ends before *pathname2* ends, then FC displays

```
***Data left in pathname2***
```

**number** is the number of lines that must match for the file to be considered as matching again after FC finds a difference. It can be from 1 through 9. If you don't specify a *number*, it defaults to 3. Use this parameter only in source comparisons.

**/W** compresses "white" spaces (tabs and spaces) for the comparison. Thus, if several whites are together on one line, FC considers them as one white space. FC ignores whites that are at the beginning and end of lines.

For example (an underscore represents a white space),

`__More__data.to.be_found__`

matches

`More.data.to.be_found`

and

`____More____data.to.be__found____`

but does not match

`__Moredata.to.be_found`

Use the **/W** parameter only in source comparisons.

**/C** causes FC to ignore the case of letters. All letters in the files are considered uppercase letters. For example,

`Much_MORE.data.IS.NOT.FOUND`

matches

`much_more.data.is.not_found`

Use the **/C** parameter only in source comparisons.

## Sample Use:

Use FC when you want to determine which of two files is current and what changes have been made since the old version.

Suppose your current directory contains these source files. Each letter in the files represents a program line.

<u>Name1.src</u>	<u>Name2.src</u>
A	A
B	B
M	C
D	L
O	S
S	W
W	X
X	Y
Y	Z
Z	P
R	E
U	N
N	Q
Q	V
T	
V	

To compare the files line-by-line, type

FC /1 name1.asm name2.asm **ENTER**

## Section II / Commands Reference

---

FC begins by marking off the blocks of lines. Simply put, whenever FC encounters a match, it ends a block. Therefore, FC blocks off the lines as follows:

Name1.asm	Name2.asm
A	A
B	B
M	C
D	L
O	S
S	
W	W
X	X
Y	Y
Z	Z
R	P
U	E
N	N
Q	Q
T	V
V	

FC ignores matching blocks that are before the first mismatch. It also ignores blocks in which at least the specified *number* of lines match. In this case, the *number* is 1. For each set of blocks that contains a mismatch, however, FC displays the following information:

1. -----NAME1.SRC
2. The lines in the Name1.src block that differ from those in the corresponding Name2.src block
3. The line that is the same (the last line in the block)
4. -----NAME2.SRC

5. The lines in the corresponding Name2.src block that differ from those in the Name1.src block
6. The line that is the same (the last line in the block)

Therefore, for the files Name1.src and Name2.src, FC displays:

-----NAME1.SRC

M  
D  
O  
S

-----NAME2.SRC

C  
L  
S

-----NAME1.SRC

R  
U  
N

-----NAME2.SRC

P  
E  
N

-----NAME1.SRC

T  
V

-----NAME2.SRC

V

You can print the differences on the line printer using the same two source files. To do this, type

FC name1.src name2.src >PRN

## Examples:

```
FC /B test1.src test2.src >test3.src (ENTER)
```

does a binary comparison of the files Test1.src and Test2.src that are in the current directory and redirects output to the file Test3.src that is also in the current directory.

```
FC /4 /W /C B:\USER\myfile.src  
B:\USER\myfile1.src (ENTER)
```

does a line-by-line comparison of B:\USER\myfile.src and B:\USER\myfile1.src, compressing white spaces and ignoring case. After FC finds a mismatch, at least four lines in a row must match for the file to be considered matching.

## Limitations On Source Comparisons:

FC uses a large amount of memory as buffer (storage) space to hold the source files. If the source files are larger than available memory, FC compares what can be loaded into the buffer. If no lines match in the portions of the files in the buffer, FC displays only the message

```
*** Files are different ***
```

For binary files larger than available memory, FC compares both files completely, overlaying the portion in memory with the next portion from disk. This does not affect the command output.

## Errors:

If FC finds many differences (involving many lines), it only reports that the files are different. Then it stops.

If it finds no matches after the first mismatch, FC displays

```
*** Files are different ***
```

and returns to the system prompt (for example, `A>`).



FC may also display these messages:

- **Invalid parameter;***parameter*

One switch is invalid.

- **File not found;***pathname*

FC could not find the file you specified.

- **Read error in;***pathname*

FC could not read the entire file.

- **Invalid number of parameters**

You have specified the wrong number of parameters.

## FIND [/V] [/C] [/N] "*string*" [*pathname . . .*]

Searches for the specified *string* of text in one or more files, which are specified by one or more *pathnames*.

Capitalization and punctuation in the *string* must be the same as in the file. Otherwise, MS-DOS cannot find it.

FIND is a filter.

### Parameters:

If you omit the *pathname* option, MS-DOS searches for the *string* among the lines on the current screen display.

Enclose the *string* in quotes. For example, to find all occurrences of this *string*:

#### The whiteness of the whale

that are in the files Book1.txt and Book2.txt in the current directory, type

```
FIND "The whiteness of the whale" book1.txt  
book2.txt (ENTER)
```

MS-DOS displays the lines in the order in which the files are specified.

If the *string* contains quotations, enclose each quotation in double quotes. For example, to find all occurrences of this string:

#### Aye, "Moby Dick" is a whale of a story.

that are in the file Book1.txt in the current directory, type

```
FIND "Aye, " "Moby Dick" " is a whale of a story."  
Book1.txt (ENTER)
```

**/V** causes FIND to display all lines that do **not** contain the *string*. Do not use this with the **/C** parameter.

**/C** causes FIND to display only the number of lines (in each file) that contain the *string*.

**/N** causes each line to be preceded by its relative line number in the file. Do not use this with the **/C** parameter.

## Sample Use:

Using FIND with the /N parameter, you can determine which lines contain the specified *string*. Then, knowing the line numbers, you can easily enter the EDLIN Edit Line command and change every occurrence of the *string*. (See Section III, "EDLIN.")

Suppose one of your suppliers, Johnson Auto Parts, buys out another, Samuel Parts. You need to change all occurrences of Samuel Parts to Johnson Auto Parts in your B:\USER\invntory.dat file. To find the line number of each occurrence, type

```
FIND /N "Samuel Parts" B:\USER\invntory.dat
(ENTER)
```

MS-DOS might display a list like this:

```
----- B:\user\invntory.dat
[1]spark plugs           200 6-16-83 Samuel Parts
[5]distributor caps     50 6-16-83 Samuel Parts
[9]14" rad hose (2" dia.) 25 6-16-83 Samuel Parts
```

Now you can use EDLIN to edit Lines 1, 5, and 9 as needed.

Examples:

```
FIND /C "-" B:\USER\profits.dat
```

displays the number of times a negative sign occurs in B:\USER\Profits.dat. In this way, you can see at a glance how many negative numbers you have in the file. (Notice the space after the negative sign. This is included so MS-DOS does not find hyphenated words.)

```
DIR B: | FIND /V "DAT"
```

displays all names of the files in the home directory of the disk in Drive B that do not contain the string DAT.

## Errors:

When MS-DOS finds an error, it displays one of the following error messages:

- **FIND: Invalid number of parameters**

You did not specify a string.

- **FIND: Syntax error**

You typed an illegal string.

- **FIND: File not found *filename***

FIND cannot find the specified filename.

- **FIND: Read error in *filename***

An error occurred when FIND tried to read the file.

- **FIND: Invalid parameter *parameter***

You specified a parameter that does not exist.

**FOR %f IN (*set*) DO *command* %f**  
**(regular MS-DOS command)**

**FOR %%f IN (*set*) DO *command* %%f**  
**(batch file command)**

Executes the specified *command*, sequentially, for each item in the *set*.

## Parameters:

*set* may be a list of items, each of which is separated with a space, or it may be one wild card item. Files, directories, and disk drives are only a few of the items you may list.

**%f** is the variable that MS-DOS uses to represent each item in the *set*, one at a time. DO tells MS-DOS to do the command on %f (each item).

**%%f** is the same as %f, except the second per cent sign is required because the command is in a batch file.

## Sample Uses:

Use FOR whenever you want to execute a command for several items, so you don't repeat the command unnecessarily. For example, at the system level, you may want to see the directory listing for three directories: A:\, B:\, and B:\USER. To do so, type

```
FOR %f IN (A:\ B:\ B:\USER) DO DIR %f (ENTER)
```

MS-DOS displays the listing for each directory, in order.

Suppose you want to be able to substitute other directories for those in the command above. You must use replaceable parameters. Therefore, your FOR command must be in a batch file. Create such a file by typing

```
COPY con 3dir.bat (ENTER)  
REM This file displays directory listings for %1, %2,  
and %3 (ENTER)  
REM It is called 3dir.bat (ENTER)  
FOR %%f IN (%1 %2 %3) DO DIR %%f (ENTER)  
(F6) (ENTER)
```

Suppose you want to see the listings for A:\BIN, A:\GAMES, and A:\ACCTS. To execute the batch file, type

**3dir A:\BIN A:\GAMES A:\ACCTS **ENTER****

MS-DOS first substitutes the directory names, in order, for the replaceable parameters %1, %2, and %3. Then, it does a DIR command for each directory.

### **Examples:**

**FOR %f IN (\*.asm) DO TYPE %f **ENTER****

displays all files in the current directory that have the extension .asm.

**FOR %f IN (taxfile autofile homefile) DO DEL %f**

deletes the three files, in order, from the current directory.

### FORMAT [*drive*] [/S] [/V] [/P]

(FLOPPY DISK ONLY) Prepares the blank floppy disk in the specified *drive* for use. FORMAT does this by defining the tracks and sectors and writing system information onto the disk.

**Note:** To prepare a hard disk, you must use the HFORMAT command.

You can format either a blank or already formatted disk. If the disk already is formatted, you lose all information when you reformat it.

Immediately after you enter the FORMAT command, FORMAT displays

Insert new diskette for drive A:  
and strike any key when ready

Note that FORMAT prompts you to insert the disk. If the disk is already in the drive, simply press

**(SPACEBAR)**. FORMAT displays

#### Formatting tracks

---

Each dash in the dashed line represents a “track,” an area on the disk. As the track is formatted, MS-DOS replaces the dash with a period (.). A question mark (?) in place of a period indicates that the corresponding track contains flawed “sectors.” This is no problem, however, because FORMAT locks out flawed sectors so that MS-DOS never writes to them.

When the format is complete, MS-DOS displays this message:

*nnnnnn* bytes total disk space  
*nnnnnn* bytes available on disk

Format another (Y/N)?

Press **Y** **ENTER** to format another disk in the same drive. Press **N** **ENTER** if you don't want to format another disk.

## Parameters:

**drive** can be either Drive A or B.

If the *drive* is Drive A, MS-DOS prompts:

Insert new diskette for drive A  
and strike any key when ready.

Remove the system disk from Drive A and insert the disk to be formatted. Press **SPACEBAR** to begin formatting.

If you omit the *drive* specification, MS-DOS formats the disk in the current drive.

**/S** causes FORMAT to copy the system files (IO.SYS, MSDOS.SYS, and COMMAND.COM) to the disk after it is formatted. This makes the newly formatted disk a system disk.

**/V** causes FORMAT to prompt for a volume label after the disk is formatted. If you use this parameter, you may enter a label of up to 11 characters, or you may press **ENTER** to bypass the prompt. Entering a volume label helps keep track of your disks, the same as if you affix real labels to them.

**/P** causes FORMAT to prompt for the “skew” and “interleave” factors, which together determine the order in which MS-DOS is to access disk sectors. If you omit the **/P** parameter, disk sectors are accessed in order from 1 to 9.

If you are not an advanced user, you probably are not familiar with skew and interleave factors. If not, we recommend that you do not use the **/P** parameter.



## Sample Use:

Before you can store information on a new disk, you must prepare it using `FORMAT` (unless you format it while using the `COMPDUPE` command). After formatting, record the volume label and date of creation. Store this information in a safe place. It helps you estimate how long a diskette has been in use.

Suppose you have an old disk you want to “start over” with. First, do a `DIR` command to make sure you don’t need anything on the disk. Then you can use `FORMAT` to erase all old information and lock out any flawed sectors that have developed. `FORMAT` puts the system information back on the disk and leaves the “good” sectors available for information storage.

## Examples:

`FORMAT` **(ENTER)**

formats the disk in the current drive.

`FORMAT A:` **(ENTER)**

formats the disk in Drive A.

`FORMAT B: /S /V` **(ENTER)**

formats the disk in Drive B, makes it a system disk, and asks you for a volume label. You may enter a label of up to 11 characters (for example, `DISKONE`), or press **(ENTER)** if you don’t want to label the disk.

`FORMAT /P` **(ENTER)**

formats the disk in the current drive, prompting you for the new skew and interleave factors.

## Errors:

The following errors cause MS-DOS to abort the `FORMAT` command and return to the system prompt.

- **Error writing boot sector to destination**

All floppy disks — both data and system disks — must have a boot sector so that they may be used by the system.

- **After format, one of system sectors could not be read**

The system sectors (boot, file allocation table, and directory) are required for the disk to be useful.

- **Errors writing to the system sectors, cannot continue**

The system sectors (boot, file allocation table, and directory) are required for the disk to be useful.

## GOTO *label*

Transfers control to the line (in the batch file) that follows the one that contains *:label*.

GOTO is a batch file command.

### Parameters:

*label* is a character string. MS-DOS considers only the first eight characters; it ignores the rest.

When a batch file executes, its *labels* are not displayed. Therefore, you can use *labels* to include comments in a batch file.

### Sample Use:

Use GOTO with the IF command to direct execution to particular subroutines when particular conditions exist. (See the IF command also.)

Using GOTO and IF, you can create a batch file that copies a specified source file to a specified target file only if the target does not already exist. If the target exists, the file pauses so you can abort the copy.

Create the file by typing

```
COPY con chekdest.bat (ENTER)
REM Checks to see if file exists before copying to it
  (ENTER)
REM When executing, substitute the drive for %1
  and the directory for %2 (ENTER)
REM When executing, substitute the source file for
  %3, the target for %4 (ENTER)
%1 (ENTER)
CHDIR %2 (ENTER)
IF NOT EXIST %4 GOTO G (ENTER)
TYPE %4 (ENTER)
PAUSE (ENTER)
:G (ENTER)
COPY %3 %4 (ENTER)
:END (ENTER)
(F6) (ENTER)
```

Suppose that Newfile.asm exists in the root directory of Drive B, and you execute the batch file by typing:

```
chekdest B: \ oldfile.asm newfile.asm (ENTER)
```

As instructed in the command line, the batch file replaces %1 with B:, making Drive B the current drive. It then replaces %2 with \, making the root directory the current directory. Finally, it replaces %3 with Oldfile.asm and %4 with Newfile.asm.

The batch file checks to see if Newfile.asm exists in the current directory. It does; therefore, the batch file uses the TYPE command to display Newfile.asm. Then it pauses. If you want to copy over Newfile.asm, press **(SPACEBAR)**. If not, press **(CTRL) (C)**.

If Newfile.asm does not exist in the directory, the batch file “goes to” the line following :G. Therefore, it does the copy automatically, without pausing.

### Example:

```
:G  
REM looping . . .  
GOTO G
```

produces an infinite sequence of messages:

```
REM looping . . . .  
GOTO G
```

### Error:

If *:label* is not in the batch file, MS-DOS returns the error

```
Label not found
```

## HFORMAT [*drive*][*/S*][*/V*][*/P*][*/B*]

(HARD DISK ONLY) Prepares a hard disk for use and, optionally, makes it an operating system disk by writing the system files to it.

The hard disk may be either blank or already formatted when you enter the HFORMAT command. If the disk is already formatted, all information on it is erased when you reformat.

Immediately after you enter the HFORMAT command, HFORMAT displays a screen similar to the following:

Press any key to begin formatting *drive*

Press **SPACEBAR**. HFORMAT displays

Formatting cylinders

```
-----  
-----  
-----  
-----  
-----
```

When the format is done without error, each dash, which represents an area on the disk, becomes a period. A question mark in place of a period indicates that a portion of the diskette contains flawed areas. HFORMAT locks out the flawed areas so that MS-DOS never writes to them.

When the format is complete, MS-DOS displays this message:

```
nnnnnnnn bytes total disk space  
nnnnnnnn bytes in bad sectors  
nnnnnnnn bytes available on disk
```

### Parameters:

*drive* is a drive specification of C: or greater. If you omit the *drive*, HFORMAT uses Drive C.

**/S** causes HFORMAT to copy the system files (IO.SYS, MSDOS.SYS, and COMMAND.COM) to the disk after it is formatted. This makes the newly formatted disk a system disk.

**/V** causes HFORMAT to prompt for a volume label after the disk is formatted. If you use this parameter, you may enter a label of up to 11 characters, or you may press **(ENTER)** to bypass the prompt.

**/P** causes HFORMAT to prompt for the “interleave” factor and the number of cylinders and heads. If you are not an advanced user, you probably are not familiar with interleave factors, cylinders and heads. If not, we recommend that you do not use the **/P** parameter.

**/B** causes HFORMAT to prompt for information it needs to “lock out” flawed areas on the hard disk so that MS-DOS never writes to them. (Having a small number of flawed areas is not uncommon.)

Although using **/B** is optional, we strongly recommend that you do it. When you do, HFORMAT repeatedly asks you to:

Enter next head, track pair or press <ENTER> to quit.

At this prompt, enter information from the “Media Error Map” that is provided with the computer. For example, if the map indicates flaws on Track (Cylinder) 123, Head 02 and Track 312, Head 03, type the numbers and **(ENTER)** as follows:

Enter next head, track pair or press <ENTER> to quit. 2,123 **(ENTER)**

Enter next head, track pair or press <ENTER> to quit. 3,312 **(ENTER)**

Enter next head, track pair or press <ENTER> to quit. **(ENTER)**

If the map is blank, simply press **(ENTER)**.

## Sample Use:

Before you can store information on a hard disk, you must prepare it using HFORMAT. During formatting, you have the option of transferring the system files to the hard disk, making it a system disk. The system files cannot be transferred using the COPY command. Therefore, you'll probably want to use HFORMAT /S to format your hard disk, and then type **COPY A:\*.\*C: (ENTER)** to copy all other files from your system diskette to your hard disk.

**Note:** Your MS-DOS System Diskette contains a special batch file, CONFIGHD.BAT, which automatically formats the hard disk and copies to it all files — system and non-system — that are on the diskette. If you want the option of **not** copying the non-system files, use the HFORMAT command. If you don't care if all files are copied, simply type:

```
CONFIGHD (ENTER)
```

## Example:

```
HFORMAT /S (ENTER)
```

formats Drive C and transfers the system files to it. After removing the Drive A diskette and resetting your computer, you can start up your system under hard disk control. From now on, the system prompt is C> (as long as a diskette is not in Drive A whenever you start up or reset your system).

# Notes:





**IF [NOT] *condition* command**

Allows conditional execution of commands in batch file processing. When the *condition* is true, then the *command* is executed. When it is false, the *command* is ignored.

**Parameters:**

**NOT** changes the IF command so that the *command* executes only when the *condition* is false.

The *conditions* are:

**ERRORLEVEL *number***, which indicates that the *command* is to execute only if the program previously executed by COMMAND has an exit code of *number* or higher.

***string1* = *string2***, which indicates that the *command* is to execute only if *string1* and *string2* are identical after parameter substitution. Strings may not have embedded separators.

**EXIST *filename***, which indicates that the *command* is to execute only if the file specified by *filename* exists.

**Sample Use:**

Use IF with the GOTO command to direct execution to particular subroutines when particular conditions exist. (See the GOTO command also.)

Suppose you have three programs (Myprog, Samprog, and Salprog). Each requires that you be in a separate directory. You can create a batch file to put you in the proper directory. To do so, type

```
COPY con progdir.bat (ENTER)
REM This file changes the dir for Myprog, Samprog,
  or Salprog (ENTER)
REM When executing, substitute the drive for %1,
  the program name for %2 (ENTER)
%1 (ENTER)
IF %2 = = Myprog GOTO W (ENTER)
IF %2 = = Samprog GOTO X (ENTER)
IF %2 = = Salprog GOTO Y (ENTER)
```

```
:W (ENTER)
CHDIR \USER\MYPROG (ENTER)
GOTO :Z (ENTER)
:X (ENTER)
CHDIR \USER\SAMPROG (ENTER)
GOTO :Z (ENTER)
:Y (ENTER)
CHDIR \USER\SALPROG (ENTER)
:Z (ENTER)
CHDIR (ENTER)
:END (ENTER)
(F6) (ENTER)
```

Execute the batch file by entering its filename, followed by the drive and program name. For example, to change to the correct directory for Myprog, type

```
progdir B: Myprog (ENTER)
```

The batch file substitutes B: for %1, making Drive B your current drive. Then it substitutes Myprog for %2. It goes to label :W and changes the directory to \USER\MYPROG. Then it goes to :Z. The line following :Z causes CHDIR to display the current directory to verify that it is correct.

## Examples:

```
IF NOT EXIST Myfile ECHO Can't find file (ENTER)
```

displays "Can't find file" if the file Myfile doesn't exist.

```
IF EXIST All.lst GOTO G
```

sends program execution to the line following :G, if All.lst exists.

**MKDIR** *pathname*  
**MD** *pathname*

Makes a new directory.

The MKDIR command is used to create a hierarchical directory structure.

## Parameter:

***pathname*** tells MS-DOS under which directory to create the new directory and specifies the name to give it. The *pathname* may be either relative or absolute.

## Sample Uses:

Using MKDIR, you can better organize your files. For example, you can group files according to user or use.

You can also use MKDIR to create a directory in which to put your external commands. To do so, type

```
MKDIR \BIN (ENTER)
```

MS-DOS creates the directory \BIN in the current drive.

Use COPY to copy your command files into this directory, then use PATH to tell MS-DOS where to search for the commands.

If you write an application program, you may want to include a MKDIR command so the program creates a directory the first time you run it. By following MKDIR with a CHDIR (change directory) command, you automatically enter the proper directory whenever you run the program.

## Examples:

```
MKDIR \USER (ENTER)
```

creates the subdirectory \USER in your root directory.

```
MD \USER\JOE (ENTER)
```

creates the subdirectory \USER\JOE in your \USER directory.

**MD LETTERS (ENTER)**

creates the subdirectory LETTERS under the current directory. If the current directory is \USER\JOE, MS-DOS creates the directory USER\JOE\LETTERS.



**MD B:LETTERS (ENTER)**

creates the subdirectory LETTERS under the home directory of Drive B. If the home directory is \USER\JAN, MS-DOS creates the directory \USER\JAN\LETTERS.



## MORE

Reads from standard input (such as a command from your keyboard) and displays one screen of information at a time. It then pauses and displays the message **—MORE—** at the bottom of your screen.

Press **(ENTER)** to display another screen of information. This process continues until all the input data has been read.

MORE is a filter.

### Sample Use:

Use MORE to view long files. Suppose the file Myfiles.com, is on the current directory. You want to display it one screen at a time. To do so, type


TYPE myfiles.com | MORE **(ENTER)**

### Example:

TYPE B:acctspay.dat | MORE **(ENTER)**

displays the file Acctspay.dat that is on the home directory of Drive B, one screen at a time. MS-DOS displays the message **—MORE—** at the bottom of each screen. Press **(ENTER)** to continue.

## PATH [*pathname*[:*pathname*] . . . ]


Sets a command path, which tells MS-DOS the directories or drives in which to search for external commands. MS-DOS always searches the current directory before it searches the paths set by PATH. You can also use PATH to remind you of the current path. 

Unless you reset the path or turn off the system, MS-DOS searches the specified *path(s)* each time you use an external command.

### Parameters:

*pathname* can specify either a directory or an entire drive.


If you don't include a *pathname*, PATH displays the current path setting. This serves to remind you which path MS-DOS is searching. If no path is set (MS-DOS is searching only the current directory), PATH displays the message No path.

If you specify PATH ; (PATH followed by a semi-colon), MS-DOS sets the NUL path. It searches the current directory only. 

### Sample Use:

PATH gives you the option of using commands that are not in the current directory. Suppose the root directory in Drive A contains so many files it is difficult to use efficiently. To save space, you create the directory A:\BIN (using MKDIR) and put your commands there (using COPY). Then you remove your commands from A:\ (using DEL).

Suppose now that you start up MS-DOS and immediately want to format a disk in Drive B. Because FORMAT is an external command, you must do either of the following:

- Use CHDIR to make A:\BIN your current directory
- Use PATH to make MS-DOS search A:\BIN for external commands 

To avoid encountering the problem repeatedly, use PATH. You can specify A:\BIN only, or you can specify all directories on Drive A, separating them with semicolons. To specify A:\BIN only, type:

PATH A:\BIN (ENTER)

Type PATH (ENTER) to verify the path setting.

## Examples:

PATH \BIN\USER\JOE (ENTER)

tells MS-DOS to search \BIN\USER\JOE in the current drive for external commands (after it searches the current directory).

PATH \BIN\USER\JOE;\BIN\USER\SUE;  
  \BIN\DEV (ENTER)

tells MS-DOS to search the directories specified by the above pathnames for external commands. MS-DOS searches the current directory and then those in the command, in the order in which they are listed.

PATH ; (ENTER)

tells MS-DOS to search the current directory only.

PATH (ENTER)

displays the current path setting.

**PAUSE [*comment*]**

Suspends execution of the batch file and displays the message

**Strike a key when ready . . .**

At this time, you may press **(SPACEBAR)** to continue execution, or you may press **(CTRL) (C)** to display

**Abort batch job (Y/N)?**

If you press **(Y)**, execution aborts and control returns to MS-DOS. If you press **(N)**, execution continues.

**Parameters:**

***comment*** is a message to be displayed when the file pauses. The file displays it before the message “**Strike a key when ready . . .**”

**Sample Uses:**

During the execution of the batch file, you may need to perform some action before executing the next command. For example, you may need to change disks or make sure your printer is ready.

Whenever this is the case, include a PAUSE command where necessary. For example, suppose you want to list two files that are on different disks. Create a batch file by typing

```
COPY con Typfiles.bat (ENTER)
REM This file types the files Rental.dat and Sales.dat
(ENTER)
REM It is called Typfiles.bat (ENTER)
TYPE B:\rental.dat (ENTER)
PAUSE (ENTER)
TYPE B:\sales.dat (ENTER)
(F6) (ENTER)
```

Typfiles.bat displays the file Rental.dat, then pauses and displays the message

**Strike a key when ready . . .**

Change disks; then press **(SPACEBAR)** to display the fi  
Sales.dat.



You should also use PAUSE whenever you want the option to abort execution of the batch file.

Suppose you create a batch file that combines all current directory files that have the extension .lst into the file all.lst. If the file All.lst already exists, you may not want to execute the copy. If you do so, the batch file destroys the original contents of All.lst.

To avoid destroying All.lst, create the batch file by typing

```
Copy con Comblst.bat (ENTER)
REM This file combines *.lst files into All.lst (ENTER)
IF EXIST All.lst PAUSE All.lst already exists. Press
  (CTRL) (C) to abort, or (ENTER)
PAUSE (ENTER)
COPY *.lst all.lst (ENTER)
(F6) (ENTER)
```

Suppose All.lst exists and you don't know it. When you execute the file, MS-DOS pauses and displays the message

All.lst already exists. Press (CTRL) (C) to abort or  
Strike a key when ready . . .

If All.lst does not exist, MS-DOS pauses and displays only

Strike a key when ready . . .

## Examples:

PAUSE Press (CTRL) (C) if unsure or (ENTER)

pauses execution and displays the message

Press (CTRL) (C) if unsure or Strike a key when  
ready . . .

PAUSE Insert disk to check in Drive B —

pauses execution and displays the message

Insert disk to check in Drive B — Strike a key when  
ready . . .

---

## PRINT [*pathname*[/T][/C][/P]] . . .

Lets you put up to 10 files in the print queue, so that MS-DOS prints them automatically while you process other MS-DOS commands. This is called "background printing."

The printer must be connected and ready.

### Parameters:

**/T** deletes (terminates) from the print queue all files that are there, waiting to be printed.

**/C** deletes (cancels) from the print queue the file that immediately precedes and all files that follow /C in the command line.

**/P** adds to the print queue (prints) the file that immediately precedes and all files that follow /P in the command line.

Regardless of the number of /C and /P parameters your command includes, each switch always affects the file immediately preceding it. For example, if you type this:

```
PRINT budget /P sales rentals /C (ENTER)
```

MS-DOS adds the files Budget and Sales, but cancels the file Rentals.

If you omit all parameters, PRINT displays the contents of the print queue.

### Sample Use:

Whenever you must print for a long time and must use your computer at the same time, use PRINT. The /C and /P parameters let you revise the print queue whenever doing so is most convenient. For example, you may queue these 10 files:

```
Jan  
Feb  
March  
April  
May  
June  
July
```

Aug  
Sept  
Oct

by typing

```
PRINT jan feb march april may june july aug sept oct  
(ENTER)
```

By the time you finish working on something else, the Jan and Feb files are printed. Because these are no longer in the queue, there is room now to add the Nov and Dec files. In the meantime, you have changed your mind about printing Aug. To update the print queue accordingly, type

```
PRINT nov /P dec aug /C (ENTER)
```

The file March is being printed, and the queue contains these files:

April  
May  
June  
July  
Sept  
Oct  
Nov  
Dec

## Examples:

```
PRINT /T (ENTER)
```

empties the print queue.

```
PRINT A:temp1.tst /C A:temp2.tst A:temp3.tst (ENTER)
```

removes the files A:Temp1.tst, A:Temp2.tst, and A:Temp3.tst from the print queue.

```
PRINT temp1.tst /C temp2.tst /P temp3.tst (ENTER)
```

removes the file Temp1.tst from the queue and adds the files Temp2.tst and Temp3.tst.

## Errors:

If it finds an error, PRINT displays one of the following error messages:

### Name of list device [PRN:]

MS-DOS displays this prompt the first time you run PRINT. You may enter any current device as the PRINT output device. If you press **(ENTER)** only, the printer becomes the device.

### List output is not assigned to a device

The device specified as the PRINT output device is invalid.

### PRINT queue is full

The queue can contain no more than 10 files.

### PRINT queue is empty

There are no files in the print queue.

### No files match *pathname*

The files you tried to add to the queue do not exist. (Note that MS-DOS displays no message if you try to cancel files that are not in the queue.)

### Drive not ready

If this message occurs when PRINT attempts a disk access, PRINT keeps trying until the drive is ready. Any other error causes PRINT to cancel the current file. In such a case, PRINT outputs an error message to the printer.

PRINT may output either of the following messages to the printer. They serve only to remind you that the printout is incomplete.

### All files canceled by operator

You used the /T parameter to cancel the printing of all files in the print queue.

### File canceled by operator

You used the /C parameter to cancel the current file in the print queue.

## PROMPT [*prompt-text*]

Changes the MS-DOS system prompt to *prompt-text*.

### Parameters:

***prompt-text*** is a string of characters to set as the prompt. It can be any of the following:

- A string of characters, such as your name
- A special prompt in the format *\$character* where *character* is one of those in the chart below.
- A combination of special prompts or of a string and special prompt(s)

Character	Prompt
\$	The \$ character
t	The current time
d	The current date
p	The current directory
v	The MS-DOS version number
n	The current drive specification
g	The > symbol
l	The < symbol
b	The   symbol
-	A carriage return and line feed
s	A leading space
h	A backspace
e	The escape sequence

If you omit *prompt-text*, MS-DOS sets the current drive specification as the prompt.

### Sample Uses:

There are several reasons for using a special system prompt.

For example, by setting the current directory as the prompt, you need not enter the CHDIR command to remind you which directory you're in. To set the directory as the prompt, type

```
PROMPT $p (ENTER)
```

By setting the time as the prompt, you can check the time without entering the TIME command. In this way, you can also time the execution of commands and programs. To set the time as the prompt, type

PROMPT \$t (ENTER)

Notice how you can use `$\n` to insert a carriage return and line feed in your prompt:

PROMPT Time = \$t\$.Date = \$d (ENTER)

In this case, the new system prompt is similar to the following:

```
Time = 0:07:04.07
Date = Thu 11-15-1984
```

Because your computer has an ANSI escape sequence driver (ANSI.SYS) you can use escape sequences in your prompts, if the ANSI device driver is configured into your system. (See Appendix C.) For example,

PROMPT \$e[7m\$N:\$e[m (ENTER)

sets the prompts in reverse video mode and returns to video mode for other text.

## Examples:

PROMPT \$n\$g (ENTER)

sets the current drive, followed by a greater-than sign (>) as the prompt. For example, when you change to Drive B, the prompt changes to `B>`. When you receive your system, PROMPT is set to `$n$g`.

PROMPT A\$g (ENTER)

sets the prompt `A>`. Regardless of which drive you're in, this is the prompt.

PROMPT \$p

sets the current directory, including the current drive, as the prompt.

**Note:** As you can see from the `A$g` example, some prompts hinder, rather than help. Use PROMPT carefully.

## RECOVER *pathname* RECOVER *drive*

Recovers (1) a file that contains bad sectors, or (2) all files on a disk that contains bad sectors in its directory.

In the first case, MS-DOS reads the file sector-by-sector. It marks the bad sectors in a system table, so the data is never again allocated to them.

In the second case, MS-DOS scans the disk file allocation table (FAT) for chains of allocation units. It creates a new root directory for each chain. Use the RECOVER *drive* command only if the disk's directory is unusable.

If there is not enough room in the root directory, RECOVER displays a message to this effect. It then stores information about the extra files in the File Allocation Table. When there is enough room, you can use RECOVER again to regain the files.

### Parameters:

*pathname* specifies the file to recover. You must include the *filename* part of the *pathname*.

*drive* specifies the disk to recover.

### Sample Use:

If you have trouble storing and manipulating information, the disk may have a flawed sector. Running CHKDSK (check disk) should indicate whether this is the case. If the file that contains the flawed sector is a text file, recover the file. This saves all information except that in the flawed sector. Re-enter the lost information. If the file is a data file, recovering the file may or may not help.

**Note:** If you are a beginner, you may prefer re-entering all information to trying to recover a file.

### Examples:

```
RECOVER \USER\SAM\pamphlet.txt (ENTER)
```

recovers the file \USER\SAM\Pamphlet.txt that is on the current disk.



**RECOVER oldbook.txt **(ENTER)****

recovers the file Oldbook.txt that is in the current directory.

**RECOVER B: **(ENTER)****

recovers the disk in Drive B, if the bad sectors are in the directory.

## REM [*remark*]

Lets you include the specified *remark* in a batch file.

### Parameter:

A *remark* is a line the batch file displays (during execution) but does not try to execute. It displays the *remark* only if ECHO is on.

The space, tab, and comma are the only separators allowed in the *remark*.

### Sample Uses:

Use REM as needed to do the following:

- Remind you of the file's name and use
- Keep track of what a particular command is doing
- Include warnings in the file

In this file, the first remark reminds you to replace parameters %1, %2, and %3 when executing the file. The second remark simply reminds you of the file's name.

```
COPY con 3dir.bat (ENTER)
REM This file displays directory listings for %1, %2,
and %3 (ENTER)
REM It is called 3dir.bat (ENTER)
FOR %%f IN (%1 %2 %3) DO DIR %%f (ENTER)
(F6) (ENTER)
```

### Examples:

```
REM This file is called Billfile.bat (ENTER)
```

displays the following message, if ECHO is on:

```
REM This file is called Billfile.bat
```

```
REM pathname1 replaces %1, pathname2 replaces
%2
```

displays the following message, if ECHO is on:

```
REM pathname1 replaces %1, pathname2 replaces
%2
```

## **REN *pathname filename***

Changes the name of the file specified by *pathname* to *filename*.

### **Parameters:**

If you include a wild card in the *filename*, MS-DOS leaves the corresponding characters as they were. For example, if you type the following:

```
REN newfile old???? (ENTER)
```

MS-DOS changes the name of the file Newfile that is in the current directory to Oldfile.

If you include a wildcard in the *pathname*, also, MS-DOS renames all files in the specified directory that match the *pathname*. For example, if you type the following:

```
REN *.lst *.prn (ENTER)
```

MS-DOS changes only the extension of all .lst files that are in the current directory. The new extension is .prn. The file Suefile.lst, for example, becomes Suefile.prn.

### **Sample Uses:**

There are several reasons to rename a file. Here are a few:

- To indicate the owner of the file
- To indicate that one file is newer than a similar file
- To make it easier to do operations on several files of the same type
- To correct an error you made in the name

For example, suppose your current directory contains two similar files, Oldfile and Newfile, and you plan to create a third. If you don't need Oldfile, delete it by typing

```
DEL oldfile (ENTER)
```

Then change the name of Newfile to Oldfile by typing

```
REN newfile oldfile (ENTER)
```

Use EDLIN to create the newest file, naming it Newfile (see Section III, "EDLIN"). Now you can easily tell, from the filenames, which data is new.

Suppose you have several files that have the extension .lst and several that have the extension .prn. You often do the same operations on both sets of files (example: DIR \*.lst and DIR \*.prn). To avoid this, rename one set by typing

```
REN *.lst *.prn (ENTER)
```

Now you need only this command to see the listing:

```
DIR *.prn (ENTER)
```

**Note:** Perhaps the most practical use of RENAME is to rename a file that contains a flawed sector. If a file is flawed, you can use RECOVER to save all information except that in the flawed sector. To prevent MS-DOS from writing any new files to that area, however, you should keep the flawed file there. Rename the file to Badspot to remind you of the flaw.

## **Examples:**

```
REN B:\USER\GL1.dat GL2.dat (ENTER)
```

changes the name of the GL1.dat file that is in B:\USER to GL2.dat.

```
REN B:Mufile ?y???? (ENTER)
```

changes the name of Mufile, a file in the home directory in Drive B, to Myfile.

**Error:**

If you try to give a file a name that already exists, MS-DOS displays the following error message:

**Duplicate file name or File not found**

For example, suppose your current directory contains the files Myfile.lst and Myfile.prn, in that order. If you use this command:

```
REN Myfile.* Suefile.prn (ENTER)
```

MS-DOS changes the name of Myfile.lst to Suefile.prn. Then because Suefile.prn now exists, it does not rename Myfile.prn. Instead, it displays the error message.

## RESTORE *drive* [*pathname*]/S[/P]

(HARD DISK ONLY) Restores one or more files from diskettes to a hard disk. RESTORE copies only files that were stored on diskette using the BACKUP command.

**Note:** RESTORE works best if BUFFERS = 5 (or greater) in the CONFIG.SYS file. (See Appendix C.)

### Parameters:

***drive*** specifies the drive containing the backup files.

***pathname*** specifies the hard disk file you want to restore.

**/S** causes RESTORE to restore all files in the specified directory and all directories below it.

**/P** causes RESTORE to prompt you before restoring files that have been changed since the last backup and all "read-only" files.

### Sample Use:

As long as you have a complete library of backup diskettes, restoring your hard disk should be straightforward. Suppose you want to restore all the files that were backed up from all directories on Drive C. Insert the first backup diskette into Drive A; then type

```
RESTORE A: C:\S (ENTER)
```

**Note:** If you have several files to restore — but not **all** files, as in the previous example — you may want to create a batch file consisting of the necessary backup commands. Then you can do all the backups simply by running the batch file.

### Examples:

```
RESTORE A: C:*.dat/P (ENTER)
```

restores all files from Drive A that have the extension .dat and that were backed up from the current directory of Drive C.

```
RESTORE A: C:\USER\store1.dat (ENTER)
```

restores from Drive A the file Store1.dat that was backed up from the USER directory of Drive C.

# RMDIR (Remove Directory)

Internal

**RMDIR** *pathname*

**RD** *pathname*

Removes from the disk the sub-directory specified by *pathname*.

**Note:** You cannot remove the root directory or the current directory.

The directory must be empty except for the . and .. symbols.

Before you try to remove a directory, use the DIR command to see what files are in the directory. If you may need a file, use the COPY command to move it to another directory. Then use the DEL command to delete all files you don't need.

## Sample Use:

Whenever you delete or copy all of a directory's files, you may want to remove the directory. Doing so saves valuable disk space.

Suppose a sales representative, Ann, retires, and you want to divide her accounts between Sue and Jim. If \USER\ANN\ACCTS is not your current directory, use a CHDIR command to make it so. Now copy each account file to either \USER\SUE\ACCTS or \USER\JIM\ACCTS, using a command similar to the following:

```
COPY drugstor.dat B:\USER\SUE (ENTER)
```

**Note:** Changing directories to \USER\ANN\ACCTS saves you from entering a complete pathname for the source file in each COPY command.

You are almost ready to remove \USER\ANN\ACCTS. Because you can't remove the current directory, however, you must first change directories. Type

```
CHDIR .. (ENTER)
```

MS-DOS puts you in \USER\ANN. Remove the directory, using either an absolute pathname

**RMDIR \USER\ANN\ACCTS (ENTER)**

or a relative pathname

**RMDIR ACCTS (ENTER)**

### **Examples:**

**RMDIR \BIN\USER\JIM (ENTER)**

removes the subdirectory \BIN\USER\JIM from the current disk.

**RMDIR MEMOS**

removes the subdirectory MEMOS from the current directory.

**RMDIR B:MEMOS**

removes the subdirectory MEMOS from the home directory in Drive B.



---

**SET [*parameter* = [*replacement parameter*]]**

Sets the *parameter* equal to the *replacement parameter* for use in later programs and batch files (until you unset the *parameter* or turn off the computer).

SET does not affect commands at the system level, other than those in batch files.

**Parameters:**

***parameter*** can be any character string, except the numbers 0 through 9. You may want to use general terms (such as drive, pathname, program name, filename, and text), or you may want to save space by using only a character or a few characters. You may include imbedded separators in the *parameter*.

***replacement parameter*** can also be any character string, except the numbers 0 through 9. Here, however, use specific parameters (such as B:, \USER, Myprog, Myfile, and This is the real text). If you omit the *replacement parameter*, MS-DOS voids the current setting of the *parameter*.

If you omit *parameter* = *replacement parameter*, SET returns the values that are set.

**Sample Use:**

The greatest advantage of the SET command is this: If you have several batch files that require the same *replacement parameter*, you don't need to substitute it repeatedly.

Suppose you have several batch files that contain the *parameter* %filename%. One might contain the command

```
TYPE %filename% (ENTER)
```

Another might contain the command

```
COPY A:%filename% B: (ENTER)
```

(Note: Within the batch file commands, the *parameter* is enclosed by per cent signs. Within the SET command, it is not.)

The first time you run all the batch files, you may want to substitute Myfile for %filename%. To do this, use the SET command before you begin. Type

```
SET filename = Myfile (ENTER)
```

MS-DOS automatically substitutes each occurrence of %filename% — in all batch files — with Myfile. Now suppose you want to substitute Samfile for %filename%. Void the current setting by typing

```
SET filename = (ENTER)
```

Then reset the parameter by typing

```
SET filename = Samfile (ENTER)
```

The SET command lets you write versatile batch files. Here is an example. As you may already know, the PROMPT command lets you set your own system prompt. (The default prompt is the current drive followed by a greater-than sign, such as A>.) Suppose you have four system prompts, all of which you use often. Rather than repeatedly entering the PROMPT command with parameters, use batch files to do the work for you.

First, assign each prompt a unique *parameter* that is to represent it in a batch file.

- promptd = default prompt (current drive)
- prompt1 = time and date
- prompt2 = current directory
- prompt3 = version number and a greater-than symbol

Then set each *parameter* equal to its *replacement parameter* (the code required by the PROMPT command), by typing

```
SET promptd = $n$g (ENTER)
```

```
SET prompt1 = The time is $t$.The date is  
$d (ENTER)
```

```
SET prompt2 = $p (ENTER)
```

SET prompt3 = \$v\$g (ENTER)

Now create four batch files, each containing a PROMPT command, as follows:

COPY con promptd.bat (ENTER)

REM This file changes the system prompt to the default (current directory) (ENTER)

PROMPT %promptd%

(F6) (ENTER)

COPY con prompt1.bat (ENTER)

REM This file changes the system prompt to the current time and date (ENTER)

PROMPT %prompt1%

(F6) (ENTER)

COPY con prompt2.bat (ENTER)

REM This file changes the system prompt to the current directory (ENTER)

PROMPT %prompt2% (ENTER)

(F6) (ENTER)

COPY con prompt3.bat (ENTER)

REM This file changes the system prompt to the version number and greater-than symbol (ENTER)

PROMPT %prompt3% (ENTER)

(F6) (ENTER)

Now you can quickly change your prompt to any of the four, simply by executing the appropriate batch file. For example, execute prompt1.bat by typing

prompt1 (ENTER)

The file replaces the parameter *prompt1* with \$n\$g and changes the system prompt accordingly.

If you want to use prompts other than those set, you can either set them or change an existing setting. Remember, the settings continue to affect these files and other batch files until you unset them or reset or turn off the computer.

**Note to Advanced Users:** You can use SET to affect application programs, as well as batch files. The *MS-DOS Programmer's Reference Manual*, Catalog #26-5403, tells you what you need to know about "Program Segment Prefixes" to be able to do this. The manual is available at your Radio Shack Computer Center.

**Examples:**

SET drive = B: (ENTER)

sets B: to replace the parameter *drive* in later programs.

SET drive = (ENTER)

voids the current setting for the parameter *drive*.

SET (ENTER)

displays the current settings for COMPSPEC, PATH, PROMPT, and SET.

## SHIFT

Lets you use more than the usual 10 replaceable parameters (0%-9%) in batch file processing.

Here is how SHIFT works. Suppose your batch file, Money.bat, contains replaceable parameters, defined as on the left. When you enter the SHIFT command, each definition shifts one place. Instead of replacing %*n*, it replaces %*n-1*.

<u>Definitions Before Shift</u>	<u>Definitions After One Shift</u>
%0 = Money	%0 = Pharm.dat
%1 = Pharm.dat	%1 = Autodeal.dat
%2 = Autodeal.dat	%2 = Hardware.dat
%3 = Hardware.dat	%3 = Grocery.dat
%4 = Grocery.dat	%4 = Dimestor.dat
%5 = Dimestor.dat	%5 = Theater.dat
%6 = Theater.dat	%6 = Cafe.dat
%7 = Cafe.dat	%7 = Photo.dat
%8 = Photo.dat	%8 = Beauty.dat
%9 = Beauty.dat	%9 = undefined

As you can see, %9 is now open to be redefined. Money, on the other hand, no longer replaces any parameter. If you must refer to Money again, there are two ways to do it:

- Use SHIFT so that Money replaces %9, or
- Use SET to set Money equal to a text parameter

You may execute as many shifts as needed. If you do, however, include ECHO %*n* in your file as needed to keep track of each replaceable parameter's current definition.

## Sample Use:

As you know, you can use SHIFT to gain access to more than 10 replaceable parameters. Even if you don't need that many parameters, however, you can use SHIFT to save you typing time.

The batch file below lets you copy as many files as you wish from A:\USER to B:\USER. It substitutes the first given file for %1 and copies it. Then it substitutes the next file, copies it, and so on. Create the batch file as follows:

```
COPY con shiftcop.bat (ENTER)
REM Stop execution at pause after all files are
  copied (ENTER)
:PAUSE (ENTER)
PAUSE (ENTER)
COPY A:\USER\%1 B:\USER\%1 (ENTER)
SHIFT (ENTER)
GOTO PAUSE
(F6) (ENTER)
```

Suppose you want to copy the files Find.exe, Name1.asm, and Name2.asm from A:\USER to B:\USER. Simply enter the name of the batch file, followed by the files to copy, as shown here:

```
shiftcop find.exe name1.asm name2.asm (ENTER)
```

The batch file pauses before each copy, giving you the option to abort. After it copies all files, press (CTRL) (C) at the PAUSE prompt (*Strike a key when ready...*).

## Example:

```
SHIFT (ENTER)
```

shifts all parameters that replace the parameters %0 through %9 down one. Suppose that, before the shift, %1 equals Denfile.dat and %2 equals Myfile.dat. After the shift, %0 equals Denfile.dat and %1 equals Myfile.dat.

**SORT** [**/R**] [**/ + n**] [**<input pathname**]  
[**>output pathname**]

Reads input from the keyboard or a file, sorts the data, and writes it to the display or a file.

SORT is a filter.

## Parameters:

**<input pathname** specifies file to be sorted. If you omit this parameter, SORT sorts keyboard input.

**>output pathname** specifies the file to receive the sorted information. If you omit this parameter, SORT sends the output to the display.

**/R** reverses the sort (sorts from Z to A). If you omit this parameter, the sort is from A to Z.

**/ + n** begins the sort at Column *n*. If you omit this parameter, the sort begins at Column 1.

## Sample Use:

You might use SORT to alphabetize a file by a certain column. Suppose you have this information in a file, Mail.dat, on the current drive:

Jan	King	2 8th St.	Lincoln	NE	68502
Sam	Beck	4 6th St.	Rapid City	SD	57001
Sal	Cleo	7 9th St.	Ft. Worth	TX	76133

You may want to sort the file alphabetically, according to last name, and then store the sorted output in another file. To do so, type

```
SORT / + 15 <mail.dat >sortmail.dat (ENTER)
```

SORT sorts the contents of Mail.dat, starting at Column 15 (last name). It then outputs the sorted data into the file Sortmail.dat.

## **Examples:**

**SORT /R <unsort.txt >sort.txt (ENTER)**

reads the file Unsort.txt, reverses the sort, and then writes the output to a file named Sort.txt.

**DIR | SORT /+ 14 (ENTER)**

pipes the output of the DIR command to the SORT filter, which sorts the directory listing, starting at Column 14 (file size). SORT then displays the output. Thus, the result of this command is a directory sorted by file size.

**DIR | SORT /+ 14 | MORE (ENTER)**

does the same thing as the command in the previous example, except that the MORE filter displays the directory one screen at a time.



## SYS *drive*

Transfers the files IO.SYS and MSDOS.SYS (the MS-DOS system files), in that order, from the current disk to the disk in the specified *drive*.

The target disk may be blank, in which case transferring the files converts it to a system disk. Or, it may contain old system files, in which case transferring the files updates the system.

IO.SYS and MSDOS.SYS are both hidden files that do not appear when the DIR command is executed. COMMAND.COM (the command processor) is not transferred. You must use the COPY command to transfer COMMAND.COM.

### Sample Use:

Data disks and many application program disks do not contain the system. As is, such a disk can be used only if you have a system disk in Drive A. With the system files on it, however, it can be used alone.

### Example:

SYS B: **(ENTER)**

transfers the system files from the current disk to the disk in Drive B.

### Error:

If SYS detects an error, MS-DOS displays one of the following messages:

#### No room for system on destination disk

If the target disk contains files you don't need, use the DEL command to delete as many as necessary to make room for the system. Otherwise, use a different disk as the target.

#### Incompatible system size

The system files IO.SYS and MSDOS.SYS do not take up the same amount of space on the target disk as the new system requires.

## TIME [*hh:mm:ss.cc*]

Displays or sets the time.

You can change the time from the keyboard or from a batch file. (Normally, MS-DOS displays a time prompt each time you start up your system. It does not, however, if you use an Autoexec.bat file. Therefore, you may want to include a TIME command in that file.)

### Parameters:

[*hh:mm:ss.cc*] specifies the time to set.

*hh* = 0-23 (hours)

*mm* = 0-59 (minutes)

*ss* = 0-59 (seconds)

*cc* = 0-99 (hundredths of a second)

If you include only part of the information (such as the hours) and press **ENTER**, the other fields default to zero.

If you omit the time, TIME displays the current time and prompts you to enter the new time. Enter it in the 24-hour format, or press the **ENTER** key if you don't want to change the time displayed.

### Sample Uses:

When you change the time known to the system, you also change the time in any application program you use. This can be very handy.

Suppose that you have a program that keeps track of customer calls according to the date and time received. For some reason, you get behind and can't enter the information at the correct time. Simply enter it later, after "turning back the clock" to the necessary time.

You can also use DATE and TIME to include the date and time on a printout. (See the DATE command.)

In addition, TIME lets you keep track of the time and can give you an idea how long it takes to run a particular program.

## Examples:

TIME 14 **ENTER**

sets the time to 2:00 p.m.

TIME 2:32 **ENTER**

sets the time to 2:32 a.m.

## Error:

If the options or separators are not valid, MS-DOS displays the message

```
Invalid time  
Enter new time:
```

It then waits for you to enter a valid time.

## TYPE *pathname*

Displays the contents of the specified file.

TYPE makes only one change to the file's format. It expands tabs to spaces consistent with a tab stop at every eighth column.

**Note:** A display of binary files causes control characters (such as end-of-file characters, bells, form feeds, and escape sequences) to be sent to your computer.

### Sample Use:

Use TYPE to see if you need to change a file. If so, use EDLIN.

Suppose you create a batch file, as follows:

```
COPY con prepdisk.bat (ENTER)
REM This is a file to prepare and check new disks
  (ENTER)
REM It is called Prepdisk.bat (ENTER)
FORMAT B: (ENTER)
PAUSE (ENTER)
CHKDSK B: (ENTER)
(F6) (ENTER)
```

Because you don't specify otherwise, MS-DOS creates the file in your current directory. Therefore, to display the file, type only

```
TYPE prepdisk.bat (ENTER)
```

MS-DOS displays

```
REM This is a file to prepare and check new disks
REM It is called Prepdisk.bat
FORMAT B:
PAUSE
CHKDSK B:
```

**Examples:**

TYPE \USER\taxfile.dat (ENTER)

displays the file Taxfile.dat that is in the \USER directory in the current drive.

TYPE B: carfile (ENTER)

displays the file Carfile that is in the home directory of Drive B.

TYPE B:\BUDGET\clothes (ENTER)

displays the file Clothes that is in B:\BUDGET.

# VER (Version)

**Internal**

---

## VER

Displays the number of the MS-DOS version that you are using.

## Sample Use:

If you have a question or comment regarding your system, you'll need to know the MS-DOS version number when you contact the Tandy customer services department.

## Example:

VER **(ENTER)**

displays the version number.



---

## VERIFY [ON|OFF]

Turns the verify switch on or off, or displays the current setting of VERIFY.

VERIFY has the same purpose as the /V switch in the COPY command. When it is on, it tells MS-DOS to verify that your files are intact (they contain no bad sectors, for example).

### Parameters:

VERIFY ON remains in effect until you change it in a program (by using a SET VERIFY system call), or until you use VERIFY OFF. When VERIFY is on, it verifies all writes to disk.

If you omit ON and OFF, MS-DOS returns the current setting of VERIFY.

### Sample Use:

By keeping VERIFY on, you always know if your files are intact. Note, however, that this slows down operation.

### Examples:

VERIFY ON **ENTER**

tells MS-DOS to verify all writes to disk.

VERIFY OFF **ENTER**

tells MS-DOS to stop verifying writes to disk.

### Error:

You receive an error message only if MS-DOS was unable to successfully write your data to disk.

## VOL [*drive*]

Displays the volume label of the disk in the specified *drive*. (Remember, you can assign a label by using the */V* parameter in the FORMAT command.)

If the disk does not have a volume label, VOL displays

Volume in drive *n* has no label

### Parameters:

If you omit the *drive*, MS-DOS displays the volume label of the current disk.

### Sample Use:

Enter the volume command whenever you forget a volume label. For the disk in Drive B, type

VOL B: **(ENTER)**

MS-DOS displays a message similar to this:

Volume in drive B is DISKONE

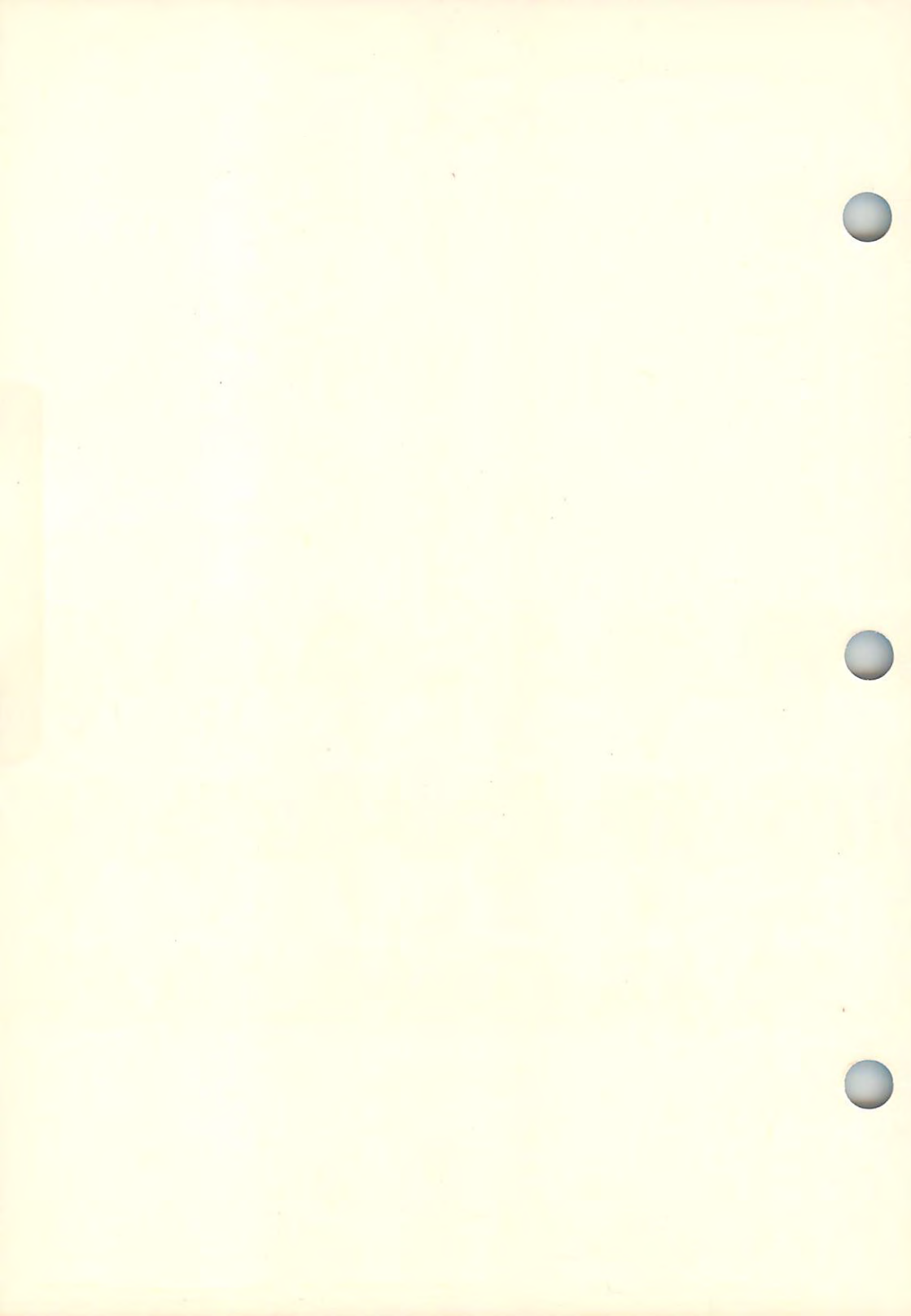
### Example:

VOL **(ENTER)**

displays the volume label of the current disk.







## Section III

---

# The Line Editor (EDLIN)

EDLIN is a text line editor program. It consists of several commands that let you create, change, and display source program or text files. This section describes the various EDLIN commands and provides some practical examples of their use.

It also explains how to use the editing keys to edit files. (You were introduced to these keys in Section I Chapter 4, which explained how to use them to edit MS-DOS commands.)

Some common uses for EDLIN include:

- Creating and saving new source files.
- Updating existing files and saving the updated and original versions.
- Deleting, editing, inserting, or displaying lines.
- Searching to replace or delete text within one or more files.

Files you create with EDLIN are divided into lines of up to 253 characters. During editing, EDLIN generates and displays a number for each line. If you insert or delete one or more lines, it automatically renumbers the following lines as needed to maintain consecutive numbering. The line numbers are only to aid you in editing; they are not actually in the file.

## Starting the EDLIN Program

To start EDLIN, simply type

EDLIN *pathname* **(ENTER)**

where *pathname* refers to either an existing file (that you wish to edit) or one that you wish to create.

## Creating a File and Saving It

If the file specified by *pathname* does not exist in the designated drive, EDLIN creates the file in memory. It gives the file the specified name and prompts you with

New file

```
*_
```

The asterisk (\*) is the EDLIN prompt and the underscore (\_) represents the cursor.

To enter text in the newly created file, type

```
I (ENTER)
```

The I command (insert lines) is discussed in further detail later. EDLIN displays the number of the first line followed by the asterisk:

```
1:*_
```

At this point, you can begin entering information. Remember, you may include up to 254 characters in each line. If you try to enter more, EDLIN generates a beep. To end one line and start another, press (ENTER). EDLIN places the line you just typed in the template and displays the next line number.

After entering all the information that you want, press (CTRL) (C) when EDLIN displays the next line number. Then type E (ENTER) to save the newly created file. EDLIN gives the file the filename and extension you specified.

## Editing an Existing File

If the file specified by *pathname* currently exists in the designated drive, EDLIN loads it into memory. It then displays the message

End of input file

```
*_
```

You may then edit the file using EDLIN commands and editing keys.

If the file is large, EDLIN loads lines until memory is 75% full. When it displays the \* prompt, you can edit those lines. To edit more of the file, you must first free some memory by writing the edited lines to the disk. (See the Write command in this section.) Then you can load more of the file for editing. (See the Append Lines command.)

## Creating a Backup File

Whenever you edit an existing file, EDLIN automatically makes a backup of the original version when you type **E** (**ENTER**) to end the file. It saves the backup file with the original filename plus the extension .bak.


**Note:** You cannot edit a file that has a .bak extension because EDLIN assumes it is a backup file. If you want to edit the backup file, rename it, giving it another extension. (See the REN command in Section II.)

## Special Editing Keys

In Chapter 4, you learned how to use the special editing keys to correct and change MS-DOS commands. You can also use them to edit lines within your files.

EDLIN sends newly typed lines to the template. This makes it possible to repeat or change file lines with the editing keys.

The table below summarizes the editing keys' functions as they apply to file editing.

Function	Key	Description
Copy <i>char</i>		Copies one character to the new line.
Copy to <i>char</i>	<b>F2</b> <i>char</i>	Copies all characters up to the specified character to the new line.
Copy all	<b>F3</b>	Copies all remaining characters in the template to the new line.
Delete <i>char</i>	<b>DELETE</b>	Deletes a character in the template. Therefore, the character is skipped (is not copied to the new line).
Delete to <i>char</i>	<b>F4</b> <i>char</i>	Deletes all characters up to the character specified. Therefore, those characters are skipped (are not copied to the new line).
Void line	<b>F8</b> or <b>CTRL X</b>	voids the current input; leaves the template unchanged.
Insert	<b>INSERT</b>	Enters/exits the insert mode.
Replace template	<b>F5</b>	Replaces the template with the characters displayed to allow further editing. (The characters are not sent to the program.)
Enter line	<b>ENTER</b>	Makes the new line the new template and sends it to the requesting program.
Tab forward	<b>TAB</b>	Moves cursor forward to the next tab stop.

## Sample Uses for the Editing Keys

In this section, you will create a sample file and practice editing with the editing keys.

### Creating a File

To create the sample file, simply insert your MS-DOS system disk in Drive A and type

```
EDLIN sampfile (ENTER)
```

If you want the file to reside in a directory other than the current directory, specify the directory. This command, for example, tells EDLIN to create the file in the home directory of Drive B:

```
EDLIN B:sampfile (ENTER)
```

EDLIN displays the message and prompt

```
New file  
*-
```

to indicate it has created the sample file.

### Entering Information in the Sample File

1. Type I (ENTER) to begin inserting text. The screen displays:

```
New file  
*|  
  1:*-
```

2. On Line 1, type

```
This is a sample file (ENTER)
```

EDLIN displays the number for the next line.

3. Type

```
The editing keys are easy to use (ENTER)
```

4. When EDLIN displays the next line number, press (CTRL) (C). This exits the insert mode and returns the EDLIN prompt (\*).

5. Press **(1)** **(ENTER)** to tell EDLIN to display Line 1. The screen looks like this:

```
*1
      1:*This is a sample file
      1:*
```

Having created the sample file, you are ready to begin using the editing keys.

### Example 1: Copy Char (**(→)**)

Using the Copy Char function, **(→)**, copy characters from the template to the new line.

1. Press **(→)** once. The screen looks like this:

```
1:*This is a sample file
1:*T_
```

2. Press **(→)** three times. The screen looks like this:

```
1:*This is a sample file
1:*This_
```

Each time you press **(→)**, EDLIN copies another character from the template to the new line.

3. To continue to the next example, press **(CTRL)** **(C)**. This exits the insert mode, voids any changes you have made to the line, and returns the EDLIN prompt (\*).

### Example 2: Copy to Char (**(F2)**)

Using the Copy to Char function, **(F2)**, copy all characters from the template, up to — but not including — the specified character.

1. At the EDLIN prompt, press **(2)** **(ENTER)**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*
```



2. Press **(F2)** and type **a**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*The editing keys _
```

3. Now copy the rest of the template, using the Copy All function, **(F3)**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*The editing keys are easy to use
```

4. Press **(CTRL) (C)** to continue to the next example.

**Note:** If the template does not contain the specified character, nothing is copied.

### Example 3: Copy All **(F3)**

Using the Copy All function, **(F3)**, copy the entire contents of the template to the new line.

1. Press **(1) (ENTER)** at the EDLIN prompt (\*). The screen looks like this:

```
1:*This is a sample file
1:*_
```

2. Press the Copy All editing key, **(F3)**. The screen looks like this:

```
1:*This is a sample file
1:*This is a sample file
```

3. Press **(CTRL) (C)** to continue to the next example.

**Note:** The Copy All function copies all characters in the template to the new line. However, if you changed part of the template, the Copy All function copies only the remaining characters — those that have not been edited.

### Example 4: Delete Char (**DELETE**)

Using the **Delete char** function, delete some characters from the template.

1. At the EDLIN prompt, press **2** **ENTER**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*
```

2. Press **DELETE** three times to delete the first three characters from the template. Then press the Copy All function key, **F3**. The screen looks like this:

```
2:*The editing keys are easy to use
2:* editing keys are easy to use
```

3. Press **CTRL** **C** to void any changes to Line 2 and continue to the next example.

### Example 5: Delete to Char (**F4**)

Using the Delete to Char function, **F4**, delete a number of characters up to — but not including — the specified character.

1. At the EDLIN prompt, press **1** **ENTER**. The screen looks like this:

```
1:*This is a sample file
1:*
```

2. Press **F4** and type **a**. The cursor remains in the same position even though the characters before the first **a** have been deleted. To see this, press the Copy All key, **F3**. The screen looks like this:

```
1:*This is a sample file
1:*a sample file_
```

3. Press **CTRL** **C** to void any changes and continue to the next example.

## Example 6: Void Line ((F8))

In this example, first make a change to Line 1 and then, using the Void Line function key, (F8), cancel that change. This leaves the template unchanged and lets you continue making changes to the same line.

1. At the EDLIN prompt, press (1) (ENTER). The screen looks like this:

```
1:*This is a sample file
1:*
```

2. Type

Names and addresses (F8)

The screen looks like this:

```
1:*This is a sample file
1:*Names and addresses\
-
```

When you press (F8), a backslash (\) appears right after any change you may have made and the cursor is displayed again on another line.

3. Press the Copy All key, (F3). The screen looks like this:

```
1:*This is a sample file
1:*Names and addresses\
  This is a sample file _
```

4. Press (CTRL) (C) to continue to the next example.

### Example 7: Insert (**INSERT**)

The **INSERT** key acts as a switch to turn the insert mode on and off. Use this key to insert a word in Line 1 of the sample file.

1. At the EDLIN prompt, press **1** **ENTER**. The screen looks like this:

```
1:*This is a sample file
1:*_
```

2. Press the Copy to Char key, **F2** and type f. The screen looks like this:

```
1:*This is a sample file
1:*This is a sample _
```

3. Now press **INSERT** to turn on the insert mode. Type **edit**, followed by a single space. Then press **INSERT** again to turn off the insert mode. The screen looks like this:

```
1:*This is a sample file
1:*This is a sample edit _
```

4. To copy the rest of the characters in the template, press the Copy All key, **F3**. The screen looks like this:

```
1:* This is a sample file
1:* This is a sample edit file
```

5. Press **CTRL** **C** to continue to the next example.

### Example 8: Replace Template (**F5**)

Using the Replace Template function, **F5**, replace the template with whatever characters are displayed in the new line.

1. At the EDLIN prompt, press **2** **ENTER**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*_
```

2. Type

#### Replacing the template

Then press **(F5)**. Notice that the @ symbol appears after the last character and the cursor moves to the next line:

```
2:*The editing keys are easy to use
2:*Replacing the template@
```

After you press **(F5)**, all the characters displayed in the new line replace those in the template. This is similar to pressing **(ENTER)**. However, there is one important difference. With **(F5)**, the displayed characters go only to the template for further editing; they do not go to the requesting program.

3. Press the Copy All function key, **(F3)**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*Replacing the template@
   Replacing the template
```

As you can see, the new sentence is now the template.

4. Press **(CTRL)(C)** to void any changes and continue to the next example.

**Note:** Pressing **(ENTER)** immediately after pressing **(F5)** empties the template.

### Example 9: Enter Line (**(ENTER)**)

Whenever you press **(ENTER)**, the displayed characters become the template and are sent to the requesting program.

1. At the EDLIN prompt, press **(2)(ENTER)**. The screen looks like this:

```
2:*The editing keys are easy to use
2:*_
```

2. Press the Copy to Char key, **(F2)**, and then type **a**. The screen shows

```
2:*The editing keys are easy to use
2:*The editing keys _
```

3. Then type

```
simplify editing tasks (ENTER)
```

The screen looks like this:

```
2:*The editing keys are easy to use
2:*The editing keys simplify editing tasks
*_
```

Notice that after you press **(ENTER)**, the EDLIN prompt returns to the screen.

4. Press **(2)** **(ENTER)**. The screen looks like this:

```
2:*The editing keys simplify editing tasks
2: _
```

As you can see, Line 2 now contains the new sentence you entered.

## EDLIN Commands

In addition to the editing keys, which are ideal for making changes and corrections to individual lines, EDLIN also supports many powerful commands that let you manipulate an entire file or several lines at a time.

Most EDLIN commands follow a certain format and conform to certain rules. Before proceeding, review these:

- Except for End Edit and Quit, EDLIN commands are preceded and/or followed by parameters (parameters are discussed further later).
- EDLIN commands and parameters may be entered in any combination of upper-case or lower-case characters.
- Except for the Edit Line command, EDLIN commands are a single letter.
- You can indicate line numbers relative to the current line. A minus sign ( - ) indicates lines that precede the current line. A plus sign ( + ) indicates lines that follow it. For example:

`- 10, + 10L ENTER`

lists the 10 lines immediately preceding the current line, the current line, and the 10 lines immediately following the current line.

- Multiple commands can be entered in the same command line without any special delimiting characters. The exceptions are:
  - when you use the Edit Line command for a single line
  - when you use the Search and Replace command

In the first case, you must use a semicolon to separate the commands. For example, the multiple command:

15;-5,+5L **(ENTER)**

edits Line 15 and then displays Lines 10 through 20.

In the second case, press **(CTRL)(Z)** (instead of **(ENTER)**) after the string to be replaced. For instance, the multiple command:

S**This string****(CTRL)(Z)**-5,+5L **(ENTER)**

searches for the string **This string** and then displays the five lines that immediately precede the matched string and the five lines that immediately follow it. If the search fails, then EDLIN displays the numbers of the relative lines.

- EDLIN commands can be entered with or without a space between the line number and the command. For example, the delete command **6D** is the same as **6 D**.
- Control characters (such as **(CTRL)(C)** and **(CTRL)(Z)**) can be inserted into text and even used for the Search and Replace commands. To do this, first press **(CTRL)(V)** and then the desired control character.
- Delimiters (spaces or commas) are required only between adjacent line numbers. You may want to use them, however, to separate commands and parameters for better readability.
- Commands become effective after you press **(ENTER)**.
- For commands that produce a large amount of display output, you should press **(CTRL)(S)** or **(HOLD)**. This suspends the display so you can read it. Press **(CTRL)(Q)** or **(HOLD)** to continue display output.
- You may stop EDLIN commands by pressing **(CTRL)(C)**.



The EDLIN commands are summarized in the following table.

Command	Purpose
<i>line</i>	Edits the specified line
A	Appends lines
C	Copies lines
D	Deletes lines
E	Ends editing
I	Inserts lines
L	Lists text
M	Moves lines
P	Pages text
Q	Quits editing
R	Replaces lines
S	Searches text
T	Transfers text
W	Writes lines

Most EDLIN commands require you to specify a line number, a string, or a range of lines to be affected by the command. These are the "command parameters." Command parameters may vary according to the EDLIN command, but, generally, they are as follows.

### ***line* (line number)**

This parameter denotes that you must indicate a line number (which isn't always a numeral, as you'll see shortly). You separate line numbers from the command, the parameters, and other line numbers. The valid separators are the comma and the space.

You may specify a line in one of three ways:

- A number less than 65534. If you specify a number larger than the number of existing lines, EDLIN adds a line to the file.
- A period (.), which indicates the current line. This is the last line edited, not necessarily the last line displayed. The current line is always marked on your screen by an asterisk (\*) between the line number and the first character.
- A pound sign (#), which indicates the line immediately following the last line edited. EDLIN adds a line to the file, as it does if you specify a number larger than the last line number.

If you do not include the *line* parameter in an EDLIN command, EDLIN substitutes that command's default value.

***string* (a group of characters)**

This parameter represents a portion of text EDLIN is to find, replace, delete, or use to replace other text. Use this parameter only with the Search and Replace commands. End each *string* with **CTRL** **Z**. Do not include spaces between strings or between a string and the command letter (unless you want those spaces to be part of the string).

# Append Lines (A)

---

**[*number*]A**

Adds the specified *number* of lines from disk to the file being edited in memory. The lines are added at the end of the file.

This command is meaningful only if the file being edited is too large to fit into memory. Before using this command, you must write the portion of the file that was loaded into memory (and which has been edited) to disk. Refer to the Write Lines command.

When the Append command has read the last line of the file into memory, EDLIN displays the message

End of input file

## Parameter:

If you omit the *number* of lines, EDLIN appends lines until available memory is 75% full. However, if memory is already 75% full, EDLIN does nothing.

## Example:

100A (ENTER)

appends 100 lines from the disk to the file in memory.

# Copy Lines (C)

---

`[line1][,line2],line3[,count]C`

Copies all lines ranging from *line1* to *line2*, placing them immediately ahead of *line3*. Using the *count* parameter, you can copy the lines as many times as you want.

## Parameters:

If you omit *line1* or *line2*, EDLIN copies the current line. If you omit the *count*, EDLIN copies the line(s) only once.

**Note:** If you omit the *line1* parameter, your command must include a comma immediately preceding the *line2* parameter.

The file is renumbered automatically after the copy and the first line copied becomes the current line. For example:

```
3,9,12C (ENTER)
```

copies Lines 3-9 to Line 12. Line 12 becomes the current line.

**Note:** Line numbers must not overlap. If they do, EDLIN displays an error message. Also, the plus and minus signs are not allowed in the *count* field.

## Examples:

Assume that the following file exists and is ready to be edited. (This file is used for the next few pages. If you want to create it, see the Insert command.)

- 1: [This sample file](#)
- 2: [shows what happens](#)
- 3: [when you use](#)
- 4: [the Copy command](#)
- 5: [to copy text in your file.](#)

You can copy this entire block of text by typing this at the EDLIN prompt:

```
1,5,6C (ENTER)
```

Almost instantly the prompt reappears to indicate that the command was executed. If you wish to see the result, type L **(ENTER)**. The screen shows

```
1: This sample file
2: shows what happens
3: when you use
4: the Copy command
5: to copy text in your file.
6:*This sample file
7: shows what happens
8: when you use
9: the Copy command
10: to copy text in your file.
```

Notice that the first line copied becomes the current line.

If you want to insert lines between other lines, use *line3* to specify the line before which you want the text copied. For example, assume that you want to insert Lines 5-8 before Line 3 in the following file:

```
1: This sample file
2: shows what happens
3: when you use the Copy command
4: to copy text in your file.
5: If you so choose,
6: you may also insert
7: a group of lines within
8: other parts of the file.
9: The copy command
10: is versatile.
```

The command 5,8,3C **ENTER** results in the following file:

- 1: This sample file
- 2: shows what happens
- 3: If you so choose,
- 4: you may also insert
- 5: a group of lines within
- 6: other parts of the file.
- 7: when you use the Copy command
- 8: to copy text in your file.
- 9: If you so choose,
- 10: you may also insert
- 11: a group of lines within
- 12: other parts of the file.
- 13: The copy command
- 14: is versatile.

# Delete Lines (D)

---

**[*line1*][,*line2*]D**

Deletes *line1* or all lines within the range *line1* to *line2* from a file.

EDLIN automatically renumbers the lines to maintain consecutive numbering in the file.

## Parameters:

You may omit the *line1* parameter, as in

*,line2*D (ENTER)

If you do, EDLIN starts deleting at the current line and ends at *line2*. Your command line must include the comma before *line2* to indicate you are omitting *line1*.

You may omit the *line2* parameter, as in

*line1*D (ENTER) or *line1*,D (ENTER)

If you do, EDLIN deletes only *line1*.

If you omit *line1* **and** *line2*, as in

D (ENTER)

EDLIN deletes only the current line.

## Examples:

Assume that the following file exists and is ready to be edited. Line 30 is the current line:

```
1: This sample file
2: shows how
3: the Delete command functions.
4: All that is required
5: in most cases
.
.
.
26: is to specify
27: a number of lines
28: to be deleted;
29: deleting can often help
30:*to clean up your files.
```

Type

5,25D **(ENTER)**

The result is

- 1: This sample file
- 2: shows how
- 3: the Delete command functions.
- 4: All that is required
- 5:\*is to specify
- 6: a number of lines
- 7: to be deleted;
- 8: deleting can often help
- 9: to clean up your files.

To delete line 4 in the above file, type

4D **(ENTER)**

The result is

- 1: This sample file
- 2: shows how
- 3: the Delete command functions.
- 4:\*is to specify
- 5: a number of lines
- 6: to be deleted;
- 7: deleting can often help
- 8: to clean up your files.

To delete the current and the following two lines, type

,6D **(ENTER)**

The result is

- 1: This sample file
- 2: shows how
- 3: the Delete command functions.
- 4:\*deleting can often help
- 6: to clean up your files.



To delete only the current line, type

D **ENTER**

The result is

- 1: This sample file
- 2: shows how
- 3: the Delete command functions.
- 4:\*to clean up your files.

# Edit Line (*line*)

---

[*line*]

Lets you load the specified *line* for editing.

EDLIN displays the specified line and — on the next line — the line's number (without text) to indicate the line is ready for editing. You can then use the editing keys, control keys, and the EDLIN commands.

## Parameters:

If you omit the *line* (you only press **ENTER**), EDLIN loads the line that immediately follows the current line.

If you don't need to change the line, and the cursor is at the beginning or end of the line, press **ENTER** to accept the line as is.

If you press **ENTER** while the cursor is in the middle of a line, EDLIN erases the rest of the line.

Finally, if you wish to cancel any changes made to a line, press **F8**.

## Example:

Assume that the following file exists and you wish to edit Line 4:

```
1: This is a sample file.  
2: used to show  
3: the editing of line  
4:*four.
```

Type 4 **ENTER** at the EDLIN prompt. The screen looks like this:

```
*4  
4: four  
4: _
```

Press **(INSERT)**, type **number** followed by a single space, and press **(INSERT)** again. Then, to copy the rest of the line, press the Copy All editing key, **(F3)**. The screen shows

\*4

4:\*four

4:\*number four\_

At this point you may:

- Save the changed line by pressing **(ENTER)**.
- Type more text at the end of the line (the insert mode is in effect whenever the cursor is at the end of the line).
- Press **(F5)** (Replace Template) to further edit the line.
- Press **(F8)** or **(CTRL) (C)** to cancel the changes made to the line.

# End Edit (E)

---

## E

Ends the EDLIN program and saves the edited file.

When EDLIN saves the edited file, it uses the drive specification, filename, and extension you specified when you started EDLIN. (If you want, you can use Copy to transfer the file to another drive.)

If you edited an existing file (rather than a newly created one), EDLIN also saves the original file (unedited file). In the original file's case, however, EDLIN gives the file the extension .bak (for backup).

When you enter the End Edit command, EDLIN appends a **CTRL** **Z** character to serve as the end-of-file mark.

After you enter the End Command, control returns to MS-DOS. The system prompt (probably **A>**) is displayed.

**Warning:** Be sure the disk contains enough free space to save the entire file. If it does not, EDLIN saves only part of the file; the rest is lost. If this occurs to a file, the portion saved has the file extension of .\$\$\$.

# Insert (I)

---

[*line*]I

Inserts lines of text immediately before the specified *line*. Also, when you create a new file, you must enter the I command to begin writing text.

Text begins with Line 1. Successive line numbers appear automatically each time you press **(ENTER)**.

EDLIN remains in the Insert mode until you press **(CTRL) (C)**. After you press **(CTRL) (C)**, the line immediately following the insertion becomes the current line. EDLIN automatically increments all line numbers that follow the insertion as necessary.

## Parameters:

If you use a period (.) to specify the *line* — or if you omit the *line* parameter — EDLIN uses the current line.

If you use a pound sign (#) to specify the *line* — or if the *line* is any number larger than the number of the last line — EDLIN appends the lines to the end of the file. In this case, the last line inserted becomes the current line.

## Examples:

Assume that the following file exists and is ready to be edited:

```
1: This is a sample file
2: to show what happens
3: when using the Insert command
4: in your files
```

To insert text before a specific line that is not the current line, type *line* **(ENTER)**. For example, type

```
3I (ENTER)
```

The result is

```
3: _
```

Now, type the new text for Line 3:

**3: to the line numbers**

To continue text insertion, press **(ENTER)**. EDLIN displays a new line number. Type

**(how they are renumbered) (ENTER)**

Then return to the EDLIN prompt by pressing **(CTRL) (C)**. Type **L (ENTER)** to display the result:

```
1: This is a sample file
2: to show what happens
3: to the line numbers
4: (how they are renumbered)
5:*when using the Insert command
6: in your files
```

To insert a line immediately before the current line, type **I (ENTER)** at the EDLIN prompt. The screen shows

```
5: _
```

Now type

**every time you insert new lines (ENTER)**

When EDLIN displays the number for the next line, press **(CTRL) (C)** to return to the EDLIN prompt (\*). Type **L (ENTER)** to list the file. The screen looks like this

```
1: This is a sample file
2: to show what happens
3: to the line numbers
4: (how they are renumbered)
5:*every time you insert new lines
6: when using the Insert command
7: in your files
```

To add lines to the end of the file, type

**8I (ENTER)**

The screen shows

8: \_

Enter the following new lines

- 8: The Insert command
- 9: can be used
- 10: to add new lines
- 11: to the end of your file

Press **CTRL** **C** when the number for Line 12 is displayed: Then, at the EDLIN prompt, type **L** **ENTER** to list the file. The screen looks like this:

- 1: This is a sample file
- 2: to show what happens
- 3: to the line numbers
- 4: (how they are renumbered)
- 5: every time you insert new lines
- 6: when using the Insert command
- 7: in your files
- 8: The Insert command
- 9: can be used
- 10: to add new lines
- 11: to the end of your file

## List (L)

---

**[*line1*][,*line2*]L**

Displays all lines within the range *line1* and *line2*.

### Parameters:

**Note:** If *line1* is more than 11 lines before the current line, EDLIN displays the same lines it does if you omit both parameters.

You may omit *line1*, as in

*line2* **(ENTER)**

If you do, the display begins before the current line and ends with *line2*. The comma is required to indicate that you omitted *line1*.

You may omit *line2*, as in

*line1* **(ENTER)**

If you do, EDLIN displays 23 lines, starting with *line1*.

If you omit both parameters, EDLIN displays 23 lines — the 11 lines immediately preceding the current line, the current line, and the 11 lines immediately following the current line. If fewer than 11 lines precede the current line, EDLIN displays more lines that follow the current line to make a total of 23 lines.



## Examples:

Assume that the following file exists and is ready to be edited:

```
1: This is a sample file
2: used to show the List command.
3: See what happens when you use
4: List (L) with different parameters
5: or without any
.
.
.
15:*The current line contains an asterisk.
.
.
.
26: to edit text
27: in your file.
```

To list a range of lines without reference to the current line, type

```
line,lineL (ENTER)
```

For example, type

```
2,5L (ENTER)
```

to produce the following display:

```
2: used to show the List command.
3: See what happens when you use
4: List (L) with different parameters
5: or without any
```

To list a range of lines beginning with the current line, type

```
.,line L (ENTER)
```

For example, type

.,26L **(ENTER)**

to produce this display:

15:\*The current line contains an asterisk.

.  
. .  
.

26: to edit text

To list a range of 23 lines centered around the current line, you need only type L **(ENTER)**. The screen shows

4: List (L) with different parameters

5: or without any

.  
. .  
.

13: The current line is listed in the middle of the range.

14: The current line remains unchanged by the L command.

15:\*The current line contains an asterisk.

.  
. .  
.

26: to edit text.

## Move Lines (M)

---

**[*line1*][,*line2*],*line3*M**

Moves all lines within the range *line1* to *line2* to the line immediately preceding *line3*. *line1* becomes the current line.

Use the Move command to move a block of lines from one place in the file to another.

### Parameters:

If you omit the *line1* parameter, the *line2* parameter, or both, EDLIN uses the current line.

It renumbers the lines according to the direction of the move. For example:

```
, +25,100M (ENTER)
```

moves the text from the *current line* + 25 to Line 100. If the line numbers overlap, EDLIN displays the message

**Entry error**

### Example:

To move Lines 20-30 to Line 100, type

```
20,30,100M (ENTER)
```

## Page (P)

---

[*line1*][,*line2*]P

Pages through a file 23 lines at a time or lists the specified block of lines.

### Parameters:

You may omit the *line1* parameter, as in

,*line2*P (ENTER)

If you do, EDLIN uses the *current line number + 1*, unless the current line is Line 1. The comma is required to indicate that you omitted the *line1* parameter.

You may omit the *line2* parameter, as in

*line1*P (ENTER)

If you do, EDLIN lists 23 lines.

The last line displayed becomes the current line. This command differs from the List command in that the Page command changes the current line (to the last line displayed).

### Example:

To display Lines 10 through 15, type

10,15P (ENTER)

To display Lines 20 through 42, type

20P (ENTER)

EDLIN displays the specified line (Line 20) and the next 22 lines. Line 42 becomes the current line.

Then, to display Lines 43 through 65, type

P (ENTER)

EDLIN displays the *current line + 1* (Line 43) and the next 22 lines. Line 65 becomes the current line.

# Quit (Q)

---

## Q

Quits the editing session without saving any changes you may have made to the file.

EDLIN prompts you to make sure you don't want to save the changes. Press **Y** if you want to quit the editing session without saving any changes. Press **N** to continue editing.

## Example:

```
*Q ENTER  
Abort edit (Y/N)?_
```

# Replace String (R)

**[*line1*][,*line2*][?]**R***string1* **(CTRL Z)** *string2***

Replaces all occurrences of *string1* with *string2*. The replacement is limited to the lines between *line1* and *line2*.

Notice that you must terminate *string1* by pressing **(CTRL Z)**, then begin *string2* immediately following the **(CTRL Z)** character. Terminate *string2* by pressing another **(CTRL Z)** or **(ENTER)**.

Each time Replace finds *string1*, it replaces the *string1* with *string2*. It displays the line in which it has made a replacement. If a line contains more than one replacement, Replace displays it once for each occurrence.

When all occurrences of *string1* in the specified range are replaced with *string2*, the Replace command terminates, and the EDLIN prompt (\*) reappears.

## Parameters:

If you omit *line1*, Replace starts the search at the line that immediately follows the current line; it stops at *line2*.

If you omit *line2*, Replace continues the search to the end of the file.

Therefore, if you omit *line1* and *line2*, searching begins at the line that immediately follows the current line and ends at the end of the file.

? tells Replace to prompt you with **O.K.?** each time it modifies a line. Press **(Y)** to accept the change. Press **(N)** to reject it. In either case, the search continues, starting at the next line.

If you omit *string2*, EDLIN **deletes** all occurrences of *string1* in the specified line range.

If you omit *string 1* and *string2*, EDLIN uses the most recent string specified with a Search (or Replace) command as *string1*, and the most recent string specified in a Replace command as *string2*.

## Examples:

Assume that the following file exists and is ready for editing:

- 1: This sample file
- 2: shows how the Replace and Search commands work.
- 3: When you use Replace and Search
- 4: certain words and phrases
- 5: may be exchanged for
- 6: other words and phrases
- 7: in your file.

To replace all occurrences of **and** with **or** in the above file, type

```
1,7Rand(CTRL)(Z)or(ENTER)
```

The result is

- 2: shows how the Replace or Search commands work.
- 2: shows how the Replace or Search commors work.
- 3: When you use Replace or Search
- 4: certain words or phrases
- 6: other words or phrases

Note that Lines 1, 5, and 7 are not displayed because they are not changed. Line 2 is displayed twice because the **and** string occurred by itself and within the word **commands**. This is an unwanted substitution.

To avoid unwanted changes, you can include the **?** parameter. For example, type

```
2,7?Rand(CTRL)(Z)or(ENTER)
```

Now, whenever Replace finds the **and** string, you have the opportunity to accept or reject the change.

# Search Text (S)

**[*line1*][,*line2*][?]S*string*(ENTER)**

Searches all lines within the range *line1* to *line2* for each occurrence of the text *string*.

## Parameters:

You must terminate the *string* by pressing (ENTER).

If you omit the question mark (?), Search displays the first line (in the specified range) that matches the *string*. That line becomes the current line, and the Search command terminates.

If the *string* does not occur between *line1* and *line2*, Search displays the message

**Not found**

If you include the question mark, Search displays the first line that contains the *string*. It then prompts you with the message

**O.K.?**

If you press either (Y) or (ENTER), the line becomes the current line, and the search terminates. If you press any other key, the search continues until it finds another match or searches all lines.

You may omit the *line1* parameter, as in

**,*line2*S*string* (ENTER)**

If you do, the search begins at the line that immediately follows the current line.

You may omit the *line2* parameter, as in

***line1*S*string* (ENTER) or *line1*,S*string* (ENTER)**

If you do, Search uses the pound sign (#), which indicates the line after the last line of the file. Omitting *line2* is the same as specifying *line1*,#*string*.



If you omit the *string*, Search uses the last search string that was used in a Replace or Search command. If there is no search string (no previous search or replace has been done), the command terminates immediately.

**Notes:**

1. Search looks for the *string* exactly as you specify it in the command. Therefore, enter the string in the same combination of upper- and lower-case characters as it appears in the text.
2. Begin the search *string* immediately after the S in the command line. End it by pressing **ENTER**. Any spaces you include are considered part of the search string.
3. In multiple line commands, terminate the *string* with **CTRL Z**, instead of **ENTER**. You may then follow the string with another command, in the next character position.

**Examples:**

Assume that the following file exists and is ready to be edited. Line 5 is the current line.

- 1: This sample file
- 2: shows how the Search command functions.
- 3: to locate and point out
- 4: a specified string
- 5:\*in a range of lines
- 6: of your file.
- 7: The Search command
- 8: may include the optional parameter ? and
- 9: two line parameters for the range.
- 10: You should also specify the string and press  
ENTER.

To search for the first occurrence of the string **and**, type

1,10 Sand(ENTER)

The screen shows

2: shows how the Search command  
functions.

\*

because the string **and** is part of the word **command**. Line 2 then becomes the current line.

This is probably not the **and** for which you were looking. To find the next occurrence of **and**, type

S (ENTER)

at the EDLIN prompt. The screen looks like this:

\* 1,10 Sand  
2: shows how the Search command  
functions.

\*S

3: to locate and point out

Line 3 becomes the current line.

To search through several occurrences of a string until you find the correct one, type

1, ? Sand

The result is

\* 1,10 Sand

2: shows how the Search command functions.

O.K.? N

3: to locate and point out

O.K.? N

7: The Search command

O.K.? N

8: may include the optional parameter ? and

O.K.?

The search continues until you press  or the file runs out of lines. At that time, Search displays the message

Not found

# Transfer Lines (T)

---

**[*line*]T[*drive*]*filename***

Inserts (merges) the contents of a specified file into the file being edited. The transferred file is inserted just ahead of the specified *line* or current *line*.



After the file is inserted, the merged file is renumbered automatically.

**Note:** The file to be transferred is read from the current directory of the specified or the default drive. If you issued a path when EDLIN was started, that path serves as the current directory. All subsequent Transfer Lines commands use that directory.

## Parameters:

If you omit the *line*, then the file contents are inserted ahead of the current line.

## Example:

10TB:myfile **(ENTER)**



insert the contents of B:Myfile into the file being edited. B:Myfile is inserted just before Line 10.



# Write Lines (W)

---

**[*number*]W**

Writes a specified *number* of edited lines from memory to disk. Writing begins with Line 1.

This command is meaningful only if the file you are editing is too large to fit into memory. When you start EDLIN, files are loaded until memory is 75% full. To edit the rest of your file, you must first write edited lines in memory to disk. Then you can load the unedited lines from disk into memory, using the Append command.

**Note:** If you omit the *number*, EDLIN writes lines until 25% of memory is freed. If at least 25% already is freed, EDLIN takes no action. All lines remaining in memory (not written to disk) are renumbered starting with Line 1.

## Example:

**100W (ENTER)**

writes Lines 1 through 100 to disk and renumbers the file in memory, starting with Line 101 as Line 1.

## Error Messages

When you enter an invalid EDLIN command, or fail to follow the proper syntax, EDLIN displays one of the following error messages:

### **Cannot edit .BAK file — rename file**

**Cause:** You are trying to edit a file with an extension of .bak. You cannot do so because EDLIN reserves the .bak extension for backup copies.

**Cure:** If you need the .bak file for editing purposes, you must either RENAME the file with a different extension or COPY it, giving it a different extension.

### No room in directory for file

- Cause: You are trying to create a file, but either the file directory is full or you are specifying an illegal disk drive or filename.
- Cure: Check the command line that started EDLIN for either an illegal filename or an illegal drive specification. If the command is no longer on the screen, and if you have not yet typed a new command, you can recover the command by pressing the Copy All key, (F3).  
  
If the command line contains no illegal entries, run the CHKDSK program for the specified disk drive. If the status report shows that the disk directory is full, remove the disk. Insert and format a new disk.

### Entry Error

- Cause: The last command typed contained a syntax error.
- Cure: Re-enter the command, using the correct syntax.

### Line too long

- Cause: During a Replace command, the string given as the replacement caused the line to expand beyond the limit of 253 characters. EDLIN aborted the Replace command.
- Cure: Divide the long line into two lines, then try the Replace command again.

### Disk Full — file write not completed

- Cause: You entered the End command, but the disk does not contain enough free space for the whole file. EDLIN aborted the End command and returned you to the operating system. Some of the file may have been written to the disk.

Cure: Only part (if any) of the file is saved. Delete that part of the file and restart the editing session. The file will not be available after this error. Before you begin any editing session, be sure the disk has sufficient free space for the file.

#### **Invalid drive name or file**

Cause: You specified an illegal drive or filename when you started EDLIN.

Cure: Specify a legal drive or filename.

#### **Filename must be specified**

Cause: You did not specify a filename when you started EDLIN.

Cure: Specify a filename whenever you start EDLIN.

#### **Invalid Parameter**

Cause: You specified a switch other than /B when starting EDLIN.

Cure: Specify the /B switch when you start EDLIN.

#### **Insufficient memory**

Cause: There is not enough memory to run EDLIN.

Cure: Free some memory by writing files to disk or by deleting files before restarting EDLIN. Use the Write command and then the Append command.

#### **File not found**

Cause: The filename specified during a Transfer command was not found.

Cure: Specify a valid filename when issuing a Transfer command.

### Must specify destination number

- Cause: You did not specify a line number in a Copy or a Move command.
- Cure: Re-enter the command, including a destination line number.

### Not enough room to merge the entire file

- Cause: There is not enough room in memory to hold the file during a Transfer command.
- Cure: Free some memory by writing some files to disk or by deleting some files before you can transfer the file.

### File creation error

- Cause: The EDLIN temporary file cannot be created.
- Cure: Verify that the directory has enough space to create the temporary file. Also, be sure that the file does not have the same name as a subdirectory in the directory where the file to be edited is located.








# Introduction

---





This section describes how to use the linker program, MS™-LINK. You should read all of the Chapter 1, “Basic Information,” before you use the linker. Advanced users may also want to read the Chapter 2, “Technical Information.” Chapter 3 contains the linker error messages.

If you are not going to compile and link programs, you do not need to read this section.

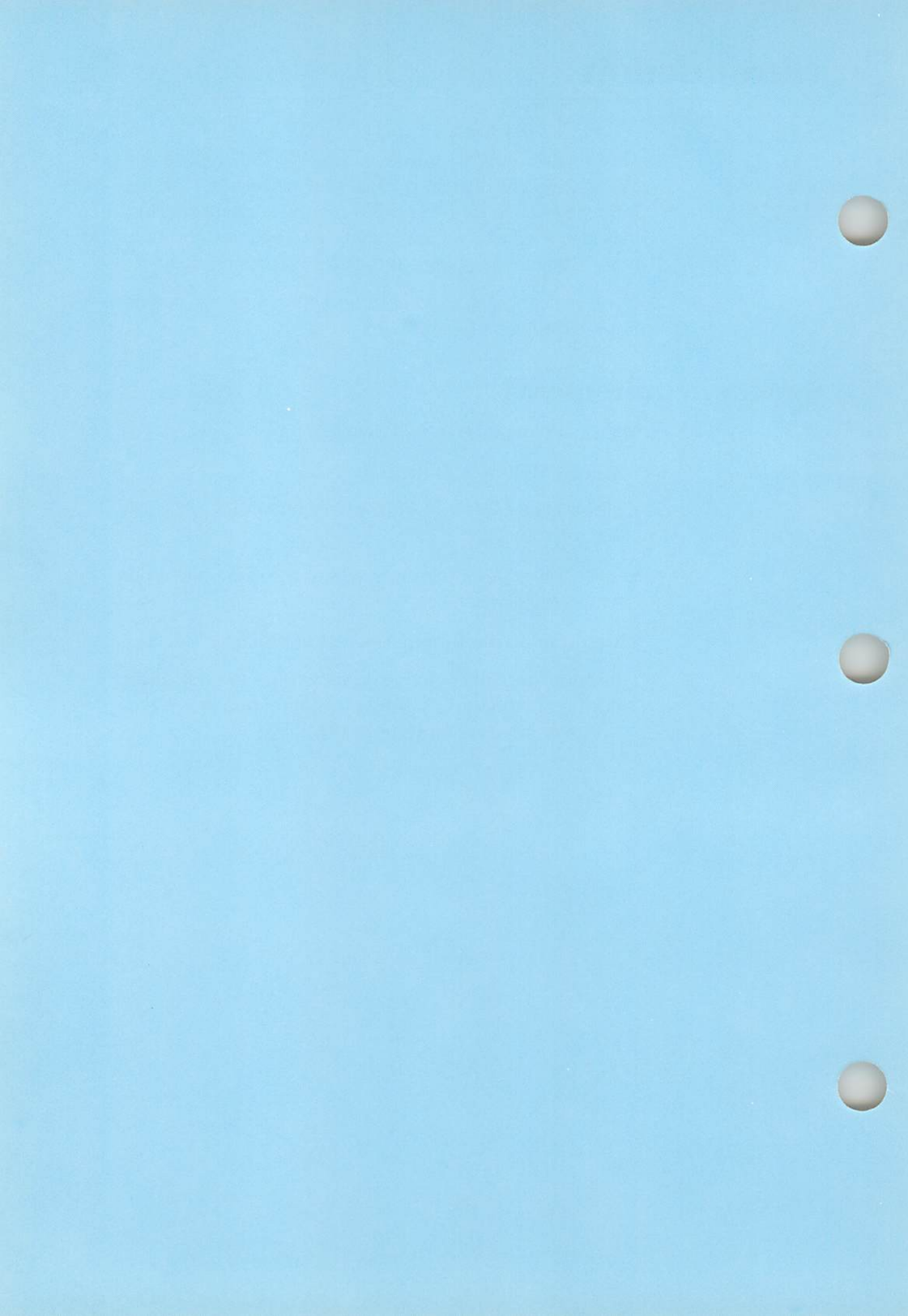
## System Requirements

The linker requires the following:

- 
- At least 50K bytes of memory.  
40K bytes are for code and data; the remaining 10K bytes are for run space.
  - One disk drive, if output is sent to the same disk from which input was taken.
  - Two disk drives, if output is sent to a different disk.



Because the linker does not allow time to swap disks during operation on a 1-drive configuration, the use of two disk drives is more practical.



# Chapter 1

---

## Basic Information

### Overview of the Linker

When you write a program, you do so in source code. Then, by passing the source code through a compiler or an assembler, you produce an object module. This object module, however, cannot be understood by the computer directly.

To produce “machine language,” code that the computer **can** understand, you must pass the object module through the link process.

Additional functions of the linker are summarized below:

### Linking Object Modules and Producing a Run File

You may wish to link (combine) several separately produced object modules and run them as one program. The linker enables you to do this. It produces one relocatable load module, or “run” file (also called an .exe or executable file).

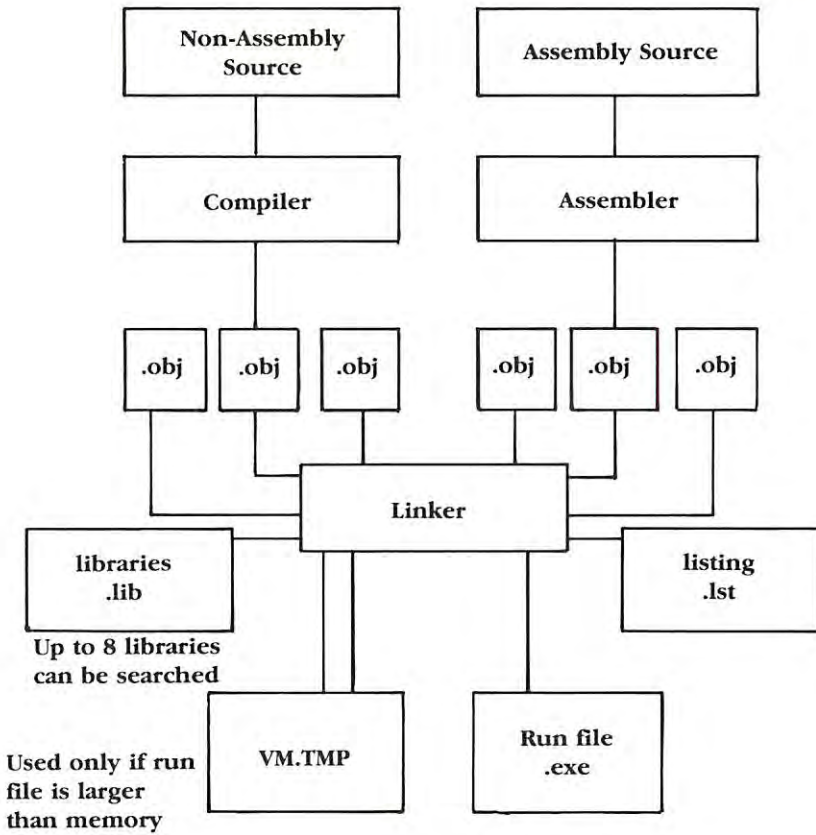
### Resolving External References

In addition to containing internal references, modules that you link may contain “external references” (references to symbols that are defined in the **other** modules). As it combines modules, the linker resolves all external references (makes sure they are defined). If any external reference is not defined in the object modules, the linker searches up to eight library files for the definition.

### Producing a List File

The linker also produces a List file, which shows external references resolved and displays any error messages.

The following diagram illustrates the various parts of the linker’s operation:



## The VM.TMP (Temporary) File

The linker uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, the linker creates a temporary file, names it VM.TMP, and puts it on the disk in the current drive. It displays

VM.TMP has been created.  
Do not change diskette in drive d:

Once this message is displayed, you should not remove the disk from the current drive until the link session ends. If you do remove the disk, the operation of the linker will be unpredictable, and the linker might display the error message

#### Unexpected end of file on VM.TMP

The contents of VM.TMP are written to the file you name at the Run File: prompt (see “Command Prompts”). VM.TMP is a working file only; the linker deletes it automatically at the end of the linking session.

**Warning:** Do not use VM.TMP as a filename. If you have a file named VM.TMP on the current drive and the linker requires the VM.TMP file, it destroys the existing VM.TMP file and creates a new VM.TMP.

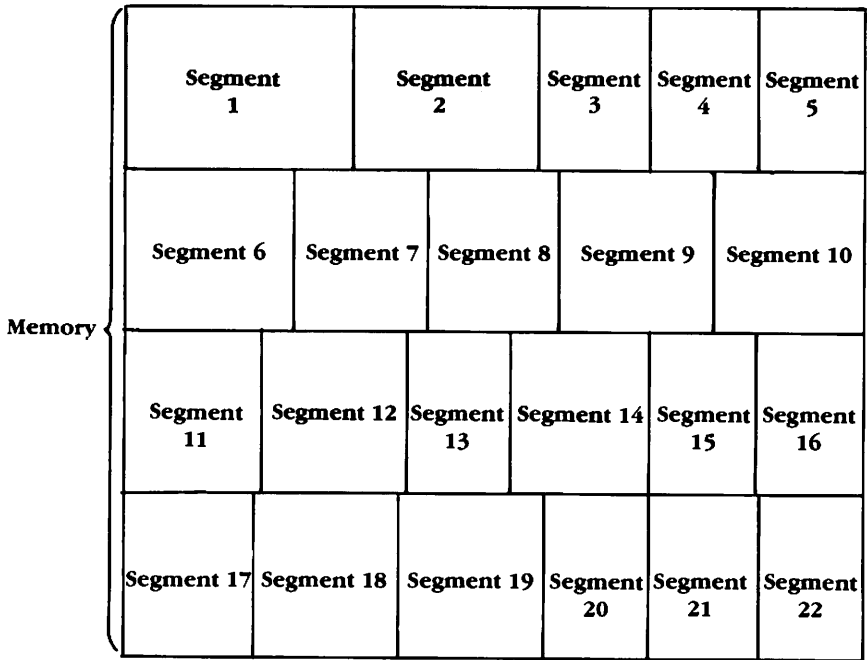
## Definitions

Some of the terms used in this section are explained below to help you understand how the linker works. Generally, if you are linking object modules compiled from BASIC, Pascal, or another high-level language, you do not need to know these terms. If you are writing and compiling programs in assembly language, however, you must understand the linker and the definitions described below. (Chapter 2, “Technical Information,” also contains useful information on how the linker works.)

In MS-DOS, memory can be divided into “segments,” “classes,” and “groups.” The diagram below illustrates these concepts.

*Section IV / The Linker*

---



shaded area = a group (64K bytes addressable)



Example:

	Segment Name	Class Name
Segment 1	PROG.1	CODE
Segment 2	PROG.2	CODE
Segment 12	PROG.3	DATA

Note that segments must have different **segment** names but may or may not have the same **class** name. Segments 1, 2, and 12 form a group; the group address is the lowest address of Segment 1 (the lowest address in memory). Definitions of the terms segment, class, and group follow.

## Segment

A segment is a contiguous area of memory up to 64K bytes long. It can be located anywhere in memory on a paragraph (16-byte) boundary. The contents of a segment are addressed by a segment address and an offset within that segment. Segments can overlap.

## Group

A group is a collection of segments that fit within a 64K byte area of memory. The segments do not need to be contiguous (see illustration). The group address is the lowest address of the lowest segment in that group. At link time, the linker analyzes the groups, then references the segments by the group address. A program may consist of one or more groups.

If you are writing in assembly language, you may assign the names of groups in your program. In high-level languages, the compiler automatically assigns the names. The linker checks to see that the object modules of a group meet the 64K-byte constraint.

## Class

A class is a collection of segments. The naming of segments to a class controls the order and relative placement of segments in memory. In an assembly-language program, you assign names to the classes. In high-level language programs, the compiler automatically assigns the names.

All segments assigned to the same class are loaded into memory contiguously. The segments are ordered within a class in the order in which the linker encounters them in the object files. One class precedes another in memory only if a segment for the first class precedes all segments for the second class in the input to the linker. Classes may be any size, and groups may span classes.

Refer to Chapter 2, “Technical Information,” for information on how to assign group and class names and on how the linker combines and arranges segments in memory.

## Command Prompts

After you start the linker (see “Starting the Linker” later in this chapter), a series of four prompts appears on your display. When you have typed a response to a prompt and pressed **(ENTER)**, the next prompt appears.

After you answer the last prompt, the linker begins linking. If it finishes successfully, the linker exits and the MS-DOS system prompt is displayed. If an error occurs, the linker displays the appropriate error message.

The linker prompts you for the names of the object, run, and list files, and for libraries. Use a standard MS-DOS pathname to reference each file.

The prompts are discussed below in the order they appear. Defaults are shown in square brackets ( [ ] ) following the prompt. To select a default, simply press **(ENTER)** in response to a prompt. [The Object Modules:](#) prompt is the only one that requires you to type a path-name.

## Object Modules [.OBJ]:

Enter a list (one or more) of the object modules to be linked. If an object module has an extension other than .obj, specify it. If you do not, the linker assumes it is .obj.

Separate module names from one another with a blank space or a plus sign (+).

Remember, the linker loads segments into classes in the order encountered. You can use this information to set the order in which the linker reads the object modules. Refer to Chapter 2, "Technical Information," for more information on this process.

## Run File [*first object pathname.EXE*]:

Enter a pathname. The linker creates a file of that name to store the run file that results from the link session. It assigns the run file the extension .exe, even if you specify an extension other than .exe.

If you do not enter a pathname in response to this prompt, the linker uses the first pathname typed in response to the Object Modules: prompt.

Example:

Run File [PROG1.EXE]: B:payroll/P **(ENTER)**

This response directs the linker to create the run file Payroll.exe on Drive B. Because the /P switch is included (see "Linker Switches" later in this chapter), the linker pauses to let you insert a new disk to receive the run file.

## List File [NUL.MAP]:

Enter a pathname. If you do not specify an extension, the linker assigns the extension .map. The list file contains an entry for each segment in the object modules. Each entry shows the offset for that segment in the run file.

If you do not enter a pathname in response to this prompt, the linker uses the default value (NUL.MAP) and thus does not produce a list file.

## Libraries [.LIB]:

Enter up to eight library filenames or simply press **ENTER**. (Pressing only **ENTER** tells the linker to search for default libraries in the object modules.) Library files must have been created by a library utility.

If you do not specify an extension, the linker assumes it is .lib.

Separate library pathnames with blank spaces or plus signs (+).

The linker searches library files in the order listed to resolve external references. When it finds the module that defines the external symbol, the linker processes that module as another object module.

If the linker cannot find a library file on the disks in the disk drives, it displays

Cannot find library *library file*  
Type new drive letter:

Press the letter for the drive designation (for example, **B**).

## Command Characters

The linker provides three command characters.

### Plus Sign

Use the plus sign (+) to separate entries and extend lines as needed. (A blank space may be used instead to separate entries.)

If the response to the Object Modules: or Libraries: prompt is too long to fit on a line, type + **ENTER** at the end of the line. The prompt appears again, and you can continue typing the response. After listing all the modules, press **ENTER**.

## Semicolon

At any time after the first prompt (Run File:), you may select default responses to the remaining prompts. To do so, type ; **(ENTER)**. This eliminates the need to press **(ENTER)** repeatedly.

**Note:** Once you have entered the semicolon, you can no longer respond to any of the remaining prompts for that link session. Therefore, do not use the semicolon to skip only a few prompts. To skip prompts, use the **(ENTER)** key.

**(CTRL) (C)**

Use **(CTRL) (C)** to abort the link session at any time. If you enter an erroneous response — such as the wrong pathname or an incorrectly spelled pathname — you must press **(CTRL) (C)** to exit the linker. Then you can restart the linker.

**Note:** If you made an error but have not pressed **(ENTER)**, use **(BACKSPACE)** to delete characters in that line.

## Linker Switches

The seven linker switches control various linker functions. When you use switches, you may group them at the end of any response or scatter them at the ends of several. To specify a switch, type a forward slash (/) followed by the switch name (which you can abbreviate to the first letter).

### /DSALLOCATE

The /D switch tells the linker to load all data at the high end of the data segment. If you omit /D, all data is loaded at the low end.

At runtime, the DS pointer is set to the lowest possible address to allow the entire DS segment to be used. Using /D and omitting /H allows any available memory below the area specifically allocated within DGroup to be allocated dynamically by the user program and remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

**Note:** Your application program may dynamically allocate up to 64K bytes (or the actual amount of memory available) less the amount allocated within DGroup.

## **/HIGH**

The /H switch causes the linker to place the run file as high as possible in memory. If you omit /H, the linker places the run file as low as possible in memory.

**Important:** Do not use /H with Pascal or FORTRAN programs.

## **/LINENUMBERS**

The /L switch tells the linker to include in the list file the line numbers and addresses of the source statements in the input modules. If you omit /L, the linker does not include the line numbers. (If the object modules do not contain line number information, the linker cannot include line numbers.)

## **/MAP**

The /M switch directs the linker to list all public (global) symbols defined in the input modules. If you omit /M, the linker lists only errors (including undefined globals).

The symbols are listed alphabetically at the end of the list file. For each symbol, the linker lists its value and its segment:offset location in the run file.

## **/PAUSE**

The /P switch causes the linker to pause in the link session when the switch is encountered. This lets you swap disks before the linker outputs the run file. If you omit /P, the linker does the link without stopping.

When the linker encounters the /P switch, it displays the message:

About to generate .EXE file  
Change disks <hit any key>

Press **SPACEBAR** to resume processing.

**Warning:** Do not remove the disk that is to receive the list file or the disk used for the VM.TMP file, if one has been created.

## **/STACK:size**

The /S switch lets you specify the stack size. If you omit /S, the linker calculates the required stack size from information in the object modules provided by the compiler or assembler.

**size** can be any positive value (in decimal) up to 65535 bytes. If you enter a value from 1-511, the linker uses 512.

At least one object (input) module must contain a stack allocation statement. If not, the linker displays the error message:

**WARNING: NO STACK STATEMENT**

## **/NO**

The /N switch tells the linker **not** to search the default (product) libraries in the object modules. For example, suppose you are linking object modules in Pascal. If you specify /N, the linker does not search the library named Pascal.lib to resolve external references.

# Starting the Linker

You can start the linker in any of these ways:

- By entering responses to the individual prompts as they are displayed
- By including all responses on the command line
- By creating a file to automatically respond to the prompts

## **Method 1: Keyboard Responses**

Type

LINK **(ENTER)**

The linker is loaded into memory and displays the four prompts, one at a time. (These prompts and possible responses are described under “Command Prompts.”)

## Method 2: Responses on Command Line

Type all prompt responses on the LINK command line. Separate the responses with commas. Use the following format:

LINK *objlist*,*runfile*,*listfile*,*liblist*[/*switch* . . .](ENTER)

***objlist*** is a list of object modules. Use a blank space or a plus sign to separate the module names.

***runfile*** is the name of the file to receive the executable output.

***listfile*** is the name of the file to receive the listing.

***liblist*** is a list of library modules to be searched. Use a blank space or a plus sign to separate the module names.

***/switch*** refers to optional switches. Switches may follow any response list (they may immediately precede any comma or immediately follow the *liblist*).

To select the default for a field, simply type a second comma with no spaces between the two commas. Example:

LINK fun + text + table + care/P/M,,  
funlist,coblib.lib (ENTER)

This command causes the linker to be loaded. Then the object modules Fun.obj, Text.obj, Table.obj, and Care.obj are loaded. The linker then pauses (because of the /P switch). When you press (SPACEBAR), the linker links the object modules and produces a global symbol map (because of the /M switch). It then creates a run file with the default name Fun.exe, creates a list file named Funlist.map, and searches the library file Coblib.lib.



## Method 3: Response File

Type

LINK @filespec **ENTER**

**filespec** is the name of an automatic response file, which contains answers to the linker prompts. The extension is optional. There is no default extension.

This method permits the command that starts the linker to be entered from the keyboard or within a batch file without requiring you to take any further action.

Before using this option, you must create an automatic response file. This file should contain several lines of text, each of which is the response to a linker prompt. The responses must be in the same order as the linker prompts discussed earlier. If desired, a long response to the Object Modules: or Libraries: prompt may be typed on several lines. Use a plus sign ( + ) at the end of a line to continue the response on the next line.

Use switches and command characters in the response file the same way as they are used for responses typed from the keyboard.

When the link session begins, each prompt is displayed, in order, with the responses from the file. If the file does not contain a response for a prompt, the linker displays the prompt and waits for you to enter a valid response. After you do so, the linker continues.

Example:

```
fun text table care
/P /M
funlist
coblib.lib
```

The first line in this file tells the linker to load four object modules: Fun, Text, Table, and Care. The second line omits a name for the run file; this tells the linker to use the name Fun.exe.

When the linker encounters the /P switch, it pauses to let you swap disks. After doing so, press **(SPACEBAR)** to continue. Because the line includes the /M switch, the linker produces a public symbol map (see “Switches”).

The third line tells the linker to name the list file Funlist.map. The fourth line tells it to search the library file Coblib.lib.

## Sample Link Session

This sample shows you the kind of information that is displayed during a link session.

In response to the MS-DOS prompt, type

LINK **(ENTER)**

The system displays the following messages and prompts, with your responses:

```
Microsoft Object Linker V.2.00  
(C) Copyright 1982 by Microsoft Inc.  
Object Modules [.OBJ]: io sysinit  
Run File [IO.EXE]: /M  
List File [NUL.MAP]: PRN /L  
Libraries [.LIB]:
```

Notes:

1. Because you specify /M, the linker displays an alphabetical and a chronological listing of public symbols.
2. By responding PRN to the List File: prompt, you redirect your output to the printer.
3. Because you specify the /L switch, the linker lists all line numbers for all modules. (The /L switch can generate a large amount of output.)
4. Because you press **(ENTER)** in response to the Libraries: prompt, the linker performs an automatic library search.

Once the linker locates all libraries, the linker map displays a list of segments in the order of their appearance within the load module. The list might look like this:

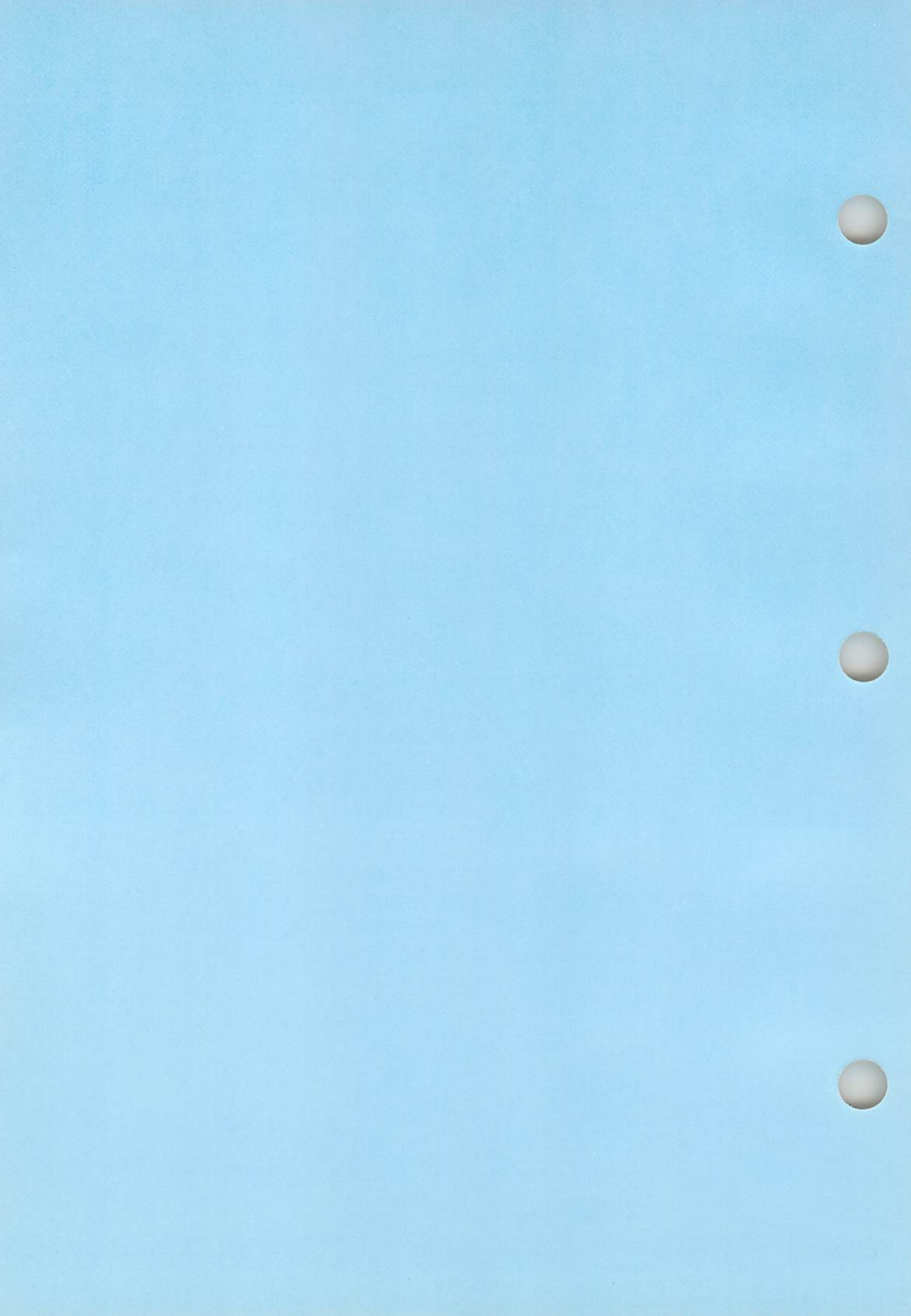
Start	Stop	Length	Name
00000H	009ECH	09EDH	CODE
009F0H	01166H	0777H	SYSINITSEG

The information in the start and stop columns shows the 20-bit hex address of each segment relative to location zero. Location zero is the beginning of the load module.

The addresses displayed are not the absolute addresses where these segments are loaded. See Chapter 2, "Technical Information," for information on how to determine where relative zero is actually located and how to determine the absolute address of a segment.

Because you used the /M switch, the linker displays the public symbols by name and value. For example:


ADDRESS	PUBLICS_BY_NAME
009F:0012	BUFFERS
009F:0005	CURRENT_DOS_LOCATION
009F:0011	DEFAULT_DRIVE
009F:000B	DEVICE_LIST
009F:0013	FILES
009F:0009	FINAL_DOS_LOCATION
009F:000F	MEMORY_SIZE
009F:0000	SYSINIT
ADDRESS	PUBLICS BY VALUE
009F:0000	SYSINIT
009F:0005	CURRENT_DOS_LOCATION
009F:0009	FINAL_DOS_LOCATION
009F:000B	DEVICE_LIST
009F:000F	MEMORY_SIZE
009F:0011	DEFAULT_DRIVE
009F:0012	BUFFERS
009F:0013	FILES



# Chapter 2

---

## Technical Information



This chapter contains detailed information about the linker that is of interest to advanced assembly language programmers.

The linker is able to link files totaling one megabyte. The output file from the linker (a run file) is not bound to specific memory addresses and, therefore, can be loaded and executed at any convenient address by the operating system. The relocation information is a list of long addresses that must change when the executable image is relocated in memory. See “Long References” later in this chapter for an explanation of long addresses.

## Definitions

The following terms describe some of the functions of the linker. For definitions of segment, group, and class, see Chapter 1, “Basic Information,” in this section.




### Alignment

Alignment refers to certain segment boundaries. These can be byte, word, or paragraph boundaries. You specify the alignment in an assembly-language program.

By specifying byte alignment, you tell the linker it may start a segment on any byte boundary (one segment may immediately follow another). Word alignment tells the linker to start segments only on even addresses. Paragraph alignment tells it to start segments only on 16-byte boundaries.

### Combine Type



A combine type is an attribute of a segment. It tells the linker how to combine segments that have the same name or it relays other information about the properties of a segment. Combine types are: stack, public, private, and common (see “How the Linker Combines and Arranges Segments”).

## Canonical Frame

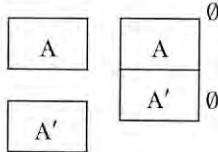
The canonical frame of a group of segments is the starting address of the first segment in the group. Offsets are calculated from this address.

## How the Linker Combines and Arranges Segments

The linker works with four combine types, which are declared in the source module for the assembler or compiler: private, public, stack, and common. The memory combine type available in Microsoft's Macro Assembler is synonymous with the public combine type. The linker does not automatically place memory combine type as the highest segments (as defined in the Intel standard).

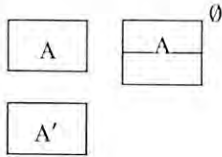
The linker arranges these combine types as follows:

Private



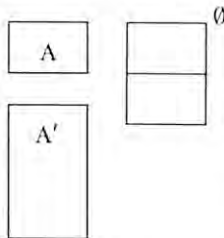
Private segments are loaded separately and remain separate. They may be physically (but not logically) contiguous, even if the segments have the same name. Each private segment has its own canonical frame.

Public and Stack



Public and stack segments of the same name and class name are loaded contiguously. Offset is from the beginning of the first segment loaded through the last segment loaded. There is only one canonical frame for all public segments of the same name and class name. Stack and memory combine types are treated the same as public. However, the stack pointer is set to the last address of the first stack segment.

Common

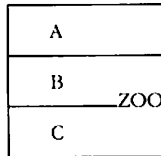


Common segments of the same name and class name are loaded overlapping one another. There is only one canonical frame for all common segments of the same name. The length of the common area is the length of the longest segment.

Placing segments in a group in the assembler provides offset addressing of items from a single canonical frame for all segments in that group.

DS:DGROUP    XXXX0H    0 — relative offset

Any number of other segments may intervene between segments of a group. Thus, the offset of ZOO may be greater than the size of segments in the group combined, but no larger than 64K.



An operand of DGROUP:ZOO in assembly language returns the offset of ZOO from the beginning of the first segment (Segment A here).

Segments are partitioned by declared class names. The linker loads all the segments belonging to the first class name encountered, then loads all the segments of the next class name encountered, and so on until all classes are loaded.

If your program contains:

- A SEGMENT 'ZOO'
- B SEGMENT 'BAZ'
- C SEGMENT 'BAZ'
- D SEGMENT 'NEW'
- E SEGMENT 'ZOO'

Linker loads the segments as:

- 'ZOO'
- A
- E
- 'BAZ'
- B
- C
- 'NEW'
- D



If you are writing assembly-language programs, you can control the order of classes in memory by writing a dummy module and listing it first after the linker's Object Modules: prompt. The dummy module declares segments into classes in the order you want the classes loaded.

**Warning:** Do not use this method with BASIC, COBOL, FORTRAN, or Pascal programs. Allow the compiler and the linker to perform their tasks in the normal way.

Example:

```
A      SEGMENT 'CODE'
A      ENDS
B      SEGMENT 'CONST'
B      ENDS
C      SEGMENT 'DATA'
C      ENDS
D      SEGMENT STACK    'STACK'
D      ENDS
E      SEGMENT 'MEMORY'
E      ENDS
```

Make sure you declare all classes to be used in your program in this module. If you do not, you lose absolute control over the ordering of classes.

Also, if you want memory combine type to be loaded as the last segments of your program, you can use this method. Simply add MEMORY between SEGMENT and 'MEMORY' in the E segment line above. Note, however, that these segments are loaded last only because you imposed this control on them, not because of any inherent capability in the linker or assembler operations.

## Segment Addresses

The 80186 must be able to address all segments in memory. Any 20-bit number can be addressed. The 80186 represents these numbers as two 16-bit numbers (for example, hex F:12). The F is a canonical frame address and the 12 is the offset.

The canonical frame address is the largest frame address or segment address that can contain the segment. An offset is the segment's location, offset from the beginning of the canonical frame.

The linker recognizes a segment by its canonical frame address and its offset within the frame.

To convert the address F:12 to a 20-bit number, shift the frame address left 4 bits and add the offset. Using the above example:

$$\begin{array}{r} \phantom{+} \phantom{F}0 \\ + \phantom{F}12 \\ \hline \end{array}$$

$$F:12 = \phantom{F}102 \quad (20\text{-bit address})$$

## How the Linker Assigns Addresses

To assign addresses to segments, the linker orders each segment by segment and class name. On the basis of the alignment and size of each segment (assuming the segments are contiguous), the linker assigns a frame address and an offset to each segment. This information is used for resolving relocatable references. The addresses start at 0:0.

## Relocation Fixups

The linker performs relocation fixups (or resolves) on four types of references in object modules:

- Short
- Near self-relative
- Near segment-relative
- Long

These references and the linker's fixups are described in the next sections.

## Short References

Short references are all self-relative. The implication is that the frame address of the target and source frames are the same. The linker generates the fixup error message

### Fixup offset exceeds field width

under either of the following conditions:

- The target and source frame addresses are different
- The target is more than 128 bytes before or after the source frame address

The resulting value of the short reference must fit into one signed byte.

## Near Self-Relative References

When near self-relative references are used, the frame address of the target and source frames are the same. The linker generates the fixup error message under either of the following conditions:

- The target and source frame addresses are different
- The target is more than 32K before or after the source frame address

The resulting value of the near self-relative reference must fit into one signed word (16 bits).

## Near Segment-Relative References

Given the target's canonical frame, another frame is specified (via an ASSUME directive or the : operator in assembly language or via a high-level language convention). The target must be addressable through the canonical frame specified. The linker generates the fixup error message under either of the following conditions:

- The offset of the target within the specified frame is greater than 64K or less than zero
- The beginning of the canonical frame of the target is not addressable by the specified frame

The resulting value of a near segment-relative reference must be an unsigned word (16 bits).

## **Long References**

Long references have a target and another frame (specified by an ASSUME or by a high-level language). The target must be addressable through the canonical frame specified. The linker generates the fixup error message under either of the following conditions:

- The offset of the target within the specified frame is greater than 64K or less than zero
- The beginning of the canonical frame of the target is not addressable by the specified frame.

The resulting value of a long reference must be a frame address and an offset.

# Chapter 3

## Linker Error Messages

---

All messages, except for the warning messages, cause the link session to end. After you locate and correct a problem, you must restart the linker.

Messages appear in the list file and are displayed, unless you direct the list file to CON. If you direct the file to CON, the error messages are suppressed.

### ATTEMPT TO ACCESS DATA OUTSIDE OF SEGMENT BOUNDS, POSSIBLY BAD OBJECT MODULE

This usually indicates that a bad object file exists.

### BAD NUMERIC PARAMETER

A numeric value is not given as digits.

### CANNOT OPEN TEMPORARY FILE

The linker cannot create the file VM.TMP because the disk directory is full. Insert a new disk. Do not remove the disk that is to receive the List.map file.

### ERROR: DUP RECORD TOO COMPLEX

A DUP record in an assembly-language module is too complex. Simplify the DUP record.

### ERROR: FIXUP OFFSET EXCEEDS FIELD WIDTH

An assembly-language instruction refers to an address with a short instruction instead of a long instruction. Edit the assembly-language source and reassemble.

### INPUT FILE READ ERROR

This usually indicates that a bad object file exists.

### INVALID OBJECT MODULE

An object module or modules are incorrectly formed or incomplete (as when assembly is stopped in the middle).

### SYMBOL DEFINED MORE THAN ONCE

The linker found two or more modules that define one symbol name.

**PROGRAM SIZE OR NUMBER OF SEGMENTS EXCEEDS CAPACITY OF LINKER**

The total size may not exceed 384K bytes, and the number of segments may not exceed 255.

**REQUESTED STACK SIZE EXCEEDS 64K**

Specify a size less than or equal to 64K bytes with the /STACK switch.

**SEGMENT SIZE EXCEEDS 64K**

The addressing system limit is 64K bytes.

**SYMBOL TABLE CAPACITY EXCEEDED**

The number or length of the names caused them to exceed the limit of approximately 25K bytes.

**TOO MANY EXTERNAL SYMBOLS IN ONE MODULE**

The limit is 256 external symbols per module.

**TOO MANY GROUPS**

The limit is 10 groups.

**TOO MANY LIBRARIES SPECIFIED**

The limit is eight libraries.

**TOO MANY PUBLIC SYMBOLS**

The limit is 1024 public symbols.

**TOO MANY SEGMENTS OR CLASSES**

The limit is 256 (segments and classes taken together).

**UNRESOLVED EXTERNALS: *list***

The external symbols listed have no defining module among the modules or library files specified.

#### **VM READ ERROR**

This is a disk error; it is not caused by the linker.

#### **WARNING: NO STACK SEGMENT**

None of the object modules specified contains a statement allocating stack space, but you typed the /STACK switch.

#### **WARNING: SEGMENT OF ABSOLUTE OR UNKNOWN TYPE**

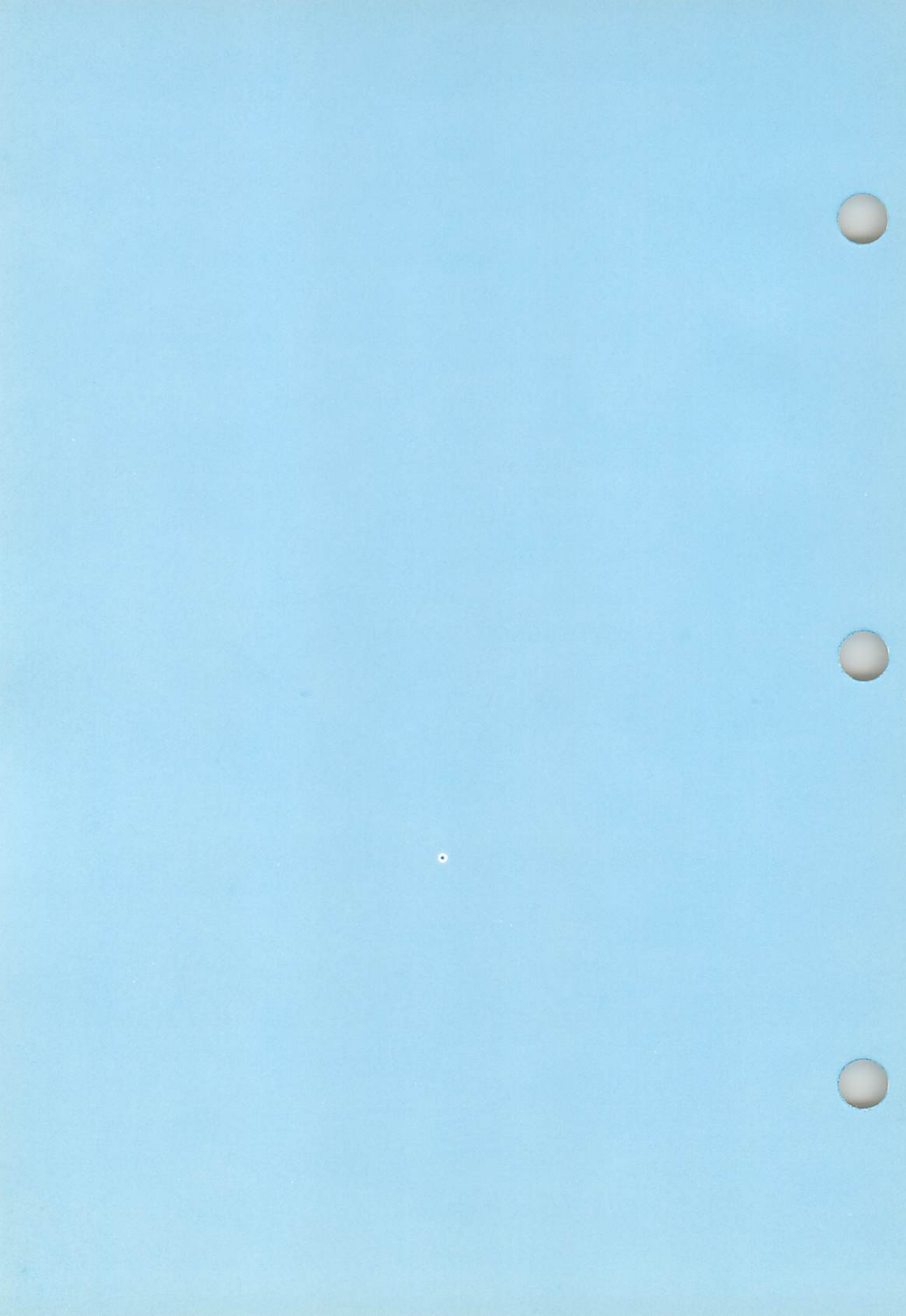
A bad object module exists or an attempt has been made to link modules that the linker cannot handle (for example, an absolute object module).

#### **WRITE ERROR IN TMP FILE**

No disk space remains in which to expand the VM.TMP file.

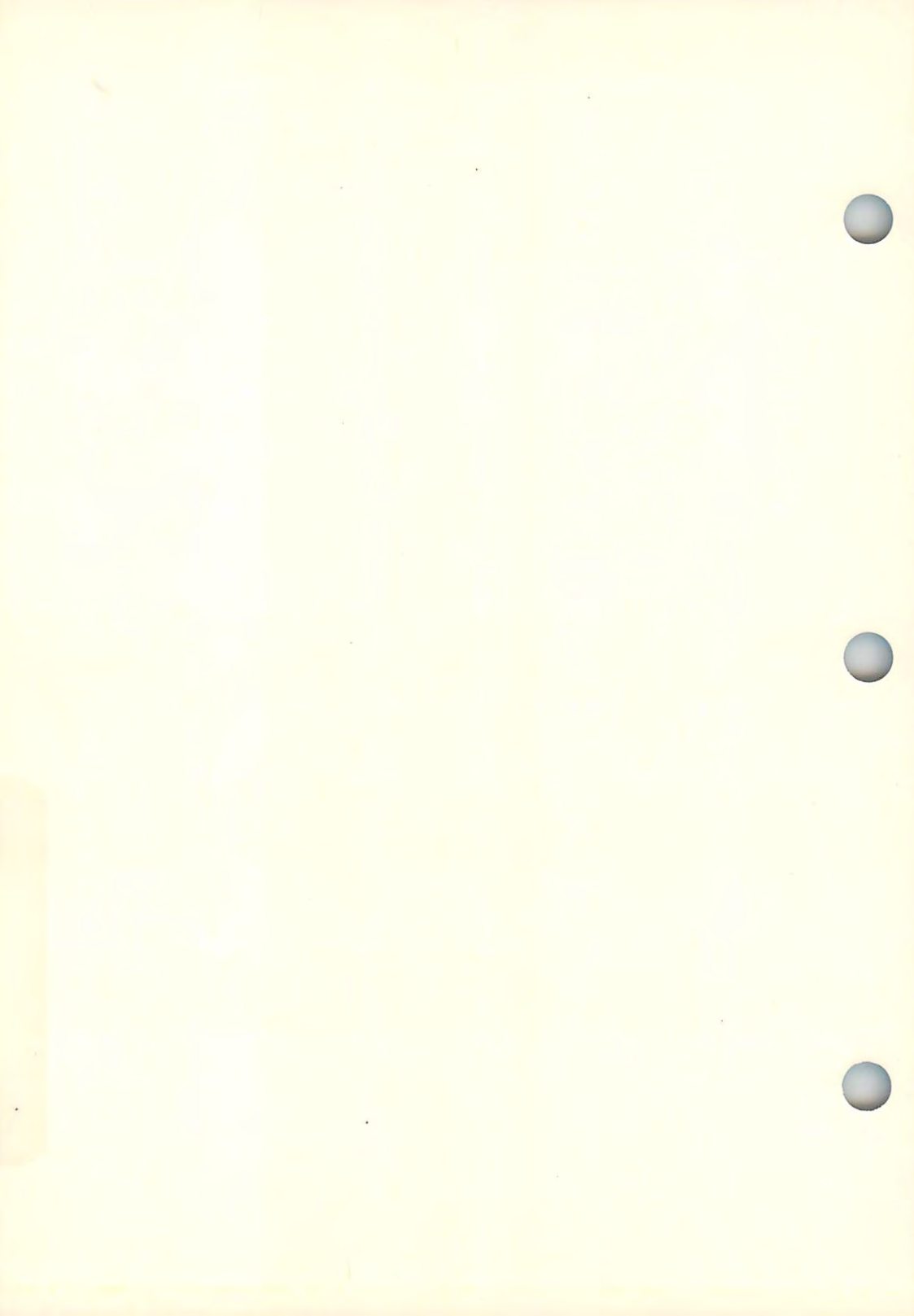
#### **WRITE ERROR ON RUN FILE**

This usually indicates there is not enough disk space for the run file.









## Section V

---

# DEBUG

The DEBUG program provides a controlled environment in which to test your executable object files. You can use DEBUG to alter the contents of a file or register, and then immediately execute the program to see if the changes are valid. You need not reassemble the program.

## How to Start DEBUG

To start DEBUG, type

DEBUG [*pathname*] [*parameters*] **(ENTER)**

*pathname* specifies the program file to be loaded into memory. DEBUG loads the file, starting at 100H in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed into memory.

If you omit the *pathname* (you type only DEBUG **(ENTER)**) you can work with the current register contents. To load a file into memory, you must use the Name and Load commands.

*parameters* is a list of *pathname* parameters and switches that are to be passed to the program when it is loaded. You may include *parameters* only if you included the *pathname*.

DEBUG displays a hyphen (-) to indicate that it is ready to accept a command.

**Note:** When DEBUG is started, it sets up a program segment prefix (PSP) at Offset 0 in the program work area. If you don't specify a *pathname* when you start DEBUG, you can overwrite the default PSP. If you are debugging a .com or .exe file, however, do not tamper with the PSP below relative address 5CH. If you do, DEBUG terminates.

When you start DEBUG, the segment registers (CS, DS, ES, and SS) are set to the bottom of free memory, and the instruction pointer (IP) is set to 0100H. All flags are cleared, and the remaining registers are set to zero.

Do not restart a program after DEBUG displays the message "Program terminated normally." To reload the program, you must use the Name and Load commands. Otherwise, it does not run properly.

## Commands

Each DEBUG command consists of one letter followed by one or more parameters. You can use any combination of upper-case and lower-case letters in commands and parameters. While using DEBUG, you can also use the control keys and the MS-DOS editing functions (described in Section I, Chapter 4).

At any time, you can abort any DEBUG command by pressing **CTRL C**.

To stop the screen from scrolling, press **CTRL S**; to continue scrolling, press **SPACEBAR**.

If a syntax error occurs in a DEBUG command, DEBUG displays the command line with the error indicated by an up-arrow (^) and the word "error." Example:

```
D CS:g00 CS:110
  ^ Error
```

The following table summarizes the DEBUG commands, each of which is described in detail later.

<b>Command</b>	<b>Syntax</b>
Assemble	A [ <i>address</i> ]
Compare	C <i>range address</i>
Dump	D [ <i>range</i> ]
Enter	E <i>address [list]</i>
Fill	F <i>range list</i>
Go	G[ = <i>address1</i> [ <i>address2 ...</i> ]]
Hex	H <i>value1 value2</i>
Input	I <i>portaddress</i>
Load	L [ <i>address [drive sector sectorcount]</i> ]
Move	M <i>range address</i>
Name	N <i>filespec1[filespec2]</i>
Output	O <i>portaddress byte</i>
Quit	Q
Register	R [ <i>registername</i> ]
Search	S <i>range list</i>
Trace	T[ = <i>address</i> ][ <i>value</i> ]
Unassemble	U [ <i>range</i> ]
Write	W [ <i>address [drive sector sectorcount]</i> ]

## Command Parameters

All DEBUG commands except Quit accept parameters. You may separate parameters with spaces or commas, but this is required only between two consecutive hexadecimal values. Thus, the following commands are the same.

```
DCS:100 110  
D CS:100 110  
D,CS:100,110
```

### Parameters

#### **address**

A 1- or 2-part designation in one of the following formats:

- An alphabetic segment register designation and an offset value. Example:

```
CS:0100
```

- A segment address and an offset value. Example:

```
04BA:0100
```

- An offset only, in which case the default segment is used. DS is the default segment for all commands except G, L, T, U, and W, for which the default segment is CS.

All numeric values are hexadecimal. The colon is required to separate a segment designation and an offset.

#### **byte**

A 1- or 2-character hexadecimal value to be placed in or read from an address or register.

#### **drive**

A 1-digit value that indicates which drive the data is to be loaded from or written to.

```
0 = Drive A  
1 = Drive B  
2 = Drive C  
3 = Drive D
```

**filespec**

A file specification that consists of a drive specification, filename, and filename extension. (The three fields are optional, but at least the drive specification or filename should be specified.)

**list**

A series of *strings* or *byte* values. *list* must be the last parameter on the command line. Example:

```
CS:100 FF 42 "XXX" 1A 3
```

**portaddress**

A hexadecimal value of up to four characters, which specifies a port number.

**range**

An area of memory, specified by either of these formats:

- *address1 address2*

Example:

```
CS:100 110
```

*address2* must be an offset value.

- *address L value*

*value* is the number of bytes on which the command is to operate. If you omit *L value*, DEBUG assumes a value of 80 bytes. Do not use this format if another hex value follows the *range*.

Example:

```
CS:100 L 10
```

```
CS:100
```

The limit for the *range* is 10000 hex. To specify a *value* of 10000 hex within four digits, enter 0000 (or 0).

**registername**

See the explanation of the Register command for a list of valid register names.

**sector sectorcount**

1- to 3-character hexadecimal values that indicate the relative sector number on the disk and the number of disk sectors to be written or loaded.

The first sector of Track 0 on Head 0 is Sector 0. The remaining sectors of Head 0, Track 0 are numbered consecutively. Numbering continues with the first sector of the Head 1 track that is of the same radius as Track 0. When all sectors on all heads of the track (that is of the same radius as Track 0) are numbered, numbering continues with the first sector of Head 0 of the next track.

**string**

Any number of characters enclosed in quotation marks. Quotation marks may be either single (') or double ("). The ASCII values of the characters in the string are used as a *list* of byte values.

If quotation marks are to be used within a string, they must be either the opposite set or they must be doubled.

Examples of the correct uses of quotation marks:

- 'This "string" is correct.'
- 'This 'string' is correct.'
- "This 'string' is correct."
- "This ""string"" is correct."

Examples of incorrect uses of quotation marks:

- 'This 'string' is not correct.'
- "This "string" is not correct."

**value**

A hexadecimal value of up to four characters.



# Assemble

---

## A [*address*]

Assembles Macro Assembler statements directly into memory.

***address*** is the starting address at which you want the instructions you enter to be assembled in memory.

If you omit the *address*, the Assemble command uses the address that follows the last instruction assembled by a previous Assemble command. If no Assemble statement was used previously, assembly starts at CS:0100. All numeric values are hexadecimal and can be entered as one to four characters.

If a statement contains a syntax error, DEBUG displays

^ **Error**

and redisplay the current assembly address.

The segment override mnemonics are CS:, DS:, ES:, and SS:. The mnemonic for the far return is RETF. String manipulation mnemonics must explicitly state the string size. For example, use MOVSW to move word strings and MOVSB to move byte strings.

Prefix mnemonics must be specified in front of the opcode to which they refer.

The assembler automatically assembles short, near, or far jumps and calls, depending on byte displacement to the destination address. These may be overridden with the NEAR or FAR prefix. Examples:

```
0100:0500 JMP 502 ; a 2-byte short jump
0100:0502 JMP NEAR 505 ; a 3-byte near jump
0100:0505 JMP FAR 50A ; a 5-byte far jump
```

The NEAR prefix may be abbreviated to NE.

DEBUG cannot tell whether some operands refer to a word memory location or to a byte memory location. In such a case, the data type must be explicitly stated with the prefix "WORD PTR" or "BYTE PTR". These can be abbreviated "WO" and "BY". Example:

```
NEG      BYTE PTR [128]
DEC      WO [SI]
```

DEBUG also cannot tell whether an operand refers to a memory location or to an immediate operand. DEBUG uses the convention that operands enclosed in square brackets refer to memory. Example:

```
MOV      AX,21      ; Load AX with 21H
MOV      AX,[21]    ; Load AX with the contents of
                   ; memory location 21H
```

Two popular pseudo-instructions are available with the Assemble command. The DB opcode assembles byte values directly into memory. The DW opcode assembles word values directly into memory. Example:

```
DB      1,2,3,4,"THIS IS AN EXAMPLE"
DB      'THIS IS A QUOTE: ' '
DB      "THIS IS A QUOTE: ' "
DW      1000,2000,3000,"BACH"
```

All forms of register indirect commands are supported. Example:

```
ADD      BX,34[BP + 2].[SI-1]
POP      [BP + DI]
PUSH     [SI]
```

All opcode synonyms are also supported. Example:

```
LOOPZ   100
LOOPE   100

JA      100
JNBE    100
```

For 8087 opcodes, the WAIT or FWAIT must be explicitly specified. Example:

```
FWAIT FADD ST,ST(3) ; This line will assemble an  
                    ; FWAIT prefix  
FLD TBYTE PTR [BX] ; This line will not
```

# Compare

---

## **C** *range address*

Compares the portion of memory specified by the *range* to a portion of the same size beginning at the specified *address*.

If the two areas of memory are different, DEBUG displays the differences in this format:

*address1 byte1 byte2 address2*

***address1 byte1*** refers to the address and contents of a location in the specified *range*.

***byte2 address2*** refers to the corresponding address and contents in the block starting at *address*.

If the two areas of memory are identical, DEBUG simply returns with the prompt.

If you enter only an offset for the starting address of *range*, the segment indicated by Register DS is used.

### **Example:**

The following commands have the same effect:

**C 100,1FF 300 (ENTER)**

**C 100L100 300 (ENTER)**

Each command compares the block of memory from DS:100 to DS:1FF with the block of memory from DS:300 to DS:3FF.

# Dump

---

D [*address*]

D [*range*]

Displays the contents of the specified *address* or *range* in memory.

The dump is displayed in two portions:

- A hexadecimal portion. Each byte is displayed in hexadecimal.
- An ASCII portion. Each byte is displayed as an ASCII character. Characters that cannot be displayed are shown as a period (.).

Each displayed line begins on a 16-byte boundary and shows 16 bytes. A hyphen appears between the eighth and ninth bytes.

If you type the D command and specify only a starting address, the contents of memory are displayed starting at the address.

If you enter the D command with neither parameter, 128 bytes are displayed at the first address after the last address displayed by a previous D command.

If you enter only an offset for the starting address, the segment indicated by register DS is used.

## Example:

```
D CS:100 109 ENTER
```

DEBUG displays the contents of the range C:100 109 in the following format:

```
04BA:0100 54 4F 4D 20 53 41 57 59 45 52 TOM  
SAWYER
```

# Enter

## E *address* [*list*]

Enters byte values into memory at the specified *address*.

If you type the optional *list* of values, DEBUG automatically replaces the contents of memory beginning at *address* with the new values. Example:

```
E DS:100 45 A1 "abc" 0F
```

DEBUG places the six bytes in the list into memory beginning at DS:100.

If you omit the *list*, DEBUG displays the address and its contents and then waits for your input. You can do any of the following:

- Enter a hexadecimal byte value to replace the displayed value. (Illegal or extra characters are ignored.)
- Press **(SPACEBAR)** to advance to the next byte. To change this value, simply type the new value as described above. Each press of **(SPACEBAR)** advances to the next byte without changing the current byte.

If you space beyond an 8-byte boundary, DEBUG starts a new display line.

- Press **(←)** to back up to the preceding byte. The preceding address and its contents are displayed on the next line. If you want to change this byte, simply type the new value as described above. Each press of **(←)** backs up one more byte without changing the current byte.
- Press **(ENTER)** to end the Enter command.

If you enter only an offset for the address, the segment indicated by register DS is used.

**Example:**

E CS:1004 **(ENTER)**

causes DEBUG to enter byte values into memory beginning at DS:100. DEBUG displays the address and its contents (EB) as shown here. (The underscore (\_) represents the cursor.)

04BA:0100 EB. \_

To change the value from EB to 41, type 41 (at the present cursor position) and press **(SPACEBAR)**. DEBUG stores the value 41 and displays the contents of the next byte:

04BA:0100 EB.41 10. \_

To display the contents of the next two bytes, press **(SPACEBAR)** twice more. You might see:

04BA:0100 EB.41 10. 00. BC. \_

To change BC to 42, type 42 as shown:

04BA:0100 EB.41 10. 00. BC.42 \_

If you want to go back and change 10 to 6F, press **(←)** twice to return to value 10, and then type 6F:

```
04BA:0100 EB.41 10. 00. BC.42-
04BA:0102 00.-
04BA:0101 10.6F_
```

Press **(ENTER)** to end the Enter command.

# Fill

---

## F *range list*

Fills the memory locations in the specified *range* with the values in the *list*.

If the *list* contains fewer bytes than the *range*, DEBUG uses the *list* until all locations in the *range* are filled.

If the *list* contains more bytes than the *range*, DEBUG ignores the extra values in the *list*.

If you enter only an offset for the starting address of the *range*, DEBUG uses the segment indicated by Register DS.

### Example:

```
F 04BA:100 L 100 42 45 52 54 41 ENTER
```

causes DEBUG to fill memory locations 04BA:100 through 04BA:1FF with the bytes specified. The five values are repeated until all 100H bytes are filled.



# Go

---

**G[ = *address1*[ *address2* . . . ] ]**

Executes the program currently in memory. Stops execution at specified breakpoints and displays the registers, flags, and instruction line for the next instruction to be executed.

If you include *address1*, execution begins at *address1* in the CS segment. The equal sign ( = ) is required.

If you don't include an *address*, the program executes starting with the current instruction. The address of the current instruction is determined from the contents of the CS and IP registers.

You can use the other optional addresses to set breakpoints. These allow you to examine the register contents and flag settings in effect when a specified address is reached during program execution.

You can list up to 10 addresses in any order. If you enter only an offset for a breakpoint address, the segment indicated by register CS is used. Breakpoints may be set only at addresses that contain the first byte of an 80186 opcode. Execution stops when any breakpoint is reached.

The user stack pointer must be valid and have 6 bytes available for the Go command. This command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set and the user flags, CS register, and Instruction Pointer are pushed on the user stack.

An interrupt code (0CCH) is placed at the specified breakpoint address(es). When an instruction with the breakpoint code is reached, DEBUG restores all breakpoint addresses to their original instructions. If no breakpoint is reached, the original instructions are not restored.

**Example:**

G CS:7550 **ENTER**

causes the program currently in memory to execute up to address 7550 in the CS segment. DEBUG restores the original instructions, displays the register contents and the flags, and the Go command ends.

After execution halts at a breakpoint, you can enter the Go command again. The program resumes execution at the instruction after the breakpoint.



# Hex

---

## H *value1 value2*

Performs hexadecimal arithmetic on *value1* and *value2*. DEBUG first adds *value1* and *value2*, and then subtracts *value2* from *value1*. The sum and difference are displayed on one line.

### Example:

H 19F 10A (ENTER)

causes DEBUG performs the arithmetic and displays the results:

02A9 0095

The sum of 19F and 10A is 02A9 and the difference is 0095.

# Input

---

## I *portaddress*

Inputs and displays one byte from the specified port.

A 16-bit port address is allowed. The displayed byte is in hexadecimal.

### Example:

Suppose you type the following command:

```
I 2F8 ENTER
```

causes DEBUG to input the byte at Port 2F8 and display it. If the byte is 42, DEBUG displays:

```
42
```

# Load

---

## L [*address*[ *drive sector sectorcount*]]

Loads a file into memory.

Before you use the Load command to load a file, the file must have been named either when DEBUG was started or with the Name command. Both procedures format a file-spec properly in the File Control Block at CS:5C.

If you enter the L command without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded. If you enter the L command and specify *address*, DEBUG loads the file beginning at the specified address in memory. In both cases, the file is read from the drive specified in the filespec, or from the default drive if no drive was specified.

If you enter the L command with all parameters specified, absolute disk sectors are loaded. The sectors are loaded from the specified *drive* (0 = Drive A, 1 = Drive B, and so on.). DEBUG begins loading with the specified *sector* and continues until the number of sectors indicated by *sectorcount* have been loaded.

If you enter only an offset for the starting address, the segment indicated by register CS is used.

If you load a file which has an .EXE extension, DEBUG relocates the file to the load address specified in the header of the .EXE file, and ignores any *address* you may have specified with the L command. The header itself is stripped of the .EXE file before it is loaded into memory. Thus the size of an .EXE file on disk differs from its size in memory.

If the file you name with the Name command or specify when DEBUG is started is a .HEX file, then typing the L command with no parameters causes DEBUG to load the file beginning at the address specified in the .HEX file. If the L command includes the *address* option, DEBUG adds the specified *address* to the address found in the .HEX file to determine the start address for loading the file.

**Example:**

Suppose you type the following commands:

```
DEBUG (ENTER)
N FILE.COM (ENTER)
L (ENTER)
```

DEBUG loads the file called FILE.COM from the default disk.

To load only portions of a file or certain sectors from a disk, type a command similar to the following:

```
L 04BA:100 2 0F 6D (ENTER)
```

DEBUG loads 6DH (109) consecutive sectors into memory from Drive C, beginning with absolute sector number 0F (15). The data is placed beginning at address 04BA:0100.

# Move

---

## ***M range address***

Moves the block of memory specified by *range* to the location beginning at *address*.

Overlapping moves, where some locations in the *range* are also in the block beginning at *address*, are always performed without loss of data during the transfer. The addresses in the *range* remain unchanged unless new data is written to them by the move.

If you enter only an offset for the starting address of the *range* or for the *address*, the segment indicated by register DS is used. If you specify an ending address for the *range*, enter only an offset value.

### **Example:**

```
M CS:100 110 CS:500 ENTER
```

causes DEBUG to move the data that is between CS:100 and CS:110 to the memory area beginning at CS:500.

# Name

---

## N *filespec1* [*filespec2* ...]

Assigns a filespec for a later Load or Write command. If you start DEBUG without naming any file to be debugged, then you must enter the N *filespec* command before a file can be loaded. The Name command also assigns filespec parameters to the file being debugged.

Four areas of memory can be affected by the Name command:

DS:5C	File Control Block for file 1
DS:6C	File Control Block for file 2
DS:80	Count of characters
DS:81	All characters typed

A File Control Block (FCB) for the first filespec parameter given to the Name command is set up at DS:5C. If you include *filespec2*, an FCB is set up for it at DS:6C. DS:80 contains the number of characters typed after the letter N in the Name command line. All characters typed after the N are stored beginning at DS:81.



## Examples:

```
N file1.exe (ENTER)
L (ENTER)
N file2.dat file3.dat (ENTER)
G (ENTER)
```

In the above sequence of commands, the Name command sets File1.exe as the filespec for the Load command that follows. After File1.exe is loaded into memory, the Name command is used again, this time to specify the parameters to be used by File1.exe. When the Go command is executed, File1.exe is executed as if File1 File2.dat File3.dat had been entered at the MS-DOS command level.

```
DEBUG prog.com (ENTER)
N param1 param2/C (ENTER)
G (ENTER)
```

In the above example, the Go command executes the file in memory as if PROG param1 param2/C had been typed at the MS-DOS command level.

# Output

---

## **O** *portaddress byte*

Sends the *byte* to the specified *portaddress*.

A 16-bit port address is allowed.



### **Example:**

**O 2F8 4F**

causes DEBUG to output the byte value 4F to Port 2F8.



# Quit

---

## Q

Ends the DEBUG program.

The Quit command exits DEBUG without saving the file you are debugging, and returns to the MS-DOS command level.

### **Example:**

Q **ENTER**

ends DEBUG and returns to the MS-DOS system prompt.

# Register

---

## R [*registername*]

Performs three functions:

- Displays the contents of all registers and the flag settings
- Displays the contents of a single register and lets you change the contents
- Displays the flag settings and lets you change the settings

If you enter R with no *registername*, DEBUG displays the contents of all registers and flags, together with the next instruction to be executed.

If you include a register name, DEBUG displays the 16-bit value of that register in hexadecimal, and then displays a colon prompt. You then either change the contents of the register by entering a 1- to 4-character hexadecimal value, or leave the contents unchanged by pressing **ENTER**. The valid *registernames* are:

AX	BP	SS
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

Both IP and PC refer to the Instruction Pointer.

If you enter F as the *registername*, DEBUG displays a two-letter status code for each flag, showing whether it is set or clear. To change any flag, enter the opposite code.

The flags are listed below with their codes for set and clear:

<b>Flag Name</b>	<b>Set</b>	<b>Clear</b>
Overflow (yes/no)	OV	NV
Direction (decrement/increment)	DN	UP
Interrupt (enable/disable)	EI	DI
Sign (negative/positive)	NG	PL
Zero (yes/no)	ZR	NZ
Auxiliary carry (yes/no)	AC	NA
Parity (even/odd)	PE	PO
Carry (yes/no)	CY	NC

Whenever you enter the RF command, DEBUG displays the flags in the order shown above at the beginning of a line. The hyphen prompt (-) appears at the end of the line. Now you can change any of the flag values by typing alphabetic pairs, in any order. You need not leave spaces between the flag entries. To exit the R command, press **ENTER**; the changes are made. Flags for which you did not type a new value remain unchanged.

## Examples:

Type

R **(ENTER)**

DEBUG displays all registers, flags, and the next instruction to be executed. For example, if the location is CS:11A, the display looks similar to this:

```
AX = 0E00 BX = 00FF CX = 0007 DX = 01FF
SP = 039D BP = 0000
SI = 005C DI = 0000 DS = 04BA ES = 04BA
SS = 04BA CS = 04BA
IP = 011A NV UP DI NG NZ AC PE NC
04BA:011A CD21 INT 21
```

If you type

RF **(ENTER)**

DEBUG displays the flags

```
NV UP DI NG NZ AC PE NC - -
```

Type one or more flag designations, in any order, with or without intervening spaces. Example:

```
NV UP DI NG NZ AC PE NC - PLEICY(ENTER)
```

DEBUG makes the requested changes.

If you type

R AX **(ENTER)**

DEBUG displays the contents of the single register AX:

```
AX 0E00
:-
```

To change the contents to 00FF, simply enter FF after the colon prompt.

# Search

---

## *S range list*

Searches the locations in the *range* for the *list* of bytes.

The *list* may contain one or more bytes, each separated by a space or comma. DEBUG displays the starting address for each match found.

If no addresses are displayed, the *list* was not found.

If you enter only an offset for the starting address of *range*, the segment indicated by register DS is used.

### Example:

```
S CS:100 110 41 ENTER
```

causes DEBUG to search the addresses from CS:100 to CS:110 for 41H. If two matches are found, DEBUG displays a response similar to this:

```
04BA:0104  
04BA:010D
```

# Trace

**T**[ = *address*][ *value*]

Executes one or more instructions, displaying the register contents, flags, and next instruction after each instruction executes.

If you enter T with no parameters, DEBUG executes the instruction at CS:IP (the current instruction) and displays the registers and flags. If you enter the optional = *address*, tracing begins at the specified address. The optional *value* causes DEBUG to execute and trace the number of instructions specified by *value*.

When tracing more than one instruction, remember that you can suspend the display by pressing **CTRL** (**S**) in order to study the registers and flags for any instruction. Press any other key to continue scrolling.

## Examples:

T **(ENTER)**

causes DEBUG to display the registers and flags for the current instruction. If the current instruction is 04BA:011A, DEBUG might display:

```
AX=0E00 BX=00FF CX=0007 DX=01FF
  SP=039D BP=0000
SI=005C DI=0000 DS=04BA ES=04BA SS=0
  4BA CS=04BA
IP=011A NV UP DI NG NZ AC PE NC
04BA:011A  CD21      INT      21
```

If you type

T=011A 10 **(ENTER)**

DEBUG executes 16 (10 hex) instructions beginning at 011A in the current segment and displays the registers and flags after each instruction.



# Unassemble

---

U [*address*]

U [*range*]

Disassembles instructions and displays their addresses, their hexadecimal values, and the source statements that correspond to them.

The display of disassembled code looks like a listing for an assembly-language source file. If you enter the U command without parameters, 32 bytes are disassembled at the first address following the last instruction disassembled by the previous Unassemble command. If you specify *address*, instructions are disassembled beginning with *address*.

If you enter the U command with *range* specified, DEBUG disassembles all bytes in the range.

In all cases, DEBUG may disassemble and display slightly more bytes than the default amount or the number you requested. This is because instructions are of varying lengths, and the last instruction disassembled may include more bytes than expected.

If you enter only an offset for the starting address, the segment indicated by register CS is used.

If you have altered some locations using DEBUG, you can enter the U command for the changed locations, view the new instructions, and use the disassembled code to edit the source file.

**Example:**

U04BA:100 L10 (ENTER)

causes DEBUG to disassemble 16 bytes beginning at address 04BA:0100 and displays information similar to the following:

04BA:0100	206472	AND	[SI+72],AH
04BA:0103	69	DB	69
04BA:0104	7665	JBE	016B
04BA:0106	207370	AND	[BP+DI+70],DH
04BA:0109	65	DB	65
04BA:010A	63	DB	63
04BA:010B	69	DB	69
04BA:010C	66	DB	66
04BA:010D	69	DB	69
04BA:010E	63	DB	63
04BA:010F	61	DB	61

# Write

---

**W** [*address* [*drive sector sectorcount*]]

Writes the data being debugged to a disk file.

If you enter the W command with no parameters, the file is written beginning from CS:100. If you enter it with only the *address* parameter, the W command writes the file beginning at that address. In either case, you must be certain that BX:CX contains the number of bytes to write. This value may have been set correctly by the DEBUG or Load commands, but a Go or Trace command may have altered it. (Note that if you load and modify a file, the name, length, and starting address are already set correctly to save the modified file as long as the length has not changed.)

The file must have been named either with the DEBUG startup command or with the Name command. Both commands format a filespec properly in the File Control Block at CS:5C. DEBUG writes the file to the drive specified in the filespec or to the default drive if none is specified.

If you specify all parameters in the Write command, the write begins from the memory *address* specified. The file is written to the specified *drive* (0 = Drive A, 1 = Drive B, and so on). DEBUG writes the file beginning at the sector specified by *sector*, until the number of sectors specified by *sectorcount* have been written.

If you enter only an offset for the starting address, the segment indicated by register CS is used.

If a disk write error occurs, DEBUG displays a message. To try the write operation again, press **(F3)** to redisplay the Write command, and then press **(ENTER)**.

**Note:** Be very careful when you write to absolute sectors on the disk. Data which was previously in these sectors is destroyed.

**Example:**

W CS:100 1 37 2B **ENTER**

writes the contents of memory to the disk in Drive B, beginning with memory address CS:100. The data is written beginning at relative sector 37H and consists of 2BH sectors. The DEBUG prompt is displayed when the write is complete.



## Error Messages

During a DEBUG session, you may receive any of the following error messages. Each error ends the DEBUG command under which it occurred, but does not end DEBUG itself.

### **BF (Bad flag)**

You tried to alter a flag, but the characters entered were not one of the allowable pairs of flag values. See the explanation of the Register command for the list of valid flag entries.

### **BP (Too many breakpoints)**

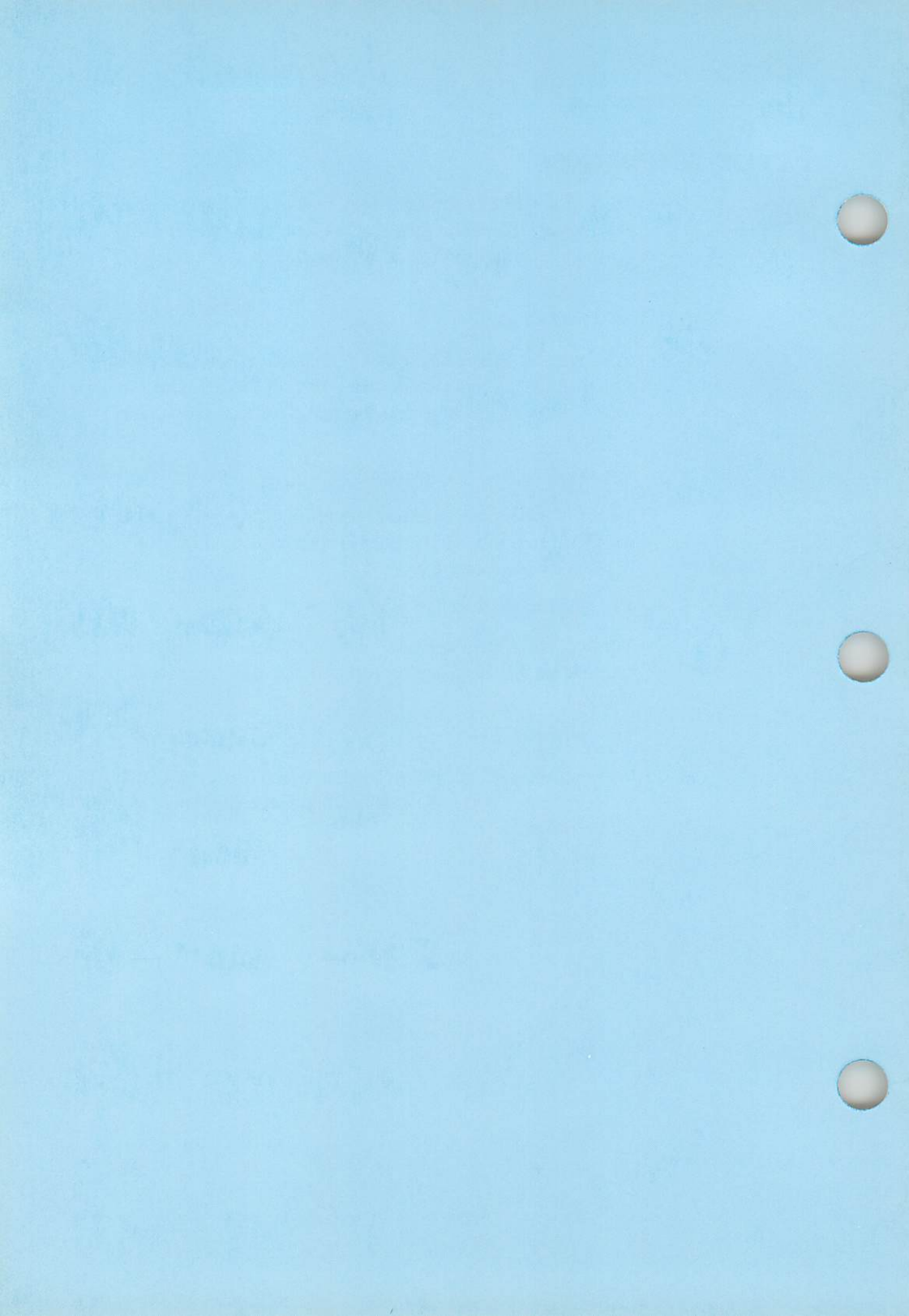
You specified more than ten breakpoints as parameters to the Go command. Retype the Go command again with ten or fewer breakpoints.

### **BR (Bad register)**

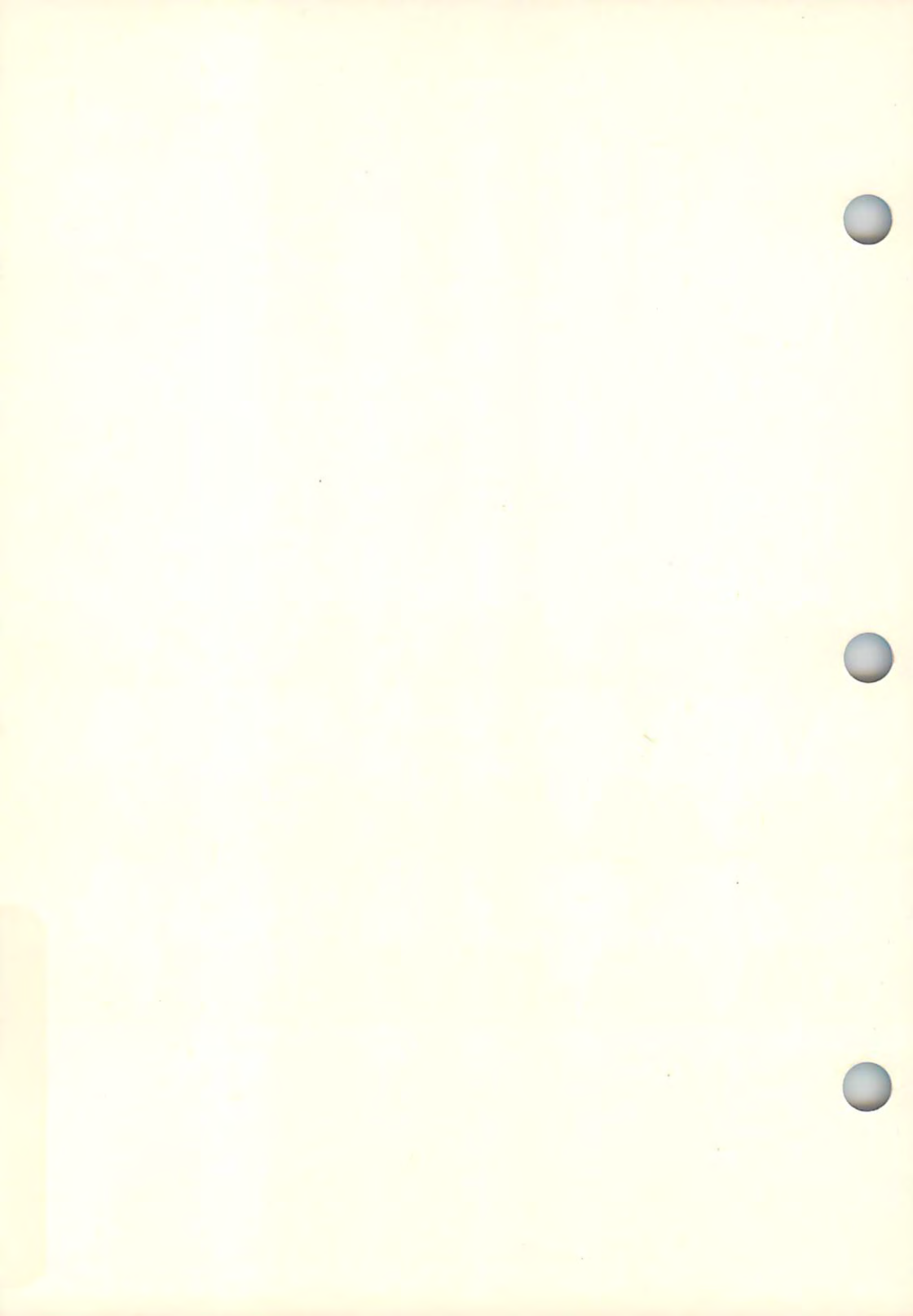
You typed the Register command with an invalid register name. See the Register command for the list of valid register names.

### **DF (Double flag)**

You entered two values for one flag when using the Register command. You may specify a flag value only once per RF command.









# Appendix A

---

## Problems and Error Messages

This appendix consists of two parts — the “Device Error Messages” and the “Command Error Messages.”

Device errors are errors that occur while MS-DOS is writing to a disk or another system device (such as a printer). All other errors, except those given by a special program (such as the linker), are MS-DOS or command errors.

If your computer displays a message not listed in this appendix, the error may be from your application program. See your application program manual.

**Note:** Error messages for the Linker, the Editor and DEBUG are included in those sections of this manual.

## Device Error Messages

If a device error occurs at any time during a command or program, MS-DOS returns an error message in either of the following formats:

*type* error while reading on drive *drive*  
Abort,Ignore,Retry:

*type* error while writing on drive *drive*  
Abort,Ignore,Retry:

*drive* is the drive in which the error occurred.

*type* is one of the following error types:

### **Bad call format**

The device driver was passed an incorrect length request header. Contact the dealer from whom you purchased the device driver.

### **Bad command**

MS-DOS issued an invalid command for the device driver. Try the command again. If it continues to fail, contact the dealer from whom you bought the device driver.

### **Bad unit**

A device driver passed an invalid sub-unit number. Contact the dealer from whom you purchased the device driver.

### **Data**

The data cannot be read or written correctly. You probably have a flawed disk.

### **Disk**

The error is of a type other than those listed above.

### **No paper**

The printer is out of paper.

### **Non-DOS disk**

The disk is not in a format recognizable by MS-DOS.

### **Not ready**

The device is not ready. Check to see that the drive latches are closed and that all devices are on and ready.

### **Read fault**

The device cannot successfully read the data. You may have a flawed disk. Try to copy all files to another disk. If this fails, you probably have a hardware problem. Contact your dealer.

### **Sector not found**

MS-DOS is requesting a sector number higher than the highest number on the disk.

### **Seek**

The disk drive or hard disk cannot locate the proper track on the disk.

### Write fault

The device cannot successfully write the data. You may have a flawed disk. Try to copy all files to another disk. If this fails, you probably have a hardware problem. Contact your dealer.

### Write protect

You tried to write to a write-protected diskette. Remove the write-protect tab, then try again.

**After displaying a device error message, MS-DOS waits for you to enter one of the following responses:**

- A** Abort. Terminate the program requesting the disk read or write.
- I** Ignore. Ignore the bad sector and pretend the error did not occur.
- R** Retry. Repeat the operation. Use this response only after correcting the error (such as a Not Ready or Write Protect error).

Attempt recovery by entering responses in this order:

1. R (to try again)
2. A (to terminate the program and try a new disk)

One other error message might be related to faulty disk read or write:

### File allocation table bad for drive *drive*

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly the disk was incorrectly formatted or not formatted before use. If this error persists, the disk is currently unusable and must be formatted prior to use.

# MS-DOS and Command Errors

## A

### **After format, one of system sectors could not be read**

FORMAT. The system sectors (boot, file allocation table, and directory) are required for the disk to be useful.

### **All files canceled by operator**

PRINT. This is a reminder only. You used the /T parameter to cancel the printing of all files in the print queue.

### **Allocation error, size adjusted**

CHKDSK. A filename is displayed with this message. The File Allocation Table (FAT) contains an invalid sector number. CHKDSK automatically truncates the file at the end of the last valid sector number.

## B

### **Bad command or filename**

Commands. The command is not valid. Check spellings and re-enter the command. If the command is an external command, make sure it is in the directory that MS-DOS is searching for commands. If you are trying to execute a batch file, make sure you are in the directory that contains that file.

### **Bad switch syntax**

COMPDUPE. You used an invalid switch syntax (such as /S only) or a switch other than /D or /S.

## C

### **Cannot CHDIR to filename**

### **Tree past this point not processed**

CHKDSK. This error is corrected automatically.

### **Cannot CHDIR to root**

### **Processing cannot continue**

CHKDSK. The disk you are checking is bad. Restart the system and try using RECOVER command.

## **D**

### **Destination disk format error**

COMPDUPE. An error occurred during formatting. The disk may be flawed or you may have inserted it in the drive incorrectly.

### **Destination disk read error**

COMPDUPE. If you are duplicating, this error causes MS-DOS to abort COMPDUPE and to return to the command ready prompt. If you are comparing, it changes the track status indicator to the appropriate character (S, F, D, or C) and does not display an error message.

### **Destination disk write error**

COMPDUPE. If you are duplicating, this error causes MS-DOS to abort COMPDUPE and to return to the command ready prompt. If you are comparing, it changes the track status indicator to the appropriate character (S, F, D, or C) and does not display an error message.

### **Disk error reading FAT**

CHKDSK. Use the COPY command to copy all files to another disk.

### **Disk error writing FAT**

CHKDSK. Use the COPY command to copy all files to another disk.

### **Disks do not compare**

COMPDUPE.

### **Drive not ready**

PRINT. If this message occurs when PRINT attempts a disk access, PRINT keeps trying until the drive is ready.

## **E**

### **Errors found, F parameter not specified Corrections will not be written to disk**

CHKDSK. Specify the /F switch to correct the errors.

### **Error writing boot sector to destination**

FORMAT. All floppy disks — both data and system disks — must have a boot sector so that they may be used by the system.

### **Errors writing to the system sectors, cannot continue**

FORMAT. The system sectors (boot, file allocation table, and directory) are required for the disk to be useful.

## **F**

### **File canceled by operator**

PRINT. This is a reminder only. You used the /C parameter to cancel the current file in the print queue.

### **File cannot be converted**

EXE2BIN. The source file is not in the correct format. CS:IP does not meet either criterion discussed in the EXE2BIN command. CS:IP meets the .com file criterion but has segment fixups. The file is not a valid executable file.

### **File cannot be copied onto itself n File(s) copied**

COPY. You tried to copy a file to a file that has the same name. Either change the name of your copy, or put it in a different directory or on a different disk.

### **File creation error**

Commands. You tried unsuccessfully to add a file to the directory. You either tried to exceed the limit of 110 files per directory or the directory does not contain enough space for the file. Run CHKDSK to determine if the directory is full or if some other condition caused the error.

### **File not found**

MS-DOS and commands. The specified file does not exist in the specified directory or the current directory. Use the DIR command to see which directory the file is in. Also, check the spelling of the filename.

### **filename is cross linked on cluster**

CHKDSK. This message names each of the two files in error. Copy both files then delete both cross-linked files (the original files).

### **\*\*\* Files are different \*\*\***

FC. No matches are found after the first mismatch.

### **First cluster number is invalid entry truncated**

CHKDISK. This error is corrected automatically.

### **Fixups needed - base segment (hex):**

EXE2BIN. The source (.exe) file contains information that indicates that a load segment is required for the file. Specify the absolute segment address at which the finished module is to be located.

## **H**

### **Has invalid cluster, file truncated**

CHKDSK. The file contains an invalid pointer to the data area. CHKDSK corrects this error automatically by truncating the file to the last valid data block.

## **I**

### **Incompatible system size**

SYS. The system files IO.SYS and MSDOS.SYS do not take up the same amount of space on the target disk as the new system requires.

### **Insufficient memory**

Commands. There is not enough available memory to run the command. Copy some files to another disk and try again.

### **Insufficient room in root directory Erase files in root and repeat CHKDSK**

CHKDSK. CHKDSK cannot process until you delete files in the root directory.

### **Invalid current directory Processing cannot continue**

CHKDSK. Restart the system and re-run CHKDSK.

### **Invalid date Enter new date:**

DATE. Enter a valid date, making sure you use hyphens (-) or slashes (/) to separate the parts of the date.

### **Invalid drive specification**

Commands. Use a valid drive specification, making sure you include the colon (:).

### **Invalid parameter**

Commands. You specified a *parameter* that does not exist. Check the command syntax in this manual.

### **Invalid sub-directory entry**

CHKDSK. This error is corrected automatically.

### **Invalid number of parameters**

Commands. Re-enter the command, including the correct number of parameters.



### **Invalid time**

#### **Enter new time:**

TIME. Check the command syntax and parameters in this manual. Then enter a valid time, making sure you use a colon (:) to separate hours, minutes, and seconds, and a period (.) to separate hundredths of a second.

### **L**

#### **Label not found**

GOTO. The specified label does not exist in the batch file.

#### **List output is not assigned to a device**

PRINT. The device specified as the PRINT output device is invalid.

#### **x lost clusters found in y chains**

#### **Convert lost chains to files (Y/N)?**

CHKDSK. If you type Y (**ENTER**), CHKDSK creates a directory entry and a file in which you can resolve this problem (files created by CHKDSK are named FILE $nnnn$ .CHK). CHKDSK displays: *X bytes disk space freed*. If you have not specified the /F switch, CHKDSK frees the clusters and displays: *X bytes disk space would be freed*.

### **N**

#### **No files match pathname**

PRINT. The files you tried to add to the queue do not exist.

#### **No room for system on destination disk**

SYS. If the target disk contains files you don't need, use the ERASE command to delete as many as necessary to make room for the system. Otherwise, use a different disk as the target.

### **P**

#### **PRINT queue is empty**

PRINT. There are no files in the print queue.

### **PRINT queue is full**

PRINT. The queue can contain no more than 10 files.

### **Probable non-DOS disk**

#### **Continue (Y/N)?**

CHKDSK. You are not using an MS-DOS disk. Type **Y** (**ENTER**) to continue processing or **N** (**ENTER**) to stop processing.

### **Read error in *pathname***

FC. FC could not read the entire file.

**S**

### **Source disk read error**

COMPDUPE. If you are duplicating, this error causes MS-DOS to abort COMPDUPE and to return to the command ready prompt. If you are comparing, it changes the track status indicator to the appropriate character (S, F, D, or C) and does not display an error message.

### **Syntax error**

Commands. Re-enter the command, using the proper syntax.

**U**

### **Unrecoverable error in directory**

#### **Convert directory to file (Y/N)?**

CHKDSK. If you type **Y** (**ENTER**), CHKDSK converts the bad directory into a file. You can then fix the directory or delete it. If you type **N** (**ENTER**), the directory becomes unusable, and you can neither fix nor delete it. You should always type **Y** (**ENTER**).

### **Use HFORMAT to format hard disks**

FORMAT. You are trying to use the FORMAT command to format a hard disk. Use HFORMAT instead.

## **W**

### **Warning-directory full**

RECOVER. The disk does not contain enough directory space to recover more files. Copy some files to another disk, then erase the originals. Run RECOVER again.




### **WARNING -Read error on EXE file Amount read less than size in header**

EXE2BIN. This is a warning message only.

# Appendix B

---

## Command Quick Reference

BACKUP	Copies information from hard disk to floppy disk (diskette)	
BREAK	Sets the <b>CTRL C</b> check	
CHDIR (CD)	Changes the current directory and displays it (CD)	
CHKDSK	Scans the directory of the current or specified drive and checks for consistency	
CLS	Clears the screen	
COMPDUPE	Copies the diskette in Drive A onto the diskette in Drive B and compares the diskettes	
COPY	Copies the specified file(s)	
CTTY	Changes the input/output device	
DATE	Displays and sets the date	
DEL (ERASE)	Deletes the specified file(s)	
DIR	Lists the requested directory entries	
DISKCOPY	Copies a disk	
ECHO	Turns the batch file echo feature on or off	
ERASE (DEL)	Deletes the specified file(s)	
EXE2BIN	Converts executable files to binary format	
EXIT	Exits a command and returns to the lower level	
FC	Compares the contents of two files	
FIND	Searches for a constant string of text	

FOR	Executes a command for each item in a set
FORMAT	Formats a floppy disk to receive MS-DOS files
GOTO	Transfers control to a certain line in a batch file
IF	Allows conditional execution of commands in batch file processing
HFORMAT	Formats a hard disk to receive MS-DOS files
MKDIR (MD)	Makes a directory
MORE	Displays output one screen at a time
PATH	Sets a command search path
PAUSE	Suspends execution of a batch file
PRINT	Lets you print while processing other MS-DOS commands
PROMPT	Sets a new system prompt
RECOVER	Recovers a bad disk
REM	Displays a comment in a batch file
RENAME (REN)	Renames a file
RESTORE	Copies backed up files from floppy disk to hard disk
RMDIR (RD)	Removes a directory
SET	Sets one string value to another
SHIFT	Increases the number of replaceable parameters in a batch process
SORT	Sorts data alphabetically, forward or backward


SYS	Transfers MS-DOS system files from Drive A to the drive specified
TIME	Displays and sets the time
TYPE	Displays the contents of the specified file
VER	Prints the MS-DOS version number
VERIFY	Verifies writes to disk
VOL	Displays the volume identification number



# Appendix C


---

## How to Configure Your System



In many cases, there are installation-specific settings for MS-DOS that need to be configured at system startup. An example of this is a standard device driver, such as an on-line printer.

The MS-DOS configuration file (CONFIG.SYS) lets you configure your system with little effort. With this file, you can add device drivers to your system at startup. The configuration file is simply an ASCII file that has certain commands for MS-DOS startup. The startup process is as follows:


1. The disk boot sector is read. The boot sector contains enough code to read MS-DOS code and the installation's BIOS (machine-dependent code).
  2. The MS-DOS code and BIOS are read.
  3. A variety of BIOS initializations are done.
  4. A system initialization routine reads CONFIG.SYS, if it exists, to perform a device installation and other user options. Its final task is to execute the command interpreter, which finishes startup.
- 

## Changing the CONFIG.SYS File

If there is not a CONFIG.SYS file on the MS-DOS disk, use the editor to create the file and save it in the root directory of the MS-DOS disk.

The following is a list of commands for the CONFIG.SYS file:

### **Buffers = *number***



Sets the *number* of sector buffers that will comprise the system list. It is installation-dependent. If the *number* is not set, the system uses 2. We recommend that you allocate at least 5 buffers. Keep in mind that each uses approximately 512 bytes of memory.

**Files = *number***

Sets the *number* of open files that the XENIX-compatible system calls can access. It is installation-dependent. If the *number* is not set, **10** is a reasonable number to use.

**Device = *filename***

Installs the device driver in the file, which is specified by pathname, into the system list. (See below.)

**Break = [ON|OFF]**

Sets MS-DOS to check for the **CTRL C** as input every time MS-DOS is called, if you specify ON. ON improves the ability to abort programs. OFF is the default.

**Shell = *pathname***

Begins execution of the shell (the top-level command processor) from the file specified by *pathname*.

A typical CONFIG.SYS file might look like this:

```
Buffers      = 10
Files        = 10
Device       = \BIN\network.sys
Break        = ON
Shell        = A:\BIN\command.com A:\BIN /P
```

Note here that the Buffers= and Files= parameters are set to 10. The system initialization routine will search for the file \BIN\Network.sys to find the device that is being added to the system. This file is usually supplied on disk with your device. Make sure that you save the device file in the pathname that you specify with the Device= parameter.

This configuration file also sets the MS-DOS command EXEC to the Command.com file located in A:\BIN. The A:\BIN tells Command.com where to look for itself when it needs to re-read from disk. The /P tells Command.com that it is the first program running on the system so that it can process the MS-DOS EXIT command.



# Appendix D

---

## ASCII and Scan Codes

The following table lists the keys, in scan code order, and the ASCII codes generated by each (which depends on the shift status). The entries in the table are:

- SCAN CODE — a value in the range 01H-5AH (hexadecimal) which uniquely describes which key is pressed.
- KEYBOARD LEGEND — the physical marking(s) on the key. If multiple markings exist, they are listed from top to bottom.
- NORMAL — the normal (unshifted) ASCII value (returned when only the indicated key is pressed).
- SHIFT — the shifted ASCII value (returned when **SHIFT** is also pressed).
- CTRL — the control ASCII value (returned when **CTRL** is also pressed).
- ALT — the alternate ASCII value (returned when **ALT** is also pressed).
- REMARK — any remarks or special functions.

All numerical values in the table are expressed in **hexadecimal**. Those values preceded by an “x” are extended ASCII codes (they are preceded by an ASCII NUL (= 00)).

A marking of -- indicates that no ASCII code is generated. A marking of \*\* indicates that no ASCII code is generated and, instead, the special function described in the REMARK column is performed.

SCAN CODE	KEYBOARD LEGEND	NORMAL	ASCII SHIFT	CTRL	ALT	REMARK
00	(none / invalid scan code)					
01	ESC	1B	1B	1B	x8B	
02	! 1	31	21	--	x78	
03	@ 2	32	40	x03	x79	
04	# 3	33	23	--	x7A	
05	\$ 4	34	24	--	x7B	
06	% 5	35	25	--	x7C	
07	^ 6	36	5E	1E	x7D	
08	& 7	37	26	--	x7E	
09	* 8	38	2A	--	x7F	
0A	( 9	39	28	--	x80	
0B	) 0	30	29	--	x81	
0C	← -	2D	5F	1F	x82	
0D	+ =	3D	2B	--	x83	
0E	BACKSPACE	08	08	7F	x8C	
0F	TAB	109	x0F	x8D	x8E	
10	Q	71	51	11	x10	
11	W	77	57	17	x11	
12	E	65	45	05	x12	
13	R	72	52	12	x13	
14	T	74	54	14	x14	
15	Y	79	59	19	x15	
16	U	75	55	15	x16	
17	I	69	49	09	x17	
18	O	6F	4F	0F	x18	
19	P	70	50	10	x19	
1A	{ [	5B	7B	1B	--	
1B	} ]	5D	7D	1D	--	
1C	ENTER	0D	0D	0A	x8F	(main keyboard)
1D	CTRL	**	**	**	**	CONTROL MODE
1E	A	61	41	01	x1E	
1F	S	73	53	13	x1F	
20	D	64	44	04	x20	
21	F	66	46	06	x21	
22	G	67	47	07	x22	
23	H	68	48	08	x23	

SCAN CODE	KEYBOARD LEGEND	ASCII				REMARK
		NORMAL	SHIFT	CTRL	ALT	
24	J	6A	4A	0A	x24	
25	K	6B	4B	0B	x25	
26	L	6C	4C	0C	x26	
27	: ;	3B	3A	--	--	
28	" '	27	22	--	--	
29	↑	x48	x85	x90	x91	
2A	SHIFT	**	**	**	**	(left) SHIFT
2B	←	x4B	x87	x73	x92	
2C	Z	7A	5A	1A	x2C	
2D	X	78	58	18	x2D	
2E	C	63	43	03	x2E	
2F	V	76	56	16	x2F	
30	B	62	42	02	x30	
31	N	6E	4E	0E	x31	
32	M	6D	4D	0D	x32	
33	< ,	2C	3C	--	--	
34	> .	2E	3E	--	--	
35	?/	2F	3F	--	--	
36	SHIFT	**	**	**	**	(right) SHIFT
37	PRINT	10	**	x72	x46	PRINT SCREEN TOGGLE
38	ALT	**	**	**	**	ALTERNATE MODE
39	SPACEBAR	20	20	20	20	
3A	CAPS	**	**	**	**	CAPS LOCK
3B	F1	x3B	x54	x5E	x68	
3C	F2	x3C	x55	x5F	x69	
3D	F3	x3D	x56	x60	x6A	
3E	F4	x3E	x57	x61	x6B	
3F	F5	x3F	x58	x62	x6C	
40	F6	x40	x59	x63	x6D	
41	F7	x41	x5A	x64	x6E	
42	F8	x42	x5B	X65	x6F	
43	F9	x43	x5C	x66	x70	
44	F10	x44	x5D	x67	x71	
45	NUM LOCK	**	**	**	**	NUMBER LOCK
46	HOLD	**	**	**	**	FREEZE DISPLAY
47	\ 7	37	5C	x93	**	†

SCAN CODE	KEYBOARD LEGEND	ASCII				REMARK
		NORMAL	SHIFT	CTRL	ALT	
48	~ 8	38	7E	x94	**	†
49	PG UP 9	39	x49	x84	**	†
4A	↓	x50	x86	x96	x97	
4B	4	34	7C	x95	**	†
4C	5	35	--	--	**	†
4D	6	36	--	--	**	†
4E	→	x4D	x88	x74	**	SMOOTH SCROLL TOGGLE
4F	END 1	31	x4F	x75	**	†
50	2	32	60	x9A	**	†
51	PG DN 3	33	x51	x76	**	†
52	0	30	x9B	x9C	**	†
53	DELETE	x53	x8A	x9D	x9E	
54	BREAK	x00	x00	**	x00	MS-DOS BREAK (INT1BH)
55	INSERT	x52	x89	x9F	xA0	
56	.	2E	xA1	xA4	xA5	(numeric keypad)
57	ENTER	0D	0D	0A	x8F	(numeric keypad)
58	HOME	x47	x4A	x77	xA6	
59	F11	x98	xA2	xAC	xB6	
5A	F12	x99	xA3	xAD	xB7	

† The **(ALT)** key provides a way to generate the ASCII codes of decimal numbers between 1 and 255. Hold down **(ALT)** while you type **on the numeric keypad** any decimal number between 1 and 255. When you release **(ALT)**, the ASCII code of the number you typed is generated and displayed.

**Note:** When the **(NUM LOCK)** light is off, the NORMAL and SHIFT columns for the numeric keypad keys should be reversed.

# Appendix E

---

## ASCII Character Codes

The table in Appendix D listed the ASCII codes (in hexadecimal) generated by each key. The table in **this** appendix lists the characters generated by those ASCII codes. (Note: All ASCII codes in this table are expressed in **decimal** form.)

You can display the characters listed by doing either of the following:

- Using the BASIC statement `PRINT CHR$(code)`, where *code* is the ASCII code.
- Pressing **(ALT)** and, without releasing it, typing the ASCII code on the **numeric keypad**.

For Codes 0-31, the table also lists the standard interpretations. The interpretations are usually used for control functions or communications.

**Note:** The BASIC program editor has its own special interpretation of some codes and may not display the character listed.

ASCII Code	Character	Control Character
000	(null)	NUL
001	☺	SOH
002	☹	STX
003	♥	ETX
004	♦	EOT
005	♣	ENQ
006	♠	ACK
007	(beep)	BEL
008	■	BS
009	(tab)	HT
010	(line feed)	LF
011	(home)	VT
012	(form feed)	FF
013	(carriage return)	CR
014	🎵	SO
015	☼	SI
016	▶	DLE
017	◀	DC1
018	↕	DC2
019	!!	DC3
020	⌘	DC4
021	§	NAK
022	▬	SYN
023	⤴	ETB
024	⤴	CAN
025	⤵	EM
026	→	SUB
027	←	ESC
028	(cursor right)	FS
029	(cursor left)	GS
030	(cursor up)	RS
031	(cursor down)	US

ASCII Code	Character	ASCII Code	Character
032	(space)	064	@
033	!	065	A
034	"	066	B
035	#	067	C
036	\$	068	D
037	%	069	E
038	&	070	F
039	'	071	G
040	(	072	H
041	)	073	I
042	*	074	J
043	+	075	K
044	,	076	L
045	-	077	M
046	.	078	N
047	/	079	O
048	0	080	P
049	1	081	Q
050	2	082	R
051	3	083	S
052	4	084	T
053	5	085	U
054	6	086	V
055	7	087	W
056	8	088	X
057	9	089	Y
058	:	090	Z
059	;	091	[
060	<	092	\
061	=	093	]
062	>	094	^
063	?	095	_

ASCII Code	Character	ASCII Code	Character
096	`	128	Ç
097	a	129	ü
098	b	130	é
099	c	131	â
100	d	132	ä
101	e	133	à
102	f	134	á
103	g	135	ç
104	h	136	ê
105	i	137	ë
106	j	138	è
107	k	139	ï
108	l	140	í
109	m	141	ì
110	n	142	Ä
111	o	143	Å
112	p	144	É
113	q	145	æ
114	r	146	Æ
115	s	147	ô
116	t	148	ö
117	u	149	ò
118	v	150	û
119	w	151	ù
120	x	152	ÿ
121	Y	153	Ö
122	z	154	Ü
123	{	155	ç
124		156	£
125	}	157	¥
126	~	158	Pt
127	☐	159	f



ASCII Code	Character	ASCII Code	Character
160	á	192	ˆ
161	í	193	˜
162	ó	194	˘
163	ú	195	˙
164	ñ	196	˚
165	Ñ	197	˛
166	ª	198	˜
167	º	199	˘
168	<	200	˙
169	⌈	201	˚
170	⌋	202	˛
171	½	203	˜
172	¼	204	˘
173	ı	205	˙
174	«	206	˚
175	»	207	˛
176	∕	208	˜
177	∕	209	˘
178	∕	210	˙
179		211	˚
180	⊥	212	˛
181	⊥	213	˜
182	⊥	214	˘
183	⊥	215	˙
184	⊥	216	˚
185	⊥	217	˛
186		218	˜
187	⊥	219	˘
188	⊥	220	˙
189	⊥	221	˚
190	⊥	222	˛
191	⌋	223	˜

ASCII Code	Character
224	$\alpha$
225	$\beta$
226	$\Gamma$
227	$\pi$
228	$\Sigma$
229	$\sigma$
230	$\mu$
231	$\tau$
232	$\Phi$
233	$\theta$
234	$\Omega$
235	$\delta$
236	$\infty$
237	$\emptyset$
238	$\epsilon$
239	$\cap$
240	$\equiv$
241	$\pm$
242	$\geq$
243	$\leq$
244	$\rho$
245	J
246	$\div$
247	$\approx$
248	$\circ$
249	$\bullet$
250	$\cdot$
251	$\sqrt{\quad}$
252	n
253	z
254	■
255	(blank 'FF')

# Index

---

## A

Addresses, Assigning 222  
Alignment 217  
Anonymous directory names 21-22  
Application program 7  
    Exiting from 92  
    Returning to 92  
Arguments 25  
ASCII codes 281-285  
ASCII files, Copying 64-74

## B

Background printing 122-125  
BACKUP 50-51  
Batch files 31-39  
    Aborting execution 32  
    Autoexec.bat 34-35  
    Advice about 33  
    Conditional execution of batch command (IF) 113-114  
    Creating 31-32  
    Displaying commands during execution (ECHO) 85-86  
    Executing 32-33  
    Including remarks in (REM) 32, 130  
    Summary of batch file process 34  
    Suspending execution 32, 120-121  
    Reminders about 38-39  
    Transferring control to a specific line (GOTO) 109-110  
    Using replaceable parameters in 35-38  
Binary files, Copying 64-74  
BREAK 52

## C

Canonical frame 218  
CHDIR (CD) 53-55  
CHKDSK 56-59  
Classes 206  
CLS 60  
Combine types 217  
    Common 219  
    Stack 219  
    Private 218  
    Public 219  
COMMAND.COM 41  
Command 9, 23-26, 41-45, 103-104  
    Aborting a command 26, 45  
    Delimiters 26  
    Editing a command 41-45  
    Errors 268-275  
    Executing one for several items (FOR) 103-104  
    Parameters 25  
    Piping commands 29

- Processor 41
- Quick reference list 276-278
- Syntax notation 47-48
- Template 41
- Types of commands 9, 23-24
- CON 13
- CONFIG.SYS 279-280
- Configuring the system 279-280
- Control character keys 45-46
- COMPDUPE 61-63
- COPY 64-68
- COPY/APPEND 69-71
- COPY/COMBINE 72-74
- CITY 75
- Current directory 18-20
- Current drive 17-18

## D

- Data, Sorting 143-144
- DATE 76-77
- DEBUG 229-263
  - ASSEMBLE 235-237
    - Commands 230
  - COMPARE 238
  - DUMP 239
  - ENTER 240-241
  - Error messages 263
  - FILL 242
  - GO 243-244
  - HEX 245
  - INPUT 246
  - LOAD 247-248
  - MOVE 249
  - NAME 250-251
  - OUTPUT 252
  - Parameters 230
  - QUIT 253
  - REGISTER 254-256
  - SEARCH 257
  - Starting 229
  - TRACE 258
  - UNASSEMBLE 259-260
  - WRITE 261-262
- DEL 87-88
- Delimiters 26
- Device 13, 75
  - Changing I/O device 75
  - Errors 265-267
  - Names 13
- DIR 79-81
- Directory 9, 13-22, 53-59, 115-116, 135-136
  - Changing (CHDIR) 19-20, 53-55
  - Checking for errors (CHKDSK) 56-59
  - Creating (MKDIR) 15-16, 115-116
  - Current 18-20
  - Home 20-21
  - Names, Anonymous 21-22
  - Removing (RMDIR) 17, 135-136

Root 9  
Using 13-15

Disk 56-59, 61-63, 79-84, 105-108, 111-112, 128-129, 195  
Checking for errors (CHKDSK) 56-59  
Comparing (COMPDUPE) 61-63  
Displaying information about (DIR) 79-81  
Duplicating (COMPDUPE, DISKCOPY) 61-62, 82-84  
Interleave factor 106, 112  
Labeling 105, 107, 112  
Preparing a floppy disk (FORMAT) 105-108  
Preparing a hard disk (HFORMAT) 111-112  
Recovering a flawed disk (RECOVER) 128-129  
Sector 105  
Skew 105  
Track 105  
Writing lines to disk 195

DISKCOPY 82-84  
Copying a datadisk 83  
Copying a system disk 83

Display  
See Screen

Drive, Current 17-18

## E

ECHO 85-86

Editing keys 42-45  
Deleting a character (**DELETE**) 42  
Deleting to a character (**F4**) 42  
Entering a line (**ENTER**) 43  
Inserting (**INSERT**) 42  
Replacing the template (**F5**) 42  
Voiding a line (**ESC**) 42

EDLIN  
See Line editor

End-of-file (EOF) 64-74

ERASE 87-88  
Using wild card with 87

Error messages 265-275

EXE2BIN 89-91

Executable file, Converting to binary 89-91

EXIT 92

Extension 10-11

External commands 9, 24, 66-67  
Telling MS-DOS where to find (PATH) 66, 118-119  
Transferring to another directory (COPY) 66-67

## F

FC Utility 93-99

File 8, 69-74, 87-91, 93-102, 128-133, 145, 148-151, 154-155, 190-193  
Appending to (COPY/APPEND) 69-71  
Combining files (COPY/COMBINE) 72-74  
Comparison, byte-by-byte (FC) 93-99  
Comparison, line-by-line (FC) 93-99  
Converting executable to binary (EXE2BIN) 89-91  
Copying (COPY) 64-68  
Copying system files (SYS) 145  
Creating files (Line editor) 154-155

- Displaying contents (TYPE) 148-149
- Displaying information about (DIR) 79-81
- Editing (Line editor) 154
- Organization of the file system 8
- Paging through 186
- Recovering a flawed file (RECOVER) 128-129
- Removing (ERASE) 87-88
- Renaming (REN) 131-133
- Saving (Line editor) 155
- Searching for a string (FIND) 100-102, 190-193
- Verifying (VERIFY) 151

Filename 10-11

- Wild cards 11-12

Filespec 25

Filters 28

- See also FIND, MORE, SORT

FIND 28, 100-102

FOR 103-104

FORMAT 105-108

## G

GOTO 109-110

Groups 205

## H

Hard disk 48-51, 111-112, 134

- Backing up files to floppy disk (BACKUP) 50-51
- Entering commands for 48-49
- Preparing (HFORMAT) 111-112
- Restoring files to (RESTORE) 134
- Transferring system files to (HFORMAT) 111-112

Home directory 20-21

## I

IF 113-114

Input, Redirecting from file 27

Interleave factor 106

Internal commands 9, 23-24

I/O device, Changing (CTTY) 75

## L

Line editor 153-198

- Creating files 154-155
- Commands 169-195
- Editing existing files 154
- Editing keys 156-164
- Error messages 195-198
- Functions of 153
- Saving files 155

Line editor commands 165-195

- Append Lines 169
- Copy Lines 170-172
- Delete Lines 173-175
- Edit Lines 176-177
- End Edit 178

- Insert Lines 179-181
- List Lines 182-184
- Move Lines 185
- Page 186
- Quit Edit 187
- Replace String 188-189
- Search Text 190-193
- Transfer Lines 194
- Write Lines 195
- Linker 199-223
  - Canonical frame 218
  - Classes 206
  - Combine types 217
  - Groups 205
  - Relocation fixups 222
  - Response file 213-214
  - Sample session 214-215
  - Segments 205
  - Starting 211-212
  - Switches 209-211
- List file 201
- Long relocation fixups 224
- LST 13

## M

- MKDIR 15-16, 115-116
- MORE 28, 117
- MS-LINK
  - See Linker

## N

- Near self-relative relocation fixups 223
- Near segment-relative relocation fixups 223-224

## O

- Output 27-28, 97
  - Appending to file 28
  - Redirecting to file 27
  - Redirecting to printer 97

## P

- Parameters 25
  - Setting for later use (SET) 137-140
  - Using more than 10 replaceable parameters (SHIFT) 141-142
- PATH 118-119
- Pathname 12-13
- PAUSE 120-121
- Piping 29
- PRINT 122-125
- Printing 45, 122-125
  - All video output 45
  - Background 122-125
  - Current display 45
- Private combine 214
- Programs, Linking
  - See Linker

PROMPT 126-127  
Public combine 215

## R

RECOVER 128-129  
Relocation fixups 222-224  
    Long 224  
    Near segment-relative 223-224  
    Near self-relative 223  
    Short 223  
REM 130  
REN 131-133  
Replaceable parameters, Using more than 10 (SHIFT) 141-142  
RESTORE 134  
RMDIR 135-136  
Root directory 9  
Run file 200

## S

Scan codes 281  
Screen 45, 60, 117  
    Clearing (CLS) 60  
    Displaying one at a time (MORE) 117  
    Printing current display ((SHIFT) (PRINT)) 45  
    Suspending scrolling ((CTRL) (S)) 45  
Sector 105  
Segment 205  
Segment address 221  
SET 137-140  
SHIFT 141-142  
Short relocation fixups 223  
Skew 106  
SORT 28, 143-144  
Source files, Comparing 98  
Stack combine 215  
String, Replacing 188-189  
String, Searching a file for 100-102  
SYS 145  
System prompt, Changing 7, 126-127  
Switches 25

## T

Template 41  
Temporary file (VM.TMP) 202-203  
TIME 146-147  
Track 105  
TYPE 148-149

## V

VERIFY 151  
VERSION 150  
VM.TMP (Temporary file) 202-203  
VOL 152

## W

Wild card filenames 11-12





**RADIO SHACK, A DIVISION OF TANDY CORPORATION**

**U.S.A.: FORT WORTH, TEXAS 76102  
CANADA: BARRIE, ONTARIO L4M 4W5**

---

**TANDY CORPORATION**

---

**AUSTRALIA**  
91 KURRAJONG ROAD  
MOUNT DRUITT, N. S. W. 2770

---

**BELGIUM**  
PARC INDUSTRIEL DE NANINNE  
5140 NANINNE

---

**U. K.**  
BILSTON ROAD WEDNESBURY  
WEST MIDLANDS WS10 7JN