PR1ME

THE NEW USER'S GUIDE
TO EDITOR AND RUNOFF

The New User's Guide to EDITOR and RUNOFF

The New User's
Guide to
EDITOR and RUNOFF

by
Daniel P. Dern

The Purpose of this User Guide*

The purpose of this user guide:
We have endeavoured to provide
Clear explanations, amplified
  By ample demonstrations.

Which show how easily you may
Use Prime computers every day
To help in each and every way
  With office occupations.

You'll Edit and Runoff reports
And then learn how to do all sorts
Of letters, files, and later thoughts
  Of alphabetizations.

This information is all here;
It's organized and very clear.
For further questions, check the rear
  Just read the ref'rence sections.

You never more need be afraid
For in this book we have displayed
Computers as your friend and aid
  Goodbye to trepidations.

*To be sung to the tune of
"How Beautifully Blue the Sky", from
The Pirates of Penzance by W. S. Gilbert
and A. Sullivan

PURPOSE: A TEXTBOOK FOR NEW USERS

You do not need to know anything about computers to read and use this manual.

The NEW USER'S GUIDE TO EDITOR & RUNOFF has been written for the new computer user who may not have had any prior experience in working with computers or

text-processing systems. This book tells you exactly what you need to know to immediately begin using the computer and EDITOR and RUNOFF.

EDITOR is a text-editing system. Using EDITOR, you can type text into the computer, edit it, and save it for later use.

RUNOFF is a formatting system for printed text. With a few simple commands, you can turn your typed input into pages which are neatly arranged in whatever manner you wish.

ORGANIZATION

The contents of this manual are:

Section 1: **INTRODUCTION** (this section). Gives you general information about computers, terminals, and the various conventions used in this manual.

Section 2: **HI–I'M PRIMOS.** Gives you basic information about how to use a Prime Computer.

Section 3: **THE ESSENTIALS OF EDITOR.** Introduces you to EDITOR, and teaches you enough about EDITOR to do most jobs.

Section 4: **MORE PRIMOS.** Additional information about Prime computers, which you will not need until after you have been using EDITOR.

Section 5: **THE ESSENTIALS OF RUNOFF.** Teaches you enough about RUNOFF so that you can use it for a number of standard tasks.

Section 6: **MORE RUNOFF.** Explanations of how to do slightly more complicated or non-standard work using RUNOFF.

Section 7: **RUNOFF DECIMALIZATION.** How to do decimalized headers and tables of contents.

Section 8: **SAMPLE SESSIONS.** Demonstration of common tasks for EDITOR and RUNOFF, plus some advanced techniques.

Section 9: **EDITOR REFERENCE SECTION.** Full information on all the EDITOR commands, in alphabetical order.

Section 10: **RUNOFF REFERENCE SECTION.** Full information on all the RUNOFF commands, in alphabetical order.

Appendix A: **COMMAND SUMMARIES.** Summarizes the PRIMOS, EDITOR, and RUNOFF commands.

We've arranged the material so you can log in to the computer and begin working almost immediately. The further you read, the more you'll be able to do.

Once you've become familiar with the computer and have had some practice using EDITOR and RUNOFF, you should have little trouble learning to use the more advanced techniques and more powerful commands. Then, when you are an experienced user, you can keep the command summaries by your terminal, and turn to the reference sections whenever specific questions arise.

TERMINALS AND COMPUTERS

It is very unlikely that you are near your Prime computer at the moment. The computer is off in the computer room, sitting in its cabinet, alongside a couple of disk drives, tape transports, line printers, terminals, and so on, all under the watchful eye of a computer operator or two. All you need to "talk" with the computer is your terminal, and – maybe – a telephone coupler.

Prime computers can be connected with up to 63 working terminals at the same time.

Some terminals are connected directly to the computer by a length of wire. Other terminals communicate with the computer over the telephone, using small devices called acoustical couplers. These devices translate the computer's and terminal's interaction into signals which can be sent through a phone line.

The top half of Figure 1-1 illustrates those aspects of the computer system which you are likely to see and use – a terminal, telephone and acoustic coupler (depending), and sometimes the line printer.

WHAT YOU DON'T SEE—ONE WAY TO THINK ABOUT COMPUTERS

As far as you're concerned, the computer is a large black box which always behaves in the manner described in this book. The bottom half of Figure 1-1 shows one way you can picture what the computer does.

Think of the computer as a company which rents a desk space plus office services. One section of the office is a storage area containing a virtually unlimited number of file cabinets, plus a well-stocked shelf containing office equipment.

The office is run by an office manager, whose name is PRIMOS. There is a specific list of jobs which PRIMOS will do for you; PRIMOS will not understand anything you say that is not on this list. These jobs are explained in manuals such as this one. There are two types of jobs: those which have PRIMOS perform a service for you, and those which have PRIMOS get you something from the supply shelf.

The supply shelf contains the office equipment available to all users. Unlike real offices, there is never any problem with getting a particular piece of equipment at the time you ask for it, no matter how many other people are using one at the same time. This manual explains how to use the "pieces of equipment" called EDITOR and RUNOFF; other manuals explain other "equipment."

People who wish to work in this office identify themselves by giving a piece of information called a user-name. Many people may know a particular user-name; some people will know several different user-names. With some of these user-names you will have to give a password; this prevents unauthorized people from using these names.

There is a labeled file drawer for every user-name. In your drawer are folders; each folder contains a file. Your drawer is always large enough to hold all the files you need. PRIMOS enables you to label each folder.

PRIMOS keeps a notebook, containing a section for every user. When you ask PRIMOS for a desk to use, PRIMOS flips through this notebook to make sure you are a valid account. If so, PRIMOS assigns you to the first available desk, signs you in, and drags over
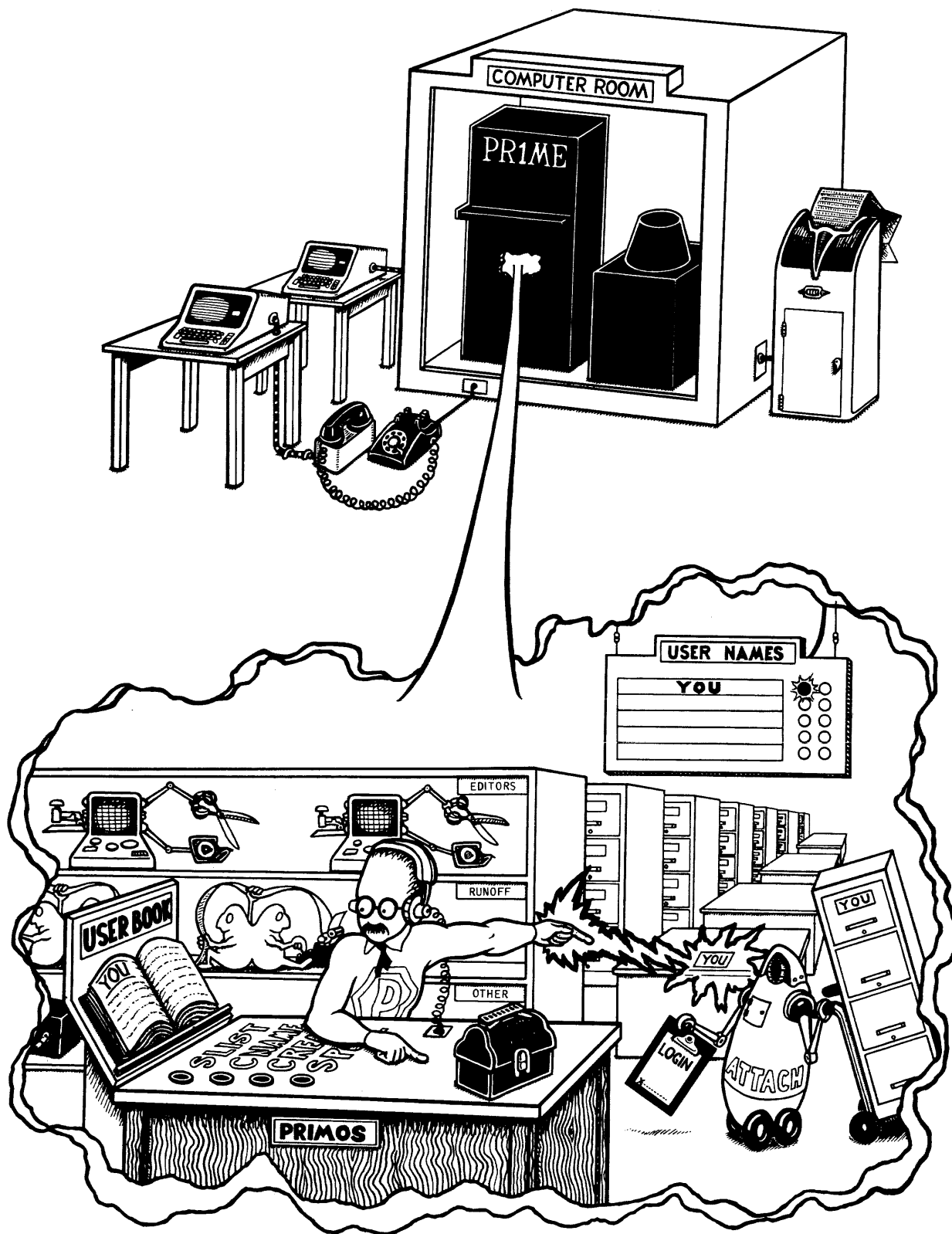
Figure 1-1. One way to think about computers

your file drawer. PRIMOS will also get you any other file drawer which you are entitled to use. Any number of people can work out of a file drawer at the same time, although sometimes only one person can work on a specific file at a time.

You can do any number of things at your desk by using the office equipment: make new files, change them, do computations, move files around, look at files, run various programs. All you do is tell PRIMOS what you want to do, and PRIMOS will take care of the precise details.

HOW COMPUTERS THINK

Computers are machines. They are complicated machines, full of large quantities of little widgets – but just as you don't have to know how an internal combustion engine works to drive a car, you don't need to understand how the computer works, or why; what you do need to know is the rules of the road – how to use it.

That's what this book does – it's a road map/instruction handbook to get you where you want to go.

Like any other machine, computers do not really "think", they just follow orders and do EXACTLY WHAT YOU TELL THEM TO DO. This means you have to tell the computer PRECISELY what you want. If you make a mistake that still can be understood, it will follow the order anyway. Computers do exactly what you want to do, in terms which the computer can understand. That is what this manual will teach you. Since the computer follows your orders precisely, you obviously want to avoid making mistakes.

To help you do this, the computer checks everything you tell it to make sure you have given it reasonable instructions, – so far as it can tell. It knows what each possible command is supposed to look like, in terms of spelling, permissible abbreviations, and, occasionally, whether the data associated with the command is of the appropriate type.

You even have a chance to correct typing errors and redo entire lines before they are read. If for some reason the system is not happy with what you have told it, it will inform you by printing out one or more error messages. These messages are clear, and you will have no trouble figuring out what has happened. If you are confused, check with your System Administrator.

What the computer cannot determine, however, is whether what you said is what you meant to say. Always keep in mind:

- *Computers have no common sense.* No matter how strange your request may be, if the system can do it, it will.

- *Computers are always consistent.* If the circumstances are the same, the computer will always react exactly the same way to the same command (or mistake).

- *Nothing breaks when you make mistakes.* Computers are built to be used by people. It's not hard to make little mistakes. It is possible, though difficult, to scramble your own files. However, it is very difficult to do something that cannot be undone, and it is impossible to do anything at your terminal that actually harms the computer.

So, with a little practice, you'll be able to analyze what it is you want to do and then determine how to make the computer do it.

Computers have no common sense
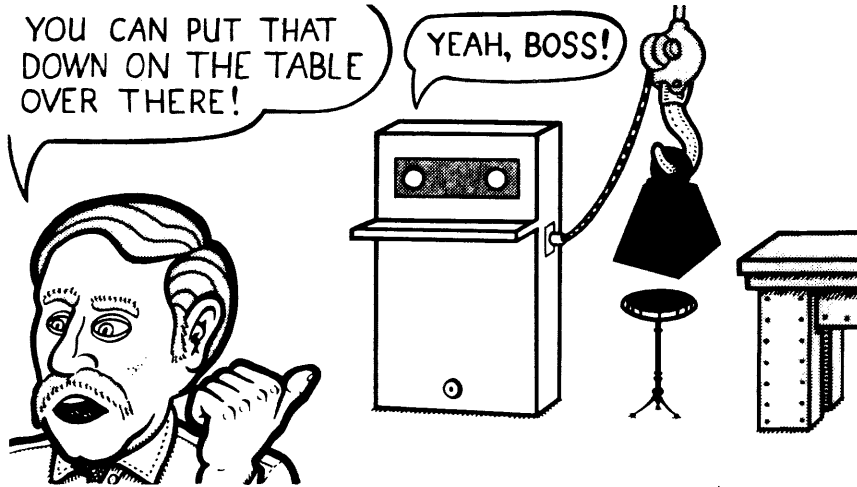
Computers are consistent
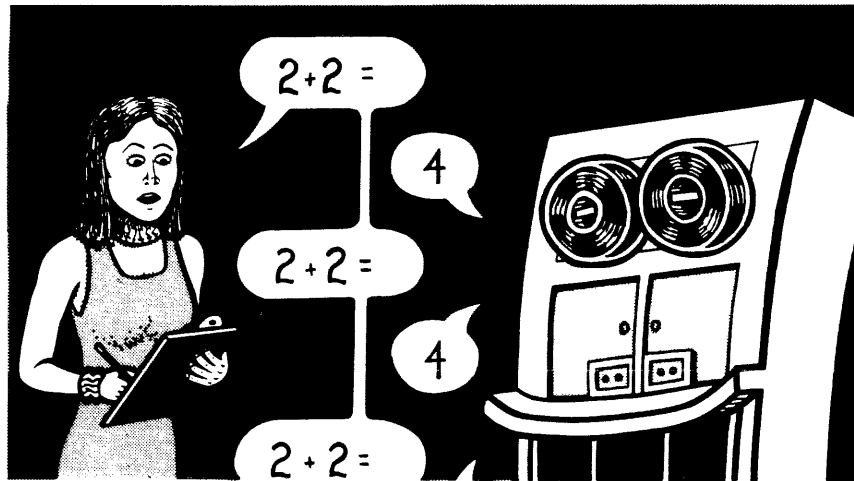
Nothing breaks when you make a mistake
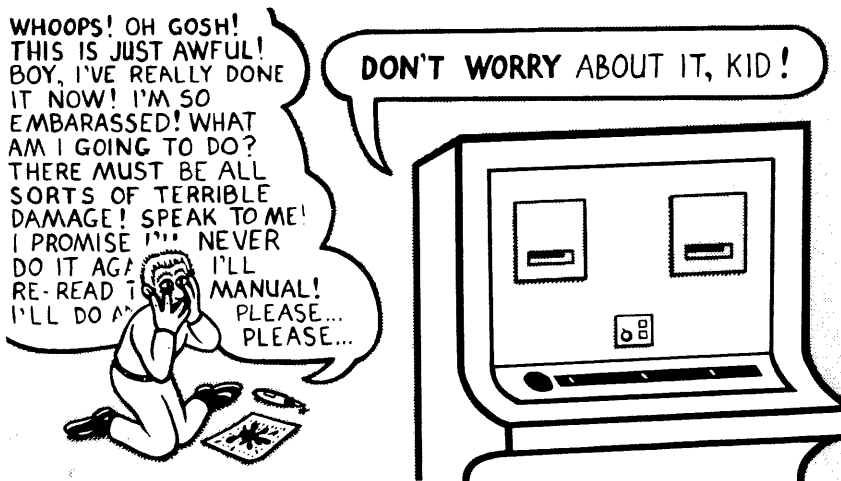
Figure 1-2. Three things to remember about computers

TALKING WITH YOUR COMPUTER—
SOME BASIC TERMS

A given period of time spent at your terminal working with the computer is called a session.

All work done in a session is done on files. A file is a series of text strings (letters, numbers and symbols) arranged in lines.

The conversation between you and the computer is called a dialog, or an interactive dialog (because you and the computer are interacting). Your portion of the dialog is called input. Input consists of commands and data.

A command is a line of input consisting of a command word, which specifies what to do, and parameters, which are what the command is done to (or with). For example, in the command: COOK dinner, COOK is a command word and dinner is a parameter. Commands do things like:

- Request information, such as the names of the files in your User File Directory or a listing of the contents of a file.

- Specify a system or program you want to use, such as EDITOR or RUNOFF.

- Manipulate your files – rename, print out and delete them, etc.

- Do specific tasks – change a line, find a word, indicate you're about to type in some data, etc.

Data is the actual information the system processes, as directed by the commands. Data consists of numbers and text strings.

Some commands require that you make choices. Certain of these commands are done one particular way 90% of the time. To save trouble, the computer will automatically assume that you want these commands done in these standard ways unless you explicitly tell it otherwise. These choices are called defaults. All defaults are defined in this book when each command is explained.

Dialog from the system is called output. The various types of output are:

- Requested information – Information you asked to see.

- Reminders – Prompts, queries and verifications, to help you get everything the way you want it.

- Messages – Error messages from the computer or messages from the System Administrator to certain or all users.

Error messages

The computer gives you an error message:

- Whenever you try to do something that is "illegal" – it cannot be done.

- Whenever you type in a command that makes no sense.
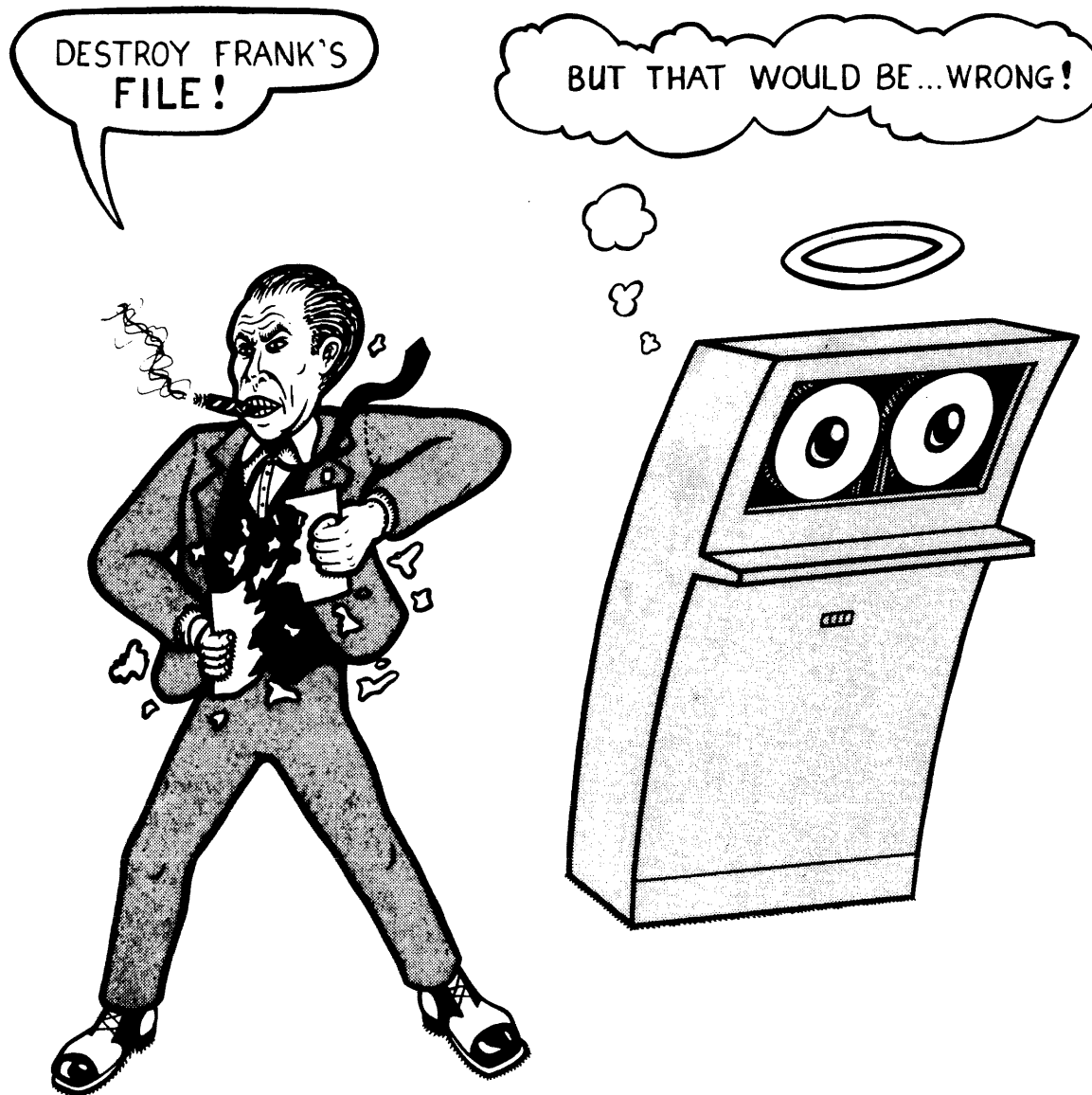
Depending on which of these categories a given error will fall into, you will either get a question mark (?), meaning "that makes no sense," or an error message containing an abbreviated explanation of what PRIMOS thinks you did wrong, such as "ATCH$$" plus the word "ER!" (for "ERROR"). Errors are usually caused by a typing error, or by your losing track of what precisely you wanted to do. Look at what you typed that caused the error, and try again.

CONVENTIONS USED IN THIS MANUAL

Commands may contain command words, parameters, and keywords. Commands are given in a general command format, as follows:

COMMAND parameter

Command words

The command word is given in upper case; however, you may input command words in any combination of upper and lower case characters. You do not always have to type the entire command word; the rust-colored letters are sufficient. (You may type any of the non-required letters, in addition, – e.g., for **PRINT,** you could say **P, PR, PRI, prin,** or **PRint.**)

Parameters

Commands often have parameters which are shown in lower-case letters. Parameters specify something about how the command should be performed. For example;

      PRINT n

means "print **n** lines", where n is a number.
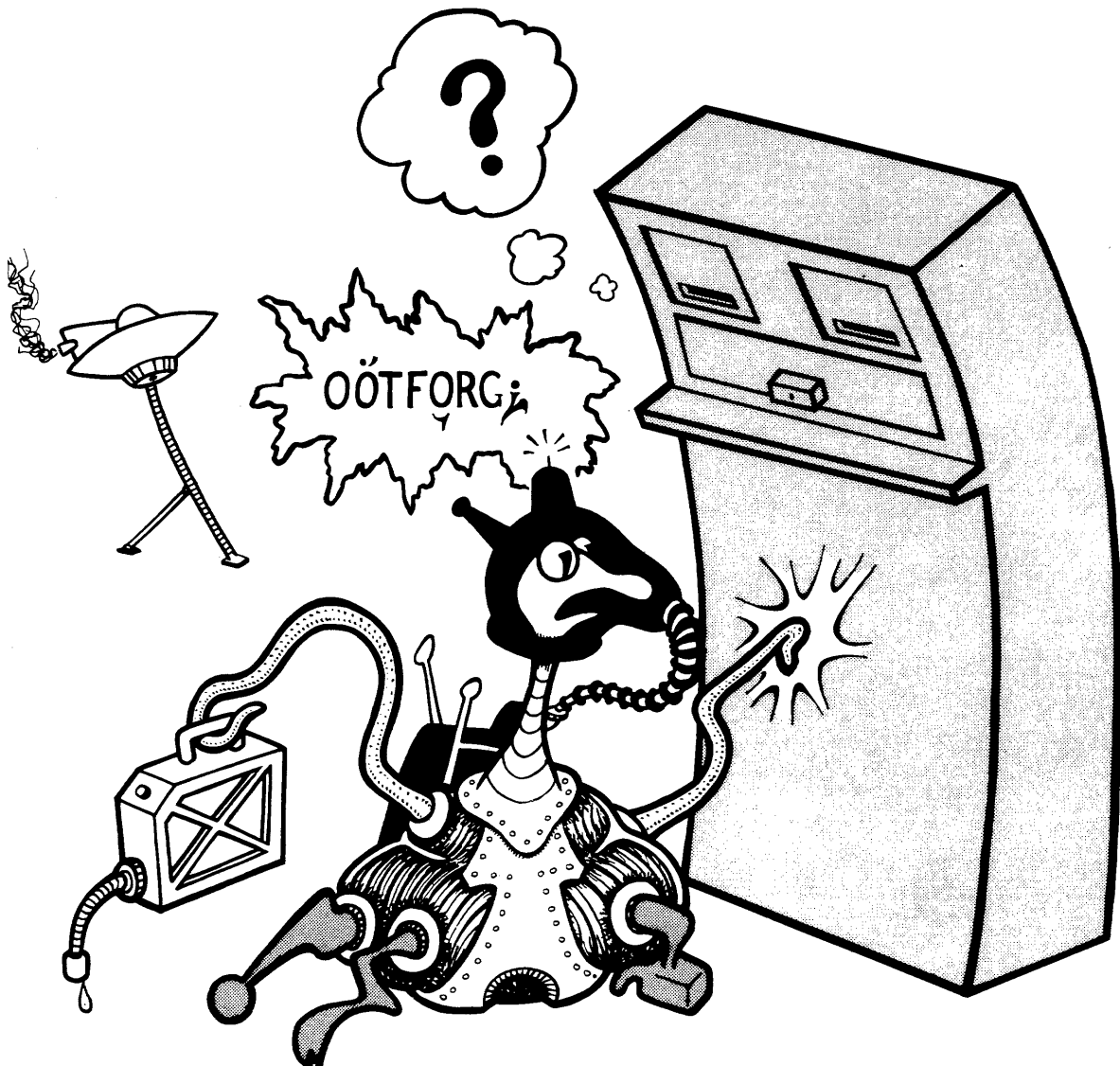
If a parameter is enclosed in brackets, – [parameter] – it is optional. For example, in the format PRINT [n], you could say PR5, PR-10 or PR (Note: these do not mean the same thing). If you do not specify a parameter, the default value is used.

There are two types of parameters: numeric and text.

A numeric parameter is represented in a command format by a lower-case letter, usually "n". Numeric parameters specify things like numbers of lines, times to do an operation, etc. As an aid, all parameters are shown in bold when they first appear in text, like this: "...**n** lines from filename." The value of n can always be positive; often n can be zero or negative (the permissible range of values is explained with each command).

If n is negative, the number must have a minus sign (–) immediately to its left; e.g., –12, –5 with no intervening spaces.

In most cases, parameters have default values, which are used if you omit the parameter from the input.

In the command format, when n is enclosed in parentheses immediately following a command word, e.g., FIND(n) – both parentheses are required, with no space between the command word (or abbreviation) and the left parenthesis. For example:

FIND(6)
F(2)
FIND(5)

A text parameter is one or more characters, sometimes including blanks, in a row. A text parameter can either be:

- A filename which is the name of a file – e.g., MEMO, DAVIDS-NOTES.

- A single character; the value of a character may be a letter, number, or symbol.

- A string – a series of characters which has no command meaning to the computer, but has meaning to the user like "PAGE 99" or "PLEASE RETURN AT ONCE". A string that contains no characters is called a null string.

Keywords

Keywords are words shown in upper-case letters (other than the command word). They are important in clarifying the use of the parameters. Keywords must always be used exactly as they appear. In this manual the most common keyword is TO.

Conventions in the examples

When examples have been taken from actual computer sessions, the user's input is shown in rust and the computer's output is in dark brown.

TERMINALS

A terminal is a device that allows you to "talk" with the computer. It must have a keyboard, something on which it can display output, and a connection to the computer.

It is important to remember that you do not have to always use the same terminal to work on the computer. Any terminal that is connected to the computer will do, provided it is the proper kind for your purpose.

Types of terminals

There are two basic types of terminals – ones that type output onto paper, and ones that display output on a TV screen.

Terminals that type are referred to as "hard-copy" terminals. These are useful when you want to save a printed copy of your session, or type out a particular file.

Terminals with screens are called either video terminals or CRT (for Cathode Ray Tube) terminals. Video terminals produce output faster than hard-copy terminals. They are also useful when you are doing a lot of work for which the end result is important but not the in-between stages. With a video terminal, you don't consume lots of paper which you would only dispose of later.

YOUR TERMINAL KEYBOARD

The layout of the terminal keyboard can vary from one type of terminal to another. Figure 1-3 shows a typical keyboard.
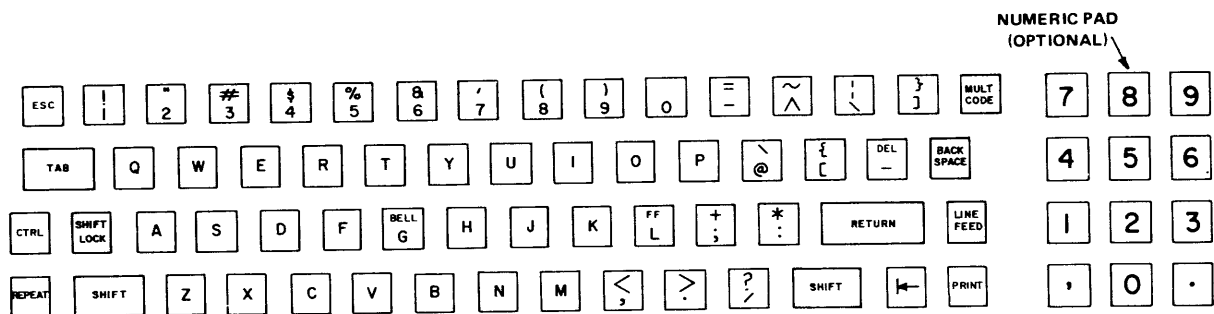
Figure 1-3. Typical terminal keyboard

Besides the usual letters, numbers and punctuation symbols, your terminal also has a variety of special symbols and keys. The number and letter keys are arranged in the same positions as on all standard typewriters. The punctuation marks may be located on different keys, depending on the terminal model. Special keys fall into the following categories:

- Terminal controls & switches
- Special keys
- Special characters

Terminal controls & switches

Terminal controls and switches affect the special ways a terminal performs. Depending on what model terminal you have, these may be switches on the front, side or bottom

of your terminal; they may be keys on the side of the keyboard. The controls and switches that you need to know about are:

ON/OFF – This is the power switch. Printing terminals often have an indicator light which glows when the power is on. On some terminal models, this switch is located on the bottom or at the rear.

LINE/LOCAL – This switch controls whether or not the terminal is sending input to the computer. In LINE mode, the terminal and the computer are connected; in LOCAL mode, the terminal acts just like a specialized typewriter. This switch is often labeled;

> **ON-LINE/OFF-LINE**
> **REMOTE/LOCAL**
> **LINE** (and an indicator light which is on in LINE mode, off in LOCAL mode)

If in doubt about which mode your terminal is in, press the carriage-return key. If the terminal advances to a new line, it is in LINE mode; if it simply returns to the beginning of the same line, it is in LOCAL mode.

UPPER-CASE/LOWER-CASE – Unlike the SHIFT key, the UPPER-CASE/LOWER-CASE key only affects the meanings of the letter keys. UPPER-CASE causes all letters to print in upper-case, no matter what the setting of the shift key is; LOWER-CASE lets you select between upper and lower-case in the standard manner (by using the shift key). On some terminals, this switch is located on the bottom instead of on the keyboard. This key is often labeled:

> **CASE**
> **UPPERLOWER**
> **UC** (for upper-case – when on)

Certain terminals do not have this switch at all; they produce only upper-case letters.

Special keys

The CONTROL key: The key labeled CONTROL (or CNTRL) indicates control characters when pressed at the same time as a character key. Control characters have meanings to the computer which are quite different from the character's normal meaning.

The RUBOUT key: The key labeled RUBOUT has a special use in RUNOFF. This key has no effect anywhere else in your Prime computer.

Special Characters

Up-Arrow: The character indicated as an Up-Arrow (↑) or a caret (⌃) is called Up-Arrow. The Up-Arrow has certain specific uses in the EDITOR and RUNOFF.

The Backslash: The Backslash symbol (\ ) has a specified meaning in EDITOR – the default TAB character.

If you have difficulty getting your terminal to work, ask for help.

WHAT IS PRIMOS?

PRIMOS is the operating system of your Prime computer. A computer's operating system (also called the Monitor, Monitor System, or Supervisor) acts as Office Manager, Security Guard, Time Administrator, Bookkeeper, File Clerk, Messenger, Runner, Lifeguard, and Information Operator. PRIMOS takes care of all these necessary chores so that you can use the computer to do what you want without worrying about how it's being done.

PRIMOS' chores include:

- Keeping track of who's allowed to use the system.
- Keeping track of who's doing what on the computer at any given time, from what terminal they're doing it, and how much computer time and space they're using.
- Giving you the appropriate "piece of equipment" – e.g., EDITOR or RUNOFF.
- Doing all the house work to clean up behind you.

YOUR USER FILE DIRECTORY NAME

In order to do any work on the computer, you must first identify yourself to PRIMOS.

PRIMOS recognizes users by the name they type when logging in. These accounts are called User File Directories (UFDs); their names are UFD-names. In order to do any work on a Prime computer, you must know a UFD-name. For each account name, there is a corresponding file area. You may, however, also use files which are not in your own UFD, if you know the name of the UFD they are in.

Passwords

In order to keep just anyone from getting access to each person's UFD, PRIMOS can be told to require a password from a prospective user. Knowing the password identifies you as someone authorized to use the associated UFD-name.

How to get your UFD

Go to your System Administrator and ask for the name of a UFD, and a password (if needed). Your UFD-name is often the name of your department or project, or your own name. Depending on circumstances, you may share a UFD with other people – or have several UFDs of your own.

HOW TO CORRECT TYPING ERRORS AT THE TERMINAL

Every time you type the Erase character, you will erase the most recent character. Unless your System Administrator has changed it, PRIMOS' Erase character is the double-quote (").

Each " you type erases the most recently typed character other than another double-quote mark. So typing two double quotes (" ") would erase the last two characters, three double-quotes (" " ") would erase three, and so on. In other words, if you had typed lygin" " " "ogim"n, the computer would read simply login.

The Kill character will erase the entire line. Unless your System Administrator has changed it, PRIMOS' Kill character is the question mark (?). For example, logim my filp?login myfile would read login myfile

HITTING THE CARRIAGE-RETURN KEY
ENTERS A LINE FROM THE TERMINAL

The computer pays no attention to what you type on the terminal until you hit the carriage-return key. When you press the ⌨RETURN key, the system reads the line of input, if any, from your terminal and:

- Inspects the line for typing corrections (Erase & Kill).
- Analyzes what you have typed. If your input can be interpreted as a command, PRIMOS does it; if not, it gives you an error message, which is an analysis of what it interpreted your command to be, and why it wasn't acceptable.

Remember: You enter a line of typing from the terminal to the computer by hitting the carriage-return key, ⌨RETURN .

GETTING READY TO LOG IN

In order to log in, you must know the name of a User File Directory (UFD). You may also need to know a password to go with this UFD; this will identify you as an authorized person. Not all UFD's will require passwords.

The general sequence of steps involved in logging in is:

1. Make sure your terminal is connected to the computer, turned on and in LINE mode.
2. Give the LOGIN command

CONNECTING THE TERMINAL WITH THE COMPUTER

If your terminal is not connected to the computer, you'll never be able to log in. This is clearly the first thing you must get done.

Turning your terminal on

Plug the terminal in and turn the power on. The power switch on your terminal may be located either on the keyboard or on the bottom of the terminal. Keyboard switches or keys are usually labeled POWER or ON/OFF. When you turn your terminal on, if it has an indicator light it will go on.

Line mode

As we explained in the Introduction (Section 1), there is a key or switch somewhere on your terminal called the Line/Local Switch. This allows you to use your termimal either as an independent machine or with the computer. In order to connect the terminal to the computer, you must be in Line mode. This switch or key is somewhere on the keyboard, or on the front or back panel. It may also be labeled:

> **LINE**
> **REMOTE/LINE**
> **ON-LINE/OFF-LINE**

The terms LINE, REMOTE, and ON-LINE all mean the same thing.

Make sure this switch is properly set. (Some terminals also have an indicator light which goes on in Line mode.)

Connecting your terminal to a telephone line

Some terminals are connected directly to the computer by wires which run between them. Other terminals connect to the computer over the telephone. If the latter is the case, in order to connect your terminal you will need the following:

1. *An acoustic coupler.* This converts the terminal and computer signals into sounds which can pass through the telephone lines. Some terminals have acoustic couplers built into them, on the back or side; these built-in couplers have a pair of holes into which you can insert a phone handset. Other terminals have a small acoustic coupler nearby.

2. *A telephone.* Any regular telephone will do; however, be sure that nobody else can pick up an extension of the line, and that there are other phone lines available for you and others to use. If you use the phone lines to connect to the computer frequently, you may want to have a special phone line put in just for this purpose.

3. *A phone number for the computer.* Obviously, you can't call the computer unless it, too, has a telephone connection. There may be several numbers, depending on how many phone connections the computer has.

If you have these three things, you are ready to dial in.

Dialing in

Dial the phone number for the computer system. Listen in the earpiece after you finish dialing. If you get a busy signal, this means all the computer's phone lines are busy. Hang up and try again later.

If the lines are not busy, the ring will stop after one or two rings, and you will hear a high-pitched tone in the earpiece of the phone handset. This means the computer is "on the line." Place the telephone handset correctly onto the openings in the acoustic coupler.

If your terminal has a LINE indicator light, it should light up. Press the ⌐RETURN⌐ once or twice.

Now that your terminal is connected to the computer, you're ready to actually log in.

LOGGING IN

If you have been given a UFD-name, and have connected the terminal to the computer, you are ready to give the LOGIN command. The format of the LOGIN command is:

> **LOGIN ufd-name [password]**

If you have done this correctly, PRIMOS will acknowledge you as being logged in.

```
login sales
SALES   (15) LOGGED IN AT 10'36 110579
```

The number in parentheses is your terminal number, assigned by PRIMOS. The time you logged in (here 10'36) is expressed in the 24-hour system, also known as Military Time. Here are a few examples of 24-hour times and their 12-hour equivalents.

0'00	Midnight
0'01	12:01 A.M.
1'00	1:00 A.M.
3'30	3:30 A.M.
11'59	11:59 A.M.
12'00	Noon
12'01	12:01 P.M.
13'00	1:00 P.M.
15'30	3:30 P.M.
23'59	11:59 P.M.

The current date is expressed as month day year. For example, the number "110579" means November 5, 1979.

PROBLEMS IN LOGGING IN

It is possible that you will not be able to log in when you try. This is most likely due to a typing error on your part, and is easily remedied. As a rule, hitting the carriage-return key and typing the LOGIN command again will do the trick.

Here's a list of the possible error messages you may get when trying to log in, along with their meanings and what you should do.

MESSAGE	MEANING(S)
LOGIN PLEASE	• You misspelled the word LOGIN. Try again.
	• You tried to start work without logging in. Try again.
	• There was a temporary transmission problem in the line. Hit two or three keys (any keys) and a `RETURN`, then try again.
NOT FOUND	• You misspelled the name of your UFD. Try again.
	• No UFD by that name exists. Check your UFD name.
ILL REM REF.	• Same reasons as NOT FOUND (above).
NO RIGHT (LOGIN)	• You forgot to type your password. Try again.
RESERVED CHARACTER	• You accidentally typed a character which may not be used in a command. Try again.
BAD PASSWORD	• You misspelled your password – or the password has been changed. See your System Administrator.
NOT A UFD	• You tried to log into something which can't be logged into. Try another UFD-name. If it was your ufd-name, and you are sure you spelled it right – try a few more times to be sure – call your System Administrator.

OTHER REASONS YOU CAN'T LOG IN

There are also a number of physical circumstances that can prevent you from logging in, *none* of which yield error messages. These are:

1. *Your terminal is not turned on.* (Remember: Some terminals have the power switch located on the bottom; carefully lift terminal up and look.) Turn terminal on. Make sure the power cord is properly connected and that the terminal is plugged in.

2. *The terminal is in LOCAL mode.* This means it is acting purely as a typewriter. If you have a LINE/LOCAL switch, push it. Then hit a key or two – any key – plus the CR key once or twice; if you've succeeded, the system will respond with the message LOGIN PLEASE.

3, *The settings inside your terminal are inappropriate.* See your System Administrator.

4. *Your terminal is physically disconnected from the computer.*

5. *If your terminal connects by way of a telephone, check the following:*
   - Is your acoustic coupler on?
   - Did you dial the correct number for the computer? (If you did you'll hear a high-pitched whistle in the earpiece.)
   - Is the phone handset properly and snugly inserted? Make sure the earpiece is over the part of the coupler that makes noise.

6. *All the phone lines to the computer are busy.* Try again.

7. *Your terminal is not working.* Call the repair person.

8. *Last, but not least, the whole computer may be "down".* In other words, OFF. If this is true, there's nothing you can do but wait. The system may not be working for a short time. If your terminal is directly connected, hit the 〔RETURN〕 key every so often to see if the system has come "up" (been turned on) again.

When you have successfully logged in, you'll get a message of the form:

ufd-name (terminal-number) LOGGED IN AT time date

If for *any* reason, you cannot succeed in logging in after a reasonable number of attempts, consult your System Administrator.

FINDING OUT WHAT'S IN YOUR UFD

Now that you're logged in, you've got a "work space" plus a User File Directory containing files to work on. (Unless you have been given a brand-new UFD, of course. In this case, there are no files in your UFD – it is "empty".)

In order to specify which file to work on, you have to know what files are in your UFD. While you could keep track of the contents of your UFD by writing down the names of files on a sheet of paper and keeping it with you all the time, PRIMOS can do this record-keeping for you faster and more reliably. PRIMOS maintains an up-to-date list of the names of all the files within your UFD. To see this list, you give the LISTF (for List Files) command. The format of the LISTF command is:

LISTF

You need only type the letter L, if you wish.

Let's suppose you have logged into a UFD named STAFF, which contains the files FRANK, MARTHA, ALBERT, and ROSEMARY. Giving the LISTF command would have this effect:

```
listf

UFD=SALES   3  0

FRANK   MARTHA  ROSEMARY      ALBERT

OK,
```

If there were no files in the STAFF UFD, doing a LISTF would have this result:

    OK, listf

    UFD=SALES   3 O

    .NULL.

    OK,

The number after the name of the UFD helps the computer tell where the UFD is. This number is not important to you, (it identifies the "logical disk"). The letter next to the number will either be O (for Owner) or N (for Non-owner). Owner means you can inspect and change files; Non-owner means a password was required and you did not give it. As a non-owner you are permitted to inspect files, but not change them. The error message NO RIGHT indicates you have not given the necessary password.

It's a good idea to give the LISTF command at the beginning and end of each session, to check up on what files you've got in your directory.

LOGGING OUT

When you have finished a session at the terminal, you must inform PRIMOS you are done. This is called logging out.

Logging out is the opposite of logging in – it tells PRIMOS that you have finished whatever you were doing (or are going to stop for the time being). PRIMOS puts all your files away, and signs you out. You log out by giving the LOGOUT command. The format of the LOGOUT command is:

LOGOUT

PRIMOS acknowledges the command with a message of the form:

ufd-name (terminal #) LOGGED OUT AT time date
TIME USED=terminal-time   CPU-time   I/O-time

- Terminal time is the amount of elapsed clock time between logging in and logging out. It tells you how long your session lasted. Most of your 'real' time the computer probably was "waiting" – i.e., doing other work until you gave it a new command.

- CPU time means how much actual time the computer spent following your commands. (CPU stands for Central Processing Unit).

- I/O time is the amount of time the computer spends moving data around.

Giving the LOGOUT command looks like this:

```
OK, logout
SALES    (15) LOGGED OUT AT 11'28 110579
TIME USED= 0'12   0'06   0'032
```

NOTES ON LOGGING OUT

It's often a good idea to log out if you're going to leave your terminal for awhile, to prevent anyone else from doing work on your account time, or from making changes in your files. There's nothing wrong with leaving yourself logged in all day, but since logging in and out is so easy, it's a good idea to be logged in only when you want to work.

EDITING

Editing a file consists of three basic operations:

- Searching
- Examining
- Changing

Changing a file means adding, altering, deleting. Editing a file consists of finding and examining what you want to change, and examining your changes.

WHAT EDITOR IS

EDITOR is a system designed to let you create and edit text files on the computer. EDITOR has a few dozen commands. Some of these do work on a file; others make it easier to do this work.

You make a file by typing it on your terminal, line by line, and then editing this text.

If you have a file typed like this:

```
'Twas broiling time, and the lithe and slimey toves
Did gyre and gimble
It was many and many a year ago
And the mome raths outgrabe.
All mimsy were the borogroves
```

EDITOR lets you correct and change it to:

```
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
```

CONVENTIONS IN EDITOR

The erase and kill characters

EDITOR's erase and kill characters are the same as PRIMOS'. Unless you or your System Administrator have changed them, EDITOR's Erase character is the double-quote ("), and Kill character is the question-mark (?).

Command format

EDITOR's command format is:

COMMAND parameter

Command words: The word in capital letters is the command word. The letters shown in rust in the command word indicate the minimum required abbreviation. You must type at least these letters; you may type as many more as you wish. The following are all acceptable ways to enter a command whose format is **APPEND: A, APP, APPE, APPEND.**

Also you may type the letters of command words in any combination of upper and lower case letters. All of the following are equally valid: **A, a, ApPend, append, appEND.**

Parameters: As a rule, the parameter in a command format will be:

- The word **filename,** representing a filename.
- The letter **n,** representing a number.
- The word **character,** representing a single character.
- Any of the words **string, text,** or **newline,** representing a piece of text.

Parameters which are enclosed in brackets – [**filename**] – are optional. EDITOR will use the indicated default value, if you do not specify a value.

If the parameter is a number, represented by the letter n, you do not need to type a space between the command word and the number. For example, the following are all valid:

Print5
Pr5
PRINT 15
p-5
p   -5

Note that there cannot be a space between the minus sign and a number.

If the parameter is a filename or a character, you must have at least one space between the command word and the parameter, as in:

load memo
kill &

If the parameter is a text string (indicated by text, string, newline), EDITOR assumes that there is exactly one space between the command word (or abbreviation) and the text string, and that any subsequent spaces are to be considered part of the text string.

> **find DEPARTMENT**
> **append and henceforth**
> **insert Dear Sir or Madame,**

Any space after the first space is considered part of string. For example, the command:

> **find ⑤⑤⑤ henceforth**

would find the string ⑤⑤ **henceforth** which began with two blank spaces.

Example format

Throughout this book your input is in rust, and the computer's output is in dark brown.

HOW EDITOR WORKS

EDITOR has a special file area called the work file which is reserved for its own use. EDITOR puts all input into this work file and does all its editing on the contents of this work file.

INPUT AND EDIT MODES

EDITOR has two modes: input mode and edit mode. In input mode, EDITOR treats whatever you type as text which is put directly into your work file, line by line. In edit mode, EDITOR treats your input as commands, and executes them on the contents of the work file.

You can switch from one mode to the other without trouble. How you enter EDITOR determines whether you begin in input or edit mode, but during given session, you are likely to switch between input mode and edit mode a number of times.

ENTERING EDITOR

You enter EDITOR by typing the ED command:

> **ED [filename]**

Entering a new file

If you do not specify a filename when you give the ED command, EDITOR assumes that you want to create a brand-new file, and goes into input mode.

```
OK, ed [ RETURN ]
GO
INPUT
```

Everything you type will then be treated as input, and inserted directly into your work space until you give EDITOR the signal to switch over to EDIT mode.

Editing an existing file

If you do specify a filename when you give the ED command, EDITOR assumes you want to be in EDIT mode so you can edit this file, e.g.,

```
OK, ed memol  ⎡ RETURN ⎤
GO
EDIT
```

Here's what happens when you give the ED command and specify a filename.

1. EDITOR finds this file in your UFD.
2. EDITOR then makes a copy of this file and puts this copy in its own work file. This is called the work file, or EDITOR file. The original file is not affected by this in any way whatsoever. This process permits you to make changes, alterations, and additions to a copy without having to lose the original file. This way, you don't have to worry if you make a mistake, or change your mind – you can always go back to the original. You can also make several slightly different versions of the same original file with different names.
3. Once EDITOR has copied the file into its work space, it goes into EDIT mode, on the assumption that you want to edit the copied file. If you want to input new text somewhere into the work file, you can switch to input mode. Note that entering new text into the file does not change the original file.

Before we explain how to switch from edit to input mode, let's take a quick look at how to use input mode to enter text.

ENTERING TEXT IN INPUT MODE

When you are in input mode, EDITOR inserts whatever you type into the specified filename (either new or old), line by line. In input mode, EDITOR will interpret a typed semicolon (;) as a carriage-return. This allows you to end a line of input by typing either a semicolon or a carriage-return, for example, the input:

```
line one;line two;line three
```

would become

```
line one
line two
line three
```

Following a semicolon by *another* semicolon (in input mode)

```
line one;;line three
```

inserts a blank line in the file,

```
line one

line three
```

This use of the semicolon is particularly useful for a long list of short lines. For instance:

```
apples;bananas;dates;figs;garlic;wolfbane
```

would become

```
apples
bananas
dates
figs
garlic
wolfbane
```

This obviously means you cannot type a semicolon as part of the file. Later we'll show you how to get around this.

SWITCHING FROM INPUT TO EDIT MODE

EDITOR switches from input mode to edit mode whenever you type either:

1. A semicolon *followed* by a `RETURN` , or
2. Two `RETURN` in a row.

For example,

> **INPUT**
> **text;** `RETURN`   *or*
> **EDIT**

> **INPUT**
> **text** `RETURN`
> `RETURN`
> **EDIT**

Each of these two sequences encloses a null input line. A null input line is a line of input that contains no text or semicolon.

You enter a blank line (a line containing nothing but a `RETURN`) whenever you type:

text;;text

HOW EDITOR WORKS ON ITS FILE

EDITOR is a line-oriented system. EDITOR sees your file as a collection of lines.

The pointer and the current line

EDITOR works on one line of your file at a time. This line is called the current line. EDITOR remembers which line is the current one by positioning a pointer alongside. (Think of the pointer as a paper clip which marks your place but can be moved without leaving marks.)

Null lines

Your file also contains placemarkers called null lines. A null line does not really exist; it marks the place where a line can be inserted.

EDITOR indicates the presence of a null line by outputting:

.NULL.

This is *not* the contents of the line! It is only a reminder.

There is *always* a null line above the first and below the last line of your file to permit insertion of lines above the top line or below the bottom line. EDITOR has TOP and BOTTOM markers set beyond these null lines, to remind you when you are at the end of your file.

Whenever you delete a line, EDITOR places a null line opposite the pointer in case you want to immediately insert a new line there. The .NULL. will disappear as soon as you move the pointer to a new current line. (Note: a null line is different from a blank line. A blank line has no contents, but has a `RETURN` at its end.)

Line numbers

EDITOR assigns a list of line numbers to the lines in the work file, for its own reference. This list is constantly updated so that every number belongs to a non-null line.

EDITOR can tell you the line number of the current line, which the pointer indicates. You can move the pointer up and down, either by a given number of lines or by specifying a line number. You can delete or retype the current line, move the pointer to a line which contains a specific word, change, etc. These tasks are all done by giving commands in edit mode.

GIVING COMMANDS IN EDIT MODE

When you are in EDIT mode, you type commands *instead* of text. EDITOR's commands fall into several categories, such as:

- Commands that print lines (PRINT) or a line number (WHERE).
- Commands that reposition the pointer to a specified line (TOP, BOTTOM, NEXT, POINT) or to a line containing a specified string (FIND, NFIND, LOCATE).
- Commands that change a line (APPEND, CHANGE, DELETE, INSERT, RETYPE).
- Commands that return you to PRIMOS (FILE, QUIT).

These commands are sufficient for you to do basic work with the EDITOR. We'll explain each of these essential commands shortly.

SWITCHING FROM EDIT TO INPUT MODE

It is possible to insert a new line into your work file while in edit mode. If you have several lines to insert in the same place, it is easier to switch into input mode.

EDITOR will switch from edit mode to input mode whenever you type a null command line.

A null command is a command containing no command word or parameter. You enter one by following a non-null command with a

$$\left\{ \begin{array}{c} \text{semicolon (;)} \\ or \\ \boxed{\text{RETURN}} \\ or \\ \text{comma (,)} \end{array} \right\} \qquad followed\ by\ a \qquad \left\{ \begin{array}{c} \text{semicolon (;)} \\ or \\ \boxed{\text{RETURN}} \\ or \\ \text{comma (,)} \end{array} \right\}$$

Commas *cannot* end the APPEND, INSERT, OVERLAY and RETYPE commands as they would be treated as text.

In other words, typing any pair of semicolons, carriage-returns, and/or commas after a command, will switch you from edit mode to input mode, except when the comma follows APPEND, INSERT, OVERLAY or RETYPE.

When you switch from edit to input mode, new input will be inserted below the most recent current line in edit mode. When you switch from INPUT to EDIT, the last line of input becomes the current line.

EDITOR displays the mode whenever you switch between input mode and edit mode. It is important to remember which mode you are in. When you are in input mode, if you give a command it will be put into your file as text. In edit mode, EDITOR will try to interpret as commands what you meant as text. The results in both cases will be undesirable.

BASIC EDITOR COMMANDS

You should be able to do most of your text-editing using this selection of the EDITOR commands.

APPEND	FIND(n)	POINT
BOTTOM	INSERT	PRINT
CHANGE	LOCATE	QUIT
DELETE	NEXT	RETYPE
FILE	NFIND	TOP
FIND	NFIND(n)	WHERE

These commands are explained in the next few pages. A categorized list of the remaining commands appears at the end of this section. Once you have become proficient at using this first group, you should have little trouble learning to use any of the other commands explained in Section 8 Sample Sessions, and the EDITOR Reference Section, Section 9.

Sample file

The following file, which is a selection from Lewis Carroll's "Jabberwocky", is used in all the examples in this section.

'Twas brillig, and the slithy toves
Did gyre and gimbel in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jub-Jub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome foe he sought—
So rested he by the Tum-Tum-tree,
And stood awhile in thought.

EDITOR'S ERROR MESSAGES

In edit mode, if you give EDITOR a command which it cannot understand, you will get one of the following two error messages:

- **BAD abbreviation**–This means that you did not use the proper format for the command.

- **?**–Your input could not be interpreted as any of the EDITOR commands. This is often a result of thinking that you are in input mode when you are still in edit mode.

For example:

```
appen
BAD APPEN      Improper command format
nm
?              Meaningless command
c/brillig
BAD C          Bad command format
?
pfint6
?              Meaningless command
```

THE PRINT COMMAND

The PRINT command prints **n** lines of your file, including the current line, and makes the last line PRINTed the new current line. The format of the PRINT command is:

   PRINT [n]

If n is –1, 0, or omitted, the default value of 1 is used. If n is negative, EDITOR moves the pointer back n lines from the current line, and then prints one line, which is the new current line.

```
print 3                                     PRINT three lines
.NULL.                                      This is a null line.
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
print 6                                     PRINT six lines
Did gyre and gimble in the wabe;            from new current line
All mimsy were the borogoves,
And the mome raths outgrabe.


"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
print                                       PRINT new current line
The jaws that bite, the claws that catch!
print -5                                    Back up 5 lines
All mimsy were the borogoves,               including current line
print -1                                    PRINT new current line
All mimsy were the borogoves,
print -2                                    Back up two lines,
Did gyre and gimble in the wabe;            PRINT current line
print 0                                     PRINT current line
Did gyre and gimble in the wabe;
print                                       PRINT current line
Did gyre and gimble in the wabe;
```

The space between PRINT and n is optional. A PRINT immediately after TOP or BOTTOM yields .NULL. .

THE WHERE COMMAND

The WHERE command prints out the line number for the current line. The format of the WHERE command is:

WHERE

The WHERE command is most useful with the POINT command.

```
where
LINE        1
next3
All mimsy were the borogoves,
where
LINE        3
next5
Beware the Jubjub bird, and shun
where
LINE        8
next-12
TOP
where
LINE        1
point 7
The jaws that bite, the claws that catch!
where
LINE        7
```

POINTER-MOVING COMMANDS

Pointer-moving commands either reposition the pointer to a specific line or to a line containing a specific string. EDITOR's specific pointer-moving commands are TOP, BOTTOM, NEXT, and POINT.

The TOP command

The TOP command positions the pointer at the null line at the top of the file, just above the first line of text. The format of the TOP command is:

TOP

For example:

```
top
TOP
print
.NULL.
print2
.NULL.
'Twas brillig, and the slithy toves
```

The BOTTOM command

The BOTTOM command positions the pointer at the bottom of the file, just below the last line of text. The format of the BOTTOM command is:

BOTTOM

For example:

```
bottom
print
.NULL.
next
BOTTOM
```

The NEXT command

The NEXT command moves the pointer **n** lines and prints the new current line. Positive values of n move the pointer down towards the bottom of the file; negative values up towards the top. The format of the NEXT command is:

NEXT [n]

If n is zero or unspecified, the default value of 1 is used. If n is great enough to move the pointer beyond the top or bottom null line, the pointer stops at the null line, and either TOP or BOTTOM is printed.

```
top, next
'Twas brillig, and the slithy toves
next 2
All mimsy were the borogoves,
next 4
The jaws that bite, the claws that catch!
next -5
Did gyre and gimble in the wabe;
next 0
All mimsy were the borogoves,
next
And the mome raths outgrabe.
```

The POINT command

The POINT command positions the pointer at line **n**. The line numbers are not actually part of your file; EDITOR generates them for its own reference. The format of the POINT command is:

POINT n

The POINT command is equivalent to the sequence TOP, NEXT n. The value of n must be greater than 0. Point 0 will give you an error message. POINT 1 is equivalent to TOP,

NEXT. If n is greater than the number of lines in the file, the pointer will be left at the bottom.

```
point 3
All mimsy were the borogoves,
point 7
The jaws that bite, the claws that catch!
point2
Did gyre and gimble in the wabe;
where
LINE          2
point 6
"Beware the Jabberwock, my son!
where
LINE          6
```

STRING-FINDING COMMANDS

The FIND, NFIND and LOCATE commands reposition the pointer to the first line below the current line which contains the specified string.

These commands distinguish between upper and lower case letters in a specified string. If you are unable to find old lines in your file, but can find newly inserted ones, and your current display is set for all CAPS, the CASE control on your terminal may be in the wrong position.

The LOCATE command

The LOCATE command moves the pointer to the first line below the current line which contains **string** anywhere in that line. The format of the LOCATE command is:

LOCATE string

If no line containing string is found below the current line, BOTTOM will be printed and the pointer left at the end of the file. The string cannot contain commas.

Example:

```
locate mimsy
All mimsy were the borogoves,
```

The FIND command

The FIND command is a specialized version of LOCATE which only checks to see if **string** is at the beginning of a line (i.e., the first character is in column 1, the second in column 2 . . .). FIND repositions the pointer to the first line below the current line which begins with string. The format of the FIND command is:

FIND string

If no line beginning with string can be found, the pointer stops at the end of the file, and the word BOTTOM is printed. The string cannot contain commas.

```
find All
All mimsy were the borogoves,
```

The NFIND command

The NFIND command moves the pointer to the first line below the current line which does not begin with **string.** The format of the NFIND command is:

NFIND string

For example:

```
print 4
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
top,next
'Twas brillig, and the slithy toves
nfind Did
All mimsy were the borogoves,
```

If NFIND can't find a line which doesn't begin with string, the pointer will be left at BOTTOM. The string cannot contain commas.

Searching on a specific column

You can also FIND a **string** starting on other than column 1 of the line, by specifying the number of the column within parentheses directly after the command word.

FIND(n) string

The parentheses ( ) around the column number are required. There cannot be any spaces between FIND and (n).

```
find(5) mimsy
All mimsy were the borogoves,
```

Like FIND, you can NFIND beginning on a column other than column 1 using the format:

NFIND(n) string

For example:

```
top,print 5
.NULL.
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe;
All mimsy were the borogoves,
And the mome raths outgrabe.
next -3
'Twas brillig, and the slithy toves
nfind(5) gyre
All mimsy were the borogoves,
```

LINE-CHANGING COMMANDS

The APPEND, CHANGE, DELETE, INSERT, and RETYPE commands alter the text on one or several lines.

The APPEND command

The APPEND command attaches the specified **string** to the end of the current line. The format of the APPEND command is:

APPEND string

Remember: One blank separates the command word APPEND (or abbreviation) from the string you wish to append. All further blanks will be treated as part of the string.

```
print
'Twas brillig and
append  the slithy toves
'Twas brillig and the slithy toves
```

If you want to have one space between the last word of the current line and the first word you are appending, you must type two spaces between the command word and the first word of appended string.

```
print
'Twas brillig and
append , dig it, those slithy toves
'Twas brillig and, dig it, those slithy toves
```

If there is no blank between APPEND and string, EDITOR gives an error message, and you should try again. Text is terminated by either a ⌐RETURN⌐ or a semicolon (;) You can use commas in the APPEND string, but not semicolons.

```
print
'Twas brillig and
append the slithy toves
'Twas brillig andthe slithy toves
```

The CHANGE command

The CHANGE command replaces one string in the current line with another string. The first character after the command word CHANGE (or abbreviation) is used as the delimiter. The format of the CHANGE command is:

CHANGE/string-1/string-2/[G] [n]

For example:

```
print
And the gnome raths outgrabe.
change/gnome/mome
And the mome raths outgrabe.
```

Any character, including the semicolon (;) and the space may be used as a delimiter instead of the slash. This means that the following sample commands would all have the same effect.

```
change&gnome&mome&
change ;gnome;mome;
change KgnomeKmome
ch   gnome mome
change >gn>m>
```

If the letter **G** (for General) is specified, CHANGE will change *every* occurrence of string-1 on a line. If you don't specify G, only the *first* incidence of string-1 will be changed.

If the value of **n** is either 0 or 1, EDITOR will only make changes on the current line. (If n is either 0 or unspecified, the default of 1 is used.) If a value other than 0 or 1 is specified, EDITOR will inspect and make changes on n lines starting at the current line, and leave the pointer positioned at the nth line. If there are less than n lines in the file the message BOTTOM will be printed. EDITOR will print out all changed lines, plus the last line examined.

1. Remember to issue the TOP command before making changes on the file as a whole.

2. If you end the command with a `RETURN` , you can omit the closing delimeter.

3. You can specify the semicolon (;) as a text character within the delimeters —i.e., if you used "@" every place in your file where you wanted to use ";", then the command sequence TOP, CHANGE/@/;/G9999 would change all the @'s to ;'s. (Make sure n is greater than the number of lines in your file.)

4. You can use CHANGE to insert characters at the beginning of a line with the sequence:

change//string/

For example:

**'Twas brillig, and the slithy toves
change//THE BOOK SAID:/
THE BOOK SAID: 'Twas brillig, and the slithy toves**

The DELETE command

The DELETE command deletes **n** lines, including the current line, and leaves the pointer at a null line where the last deleted line was. The null line will be maintained, in case you wish to insert a new line, until a new command moves the pointer away. The format of the DELETE command is:

DELETE [n]

If n is not specified, the default value of 1 is used. n may be positive or negative, indicating deletion of the current line plus n-1 lines below or above the current line. Since n always includes the current line, the commands d, d1, and d-1 are equivalent.

```
print
'Twas brillig and the slithy toves
delete
print
.NULL.
```

NOTE

The DUNLOAD command, explained in the EDITOR REFERENCE section, permits you to move n lines to a new file, thus removing them from the work file without deleting them altogether. Although this technique accumulates files in your UFD, it can be useful if you are afraid of accidentally DELETEing large portions of your file.

The INSERT command

The INSERT command inserts a specified **newline** following the current line; the inserted line then becomes the current line. The format of the INSERT command is:

INSERT newline

For example:

```
print
'Twas brillig, and the slithy toves
insert Were doing their holiday shopping.
next-1
'Twas brillig, and the slithy toves
print 2
'Twas brillig, and the slithy toves
Were doing their holiday shopping.
Insert On Macy's! On Gimbel's! On Woolworth's and Kresge's!
print
On Macy's! On Gimbel's! On Woolworth's and Kresge's!
```

The RETYPE command

The RETYPE command deletes the current line and replaces it with the text specified in **string.** The format of the RETYPE command is:

RETYPE string

Remember: The first space after RETYPE separates the command word from the parameter; all further spaces are part of string.

```
print
Did gyre and gimble in the wabe;
retype Did a wild watusi in the wabe;
print
Did a wild watusi in the wabe;
```

The string is terminated by either a semicolon (;) or a ▭RETURN▭

RETYPE followed *immediately* by a space and a ▭RETURN▭ will act as a DELETE, *erasing* the current line and leaving the pointer at a blank line; RETYPE followed only by a ▭RETURN▭ will yield: **BAD R.**

ENDING AN EDITOR SESSION

Giving either the FILE command or the QUIT command tells EDITOR you are done editing the file. Each has a specific use, which is explained below.

The QUIT command

The QUIT tells EDITOR you do not want to save the EDITOR work file, but instead, want to preserve the original and want to return to PRIMOS-level. The format of the QUIT command is:

QUIT

If you have created/modified a file during the session, EDITOR will respond to a QUIT with:

FILE MODIFIED, OK TO QUIT?

This message asks whether EDITOR may throw away the work file.

A YES (or Y, YE, O, OK or null line [RETURN]) response will QUIT you; you will get back the OK prompt, meaning you're at PRIMOS-level and your session was *not* saved. Any other response will provoke a PLEASE FILE (see the explanation of the FILE command); doing a FILE, with or without a filename (depending on the circumstances), will *automatically* QUIT you, having saved your work. If you *did not* create or modify a file, saying QUIT automatically returns you to PRIMOS.

```
OK, ed poem                              Edit the file POEM
GO
EDIT
print 2                                  Look at 2 lines
.NULL.
'Twas brillig, and the slithy toves
quit                                     Give QUIT command. Since file
                                         is unchanged, QUIT succeeds.

OK, ed poem                              Edit file POEM again.
GO
EDIT
delete 3                                 Enter a change.
quit                                     Giving QUIT command provokes
FILE MODIFIED, OK TO QUIT? yes           reminder to FILE. YES
                                         response QUITs without saving
                                         changes.

OK, ed poem                              Edit file POEM again.
GO
EDIT
delete 3                                 Enter a change.
quit                                     Give QUIT command
FILE MODIFIED, OK TO QUIT? no            NO response provokes request to
PLEASE FILE                              FILE.
?                                        ? is part of PLEASE FILE message
file short.poem                          Giving FILE command saves edited
                                         version of file.
OK,
```

The FILE command

The FILE command turns the EDITOR work file, (which is so far only a temporary) into a permanent file in your UFD and returns you to PRIMOS.

WARNING

Since the work file *does not exist* outside of EDITOR, you *must* FILE if you want to save your work.

The format for the FILE command is:

FILE [filename]

If you have been creating a new file, you must specify **filename.** (The error message FILENAME MUST BE SPECIFIED will occur if you don't.)

```
file
FILE NAME MUST BE SPECIFIED
?
file memo1

OK,
```

You *cannot* have two files with the same name in the same UFD! If you give a filename which already exists in your UFD, EDITOR will *delete* the old file by that name from your UFD, and put the EDITOR work file in its place.

The same warning holds true for old files. If you have been working on an old file, and you specify the old filename, or say FILE without any filename, your old copy will be deleted, and only your new version kept. Giving a new filename will keep both the old and new versions – but be sure not to accidentally wipe out some other old file by using its name.

If you do not wish to save your work from a given session –i.e., want to save your old version, if any – type QUIT instead of FILE. if you have made any insertions or changes in your current file, EDITOR will inquire: FILE MODIFIED, OK TO QUIT? to double-check with you. A YES response QUITs you back to PRIMOS: NO provokes PLEASE FILE, at which point you give the FILE command.

```
quit
FILE MODIFIED, CK TO QUIT? no
PLEASE FILE
?
file new.memo

CK,
```

The rules for making filenames are:

1. Filenames can be up to thirty-two characters long.
2. The first character may be any legal character except a digit.
3. Filenames can contain *only* the following characters: A through Z, 0 through 9, & – $ * . __ /
4. Characters *NOT* permitted in filenames include: imbedded blanks (e.g., MY FILE), and special characters.

5. Upper and lower case letters are treated as upper-case by PRIMOS. (Letters entered in lower-case will be converted to upper-case.)

Valid filenames	*Invalid filenames*
NEWFILE	A?
Todays-Prices	TWO@JOHN
Highs&Lows	"EUREAKA"
$Monthly.Report	WHY A DUCK
R34587	
A-Tale-Of-Two-Cities	

The file command can also be used to make copies of any file, simply by typing ED plus filename and FILEing the copied work file immediately with a new filename, as in:

```
OK, ed memo1
GO
EDIT
file spare-memo1
OK,
```

MISCELLANEOUS INFORMATION CONCERNING EDITOR

Tabs

Do *NOT* try to input tabs with the terminal's TAB key! EDITOR will *not* understand them. To signal a tab, use the Backslash (\ ) character. Typing this line of input

use\them\tabs\wisely, buddy

would be interpreted as:

use   them   tabs   wisely, buddy

EDITOR has pre-set tab stops at columns 6, 12, and 30. To change these and/or add more tab stops, see the TAB command in the EDITOR REFERENCE SECTION.

Entering a double-quote or question-mark in your text

As you know, the double-quote (") and question-mark (?) have special meanings in EDITOR. Every " erases the previous character; ? kills (erases) the entire line.

You can enter a " or ? as actual characters in your text by preceding them with an up-arrow (↑). The table below shows the results of various combinations of ↑, " and ?.

You Type	Result
”	Erase previous character
↑”	Enters a ”
↑” ”	None (Enters ”, then erases it)
?	Kill line
↑?	Enters a ?
↑??	Kill entire line, including ?
↑?”	Enters a ? and then erases it
↑”?	Kills entire line
↑?↑”	Enters ?”

GENERAL INFORMATION ABOUT EDITOR

- EDITOR works on a copy of the original file, called the work file.
- The carriage-return ⌈RETURN⌋ inputs a line of typing.
- The double-quote (”) erases the previous character.
- The question-mark (?) erases the entire line.
- The backslash (\) is the tab character; EDITOR has pre-set tabs in columns 6, 12, and 30.
- The pointer indicates the current line.

EDITOR'S OTHER COMMANDS

Besides the commands you have just learned, EDITOR has additional commands. For more information about each command, see the EDITOR Reference Section.

File loading and unloading commands

LOAD filename
UNLOAD filename [n]
UNLOAD filename TO string
DUNLOAD filename [n]
DUNLOAD filename TO string

Line changing commands

DELETE TO string
MODIFY/string-1/string-2/[G] [n]
GMODIFY
MOVE

Control commands

*[n]
XEQ buffer
PAUSE

Symbol changing commands

PSYMBOL
SYMBOL name character
ERASE character
KILL character

Mode changing commands

VERIFY
BRIEF
MODE PROMPT
MODE NPROMPT
MODE NUMBER
MODE NNUMBER
MODE COLUMN
MODE NCOLUMN
MODE COUNT
MODE NCOUNT
MODE PRALL
MODE PRUPPER
MODE PRLOWER

Value setting commands

LINESZ n
PTABSET tab-1...tab-8
TABSET tab-1...tab-8

Input/output commands

INPUT (ASR)
INPUT (PTR)
INPUT (TTY)
PUNCH (ASR) n
PUNCH (PTP) n

WHAT'S NEXT?

You now know enough to log into the system, create new files with EDITOR, and edit them. Section 4 gives you more PRIMOS-level information on dealing with your files. The EDITOR Reference Section contains complete information on all EDITOR's commands. At this point, you should be able to read through this section, or whatever parts of it you want, and have no trouble learning the additional commands.

MORE PRIMOS COMMANDS

You've learned how to log into your UFD, make and edit files, get a list of these files, and log out, using the LOGIN, ED, LISTF, and LOGOUT commands, there are several other PRIMOS-level commands which can be useful. These commands are:

ATTACH	DELETE
CLOSE	FUTIL
CNAME	SLIST
CREATE	SPOOL
DATE	SORT

USING OTHER UFDS—THE ATTACH COMMAND

In Section 2, we mentioned that once you have logged into your UFD via the LOGIN command, you can not only use the files in your own UFD, but also those in other UFDs. The ATTACH command tells PRIMOS to attach you to a different UFD. Since you can only work from one UFD at a time, PRIMOS will detach you from the UFD you are currently using and then give you access to this new UFD. The format of the ATTACH command is:

ATTACH ufd-name [password]

When you type an ATTACH command, PRIMOS gives you access to this new UFD. If you type the LISTF command, PRIMOS gives you a list of the files in this new UFD. If you create a new file, when you give the FILE command, PRIMOS will put this new file into the UFD to which you are currently ATTACHed. This UFD is known as your current UFD, or current directory.

```
OK, attach sales
OK, listf

UFD=SALES  3 O

FRANK   ALBERT  ROSEMARY      MARTHA

OK,
```

You may give the LOGOUT command while attached to another UFD than your own; PRIMOS will still "sign you out" correctly.

If you specify a UFD that the system cannot find (often the result of a spelling error), the ATTACH command will be ignored and you will get a message of the form:

ufd-name NOT FOUND

If you specify a UFD that has a password, but do not give the password, you will be ATTACHed to the UFD as a non-owner. You can always check this by giving a LISTF command; on the same line as the name of the UFD will be, after a number, either the letter O for owner, or N for non-owner.

When ATTACHed to a UFD as a non-owner, you cannot EDit or SLIST any of its files. You can give the ED command and put text into your work file, but will not be able to FILE it.

To gain owners rights to the UFD under these circumstances, simply give the ATTACH command again, this time including the password after the name of the UFD.

If you misspell the password (possibly because it has been changed), you will get the message: **BAD PASSWORD.**

This will leave you logged in but not ATTACHed to any UFD at all. If you attempt to work on files at this point you will get the message: **NO UFD ATTACHED.**

Don't worry. Just give the ATTACH command again.
The following example illustrates the ATTACH and LISTF commands:

```
login staff money
STAFF   (30) LOGGED IN AT  10'19 011378
OK, listf


UFD=STAFF  3 O


FRANK   MARTHA  ALBERT  ROSEMARY


OK, attach sales
OK, listf


UFD=SALES  3 O


PHONES  SCHEDULE           LIST


OK, attach staff
OK, listf
```

```
UFD=STAFF   3 N


FRANK    MARTHA   ALBERT   ROSEMARY


OK, attach sales
OK, listf


UFD=SALES   3 O


PHONES   SCHEDULE           LIST


OK, logout
STAFF   (30)  LOGGED OUT AT  10'20 011378
TIME USED=  0'1   0'01   0'00
OK,
```

MAKING SUB-UFDS — USING THE CREATE COMMAND

The CREATE command allows you to make subdivisions in your UFD called sub-UFDs. In order to explain the CREATE command, we must first explain the concept of sub-UFDs.

As you recall, your User File Directory is a list of files. A sub-UFD is a selection of those files which have been grouped together. This grouping has a name called a sub-UFD-name. Once you have logged in to your UFD, you can ATTACH to any sub-UFD within it. This sub-UFD will then be your current directory. The LISTF command gives you a list of only those files within this sub-UFD.

Your UFD can contain FILES and sub-UFDs. You can make sub-UFDs within sub-UFDs. A sub-UFD can contain FILES and sub-UFDs.

The command which lets you define and name a sub-UFD is the CREATE command. Whenever you give a CREATE command, PRIMOS creates a sub-UFD within whatever UFD or sub-UFD you are currently in — i.e., if you are working in your UFD, PRIMOS creates a sub-UFD within the UFD; if you are working in a sub-UFD of your UFD, PRIMOS will create a sub-UFD within that sub-UFD. The format of the CREATE command is:
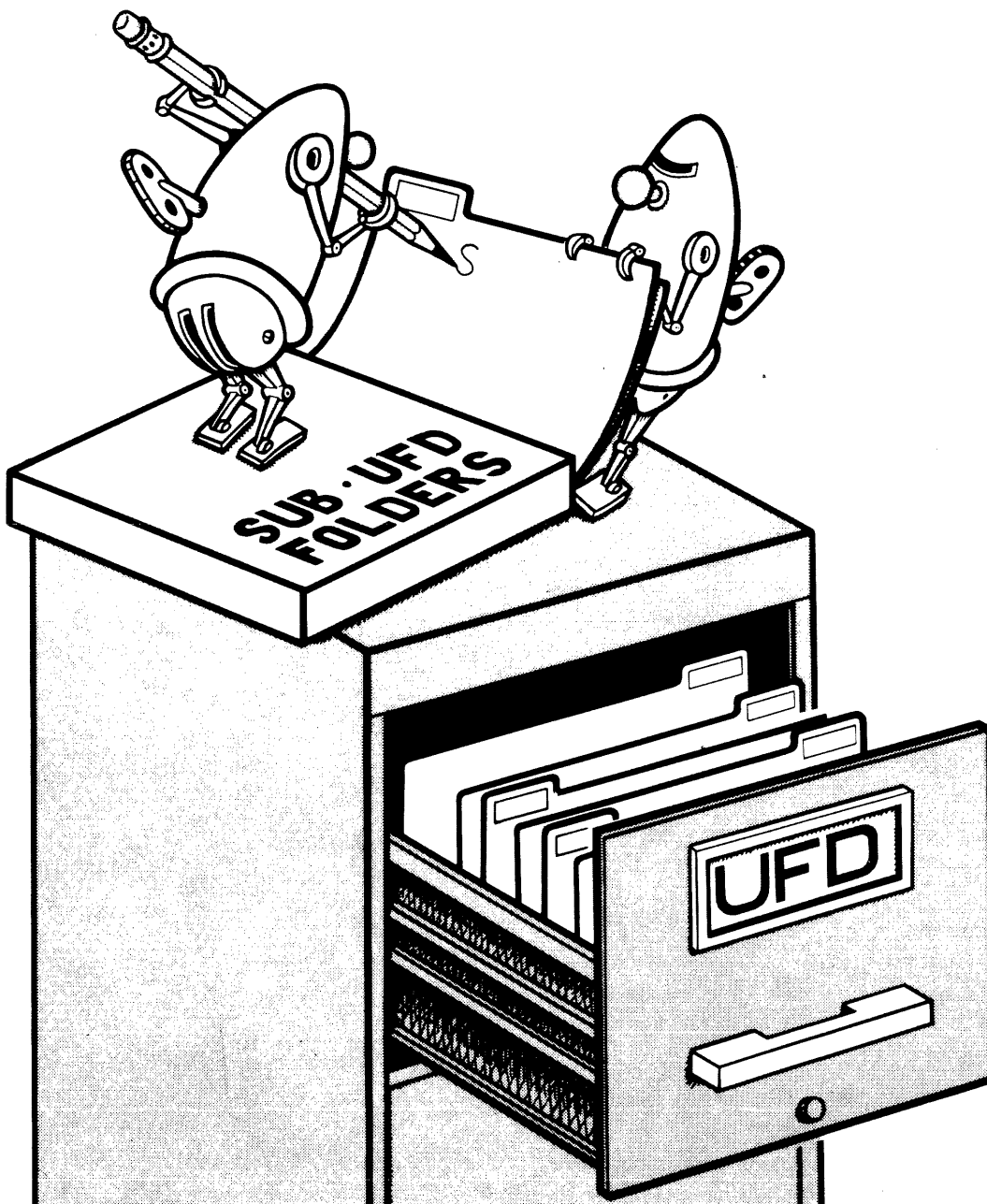
CREATE sub-ufd-name

PRIMOS does not let you have two files or sub-UFDs with the same name in your current UFD. If you attempt to CREATE a sub-UFD with a name already in use, you will get the message: **ALREADY EXISTS**

Using these sub-UFDs

In order to work within a sub-UFD, you must first ATTACH to it. The format of the ATTACH command for sub-UFDs is:

ATTACH sub-ufd-name [password] 1/2

You must include the one-slash-two (1/2) in order to ATTACH to a sub-UFD! This group of characters identifies **sub-ufd-name** as the name of a sub-UFD in your current UFD. If you omit the one-slash-two, PRIMOS will look for a UFD with the given name, and, if it fails to find one, will give the error message: **NOT FOUND**

When you have attached to a sub-UFD, it becomes your current UFD.

You have now learned the two uses for the ATTACH command:

    1. Attaching to a UFD.

    2. Attaching to a sub-UFD within a UFD or sub-UFD.

In order to attach to any other sub-UFD than one in your current UFD (remember that your current UFD may be a sub-UFD!) you must *first* ATTACH to your UFD, and then ATTACH down, sub-UFD by sub-UFD.

Suppose you have a UFD called SALES in which you want to create the sub-UFDs East.Coast and West.Coast. You also want both East.Coast and West.Coast to contain sub-UFDs named North, Central and South. You also want the sub-UFD North in the sub-UFD East.Coast to contain three further divisions: MAIN, BRANCH1, and BRANCH2. Here's how you would do it:

```
attach sales
OK, listf

UFD=SALES   3 O

EAST.COAST      WEST.COAST

OK, attach east.coast 1/2
OK, create north
OK, create south
OK, create central
OK, listf

UFD=EAST.CCAST   3 O

NORTH    SOUTH    CENTRAL

OK, attach north 1/2
OK, create main
OK, create branch1
OK, create branch2
OK, attach sales
OK, attach east.coast 1/2
OK, attach north 1/2
OK, listf

UFD=NORTH   3 0

MAIN     BRANCH1    BRANCH2

OK,
```

LOOKING AT YOUR FILES USING THE SLIST AND SPOOL COMMANDS

The SLIST and SPOOL commands allow you to look at a file in your current UFD (which may be a sub-UFD).

The SLIST command displays a file on your terminal. On CRT type terminals you should have activated CONTROL-S and CONTROL-Q via TERM -XOFF; otherwise longer files will zip past you faster than you can read them. However, if you have a hard-copy terminal nearby, you can get a copy of any file by using this command. (You'll have to login at this terminal, of course.) The format of the SLIST command is:

SLIST filename

The command **TERM -XOFF** permits you to stop and start terminal output on both CRT and hard-copy terminals via **CONTROL-S** (Stop terminal output) and **CONTROL-Q** (Resume terminal output). This permits you to "freeze" a CRT screen—in the middle of an SLIST, for example—so you can examine a portion of your output before it flows off the screen.

Other than CONTROL-Q, the only command you will be able to give after CONTROL-S is a BREAK (CONTROL-P), which will QUIT you both out of CONTROL-S and out of whatever you had been doing.

You activate CONTROL-S and CONTROL-Q by giving the command TERM -XOFF. You de-activate CONTROL-S and CONTROL-Q by giving the command **TERM -NOXOFF**. NOXOFF is the default state, and is also reset by explicit LOGOUT.

If your system has a line printer, you may prefer to use the SPOOL command, particularly if the file is long. This way you can avoid tying up a terminal while printing a long file. Line printers are faster than terminals.

The SPOOL command orders a hard copy of your file to be printed out on the line printer in the computer room. You will have to pick up the print-out when it is ready. The format of the SPOOL command is:

SPOOL filename

When you give this command, PRIMOS makes a note of the filename in the spool queue list for the line printer, and displays the message:

YOUR SPOOL FILE IS PRTxxx

xxx is a 3-digit number that identifies your file in the spool queue list. The reason there is a list, rather than just having each file SPOOLed out as the request comes, is that some requests are very long — hundreds of pages. PRIMOS spools out the shorter files as soon as possible, rather than make the users wait while the long files are printed. The distinction between SHORT and LONG files is determined by the System Administrator, who defines the maximum length of a SHORT file.

The word (SHORT) or (LONG) which follows the SPOOL message tells you which of these categories your file is in.

You can check on the status of your SPOOL request, and get a list of those files in the spool queue, by giving the command:

SPOOL –LIST

For example:

```
OK, spool poem
GO
YOUR SPOOL FILE IS PRT013   (LONG)   REV 14.00


OK, spool -list
GO
```

USER	FILE	DATE/TIME	OPTS	SIZE	NAME	FORM
COSMO	PRT005	11/09 10:03	S	5	RTN168	WHITE
MARTHA	PRT007	11/09 15:34	S	3	RTN172	WHITE
HAMPSO	PRT009	11/11 10:25	S	6	PAGTUR	
LAWLER	PRT010	11/11 10:26	S	9	L_AM9600	
RANDI	PRT011	11/11 10:26	S	1	WKINFO	
MORRIS	PRT012	11/11 10:27	S	3	L_CMP$SR	
ALICE	PRT013	11/11 10:28	L	17	POEM	

If for some reason you decide you do not want your file to be spooled — for example, you gave the wrong filename, or discovered that you still have to correct or change something — you may cancel your spool request with the command:

SPOOL –CANCEL PRTxxx

where **xxx** is the number of your spool file. You will *not* be able to cancel a spool request in the following situations:

1. If PRIMOS has begun spooling your file you will get the message:

   CAN'T CANCEL REQUEST–FILE IS OPEN OR CURRENTLY PRINT-ING.

2. If your file is done SPOOLing, or you gave a non-existent PRT number, you will get the message:

   PRTxxx NOT IN QUEUE

3. If you give the name of the file instead of the PRT number, you will get the message:

BAD PRINT FILE NAME

The SPOOL command has two other useful options: DEFER and FROM.

Since the line printer is a shared resource for all computer users, you do not want to tie it up unnecessarily for long periods of time during working hours. On the other hand, it is not uncommon for users to have files several hundred pages long, which they want SPOOLed out. Using a hard-copy terminal, of course, such a task would take all day; even on the line printer, the job would tie up the printer for a long period of time.

The DEFER option tells the system not to SPOOL the file until after the indicated time; this allows you to enter a SPOOL request at your convenience instead of waiting for the appropriate hour.

You specify the DEFER option for the SPOOL command by using the format:

SPOOL filename –DEFER time

The value for **time** can be expressed either in 24-hour type (00:00 = Midnight) or in 12-hour type, followed by AM or PM (12:00 AM = Midnight). The format is HH:MM, where HH is hours, : is any character, and MM is minutes. If you specify –DEFER but omit time, you will get the prompt:

ENTER DEFERRED PRINT TIME:

The hyphen (–) preceding DEFER *is required.*

If the time you enter is not in the correct format, you will get this prompt again, plus the information prompt:

CORRECT FORMAT IS HH:MM AM/PM.

Line printers usually print on paper which is "single copy" — commonly referred to as "computer paper" or "print-out". However, there may be various other types of paper available for use in the line-printer — for example, 5-copy sets, pre-printed forms (checks, orders, invoices), special sizes or colors of paper.

You can request special forms using the FORM option for the SPOOL command by typing:

SPOOL filename –FORM form-name

form-name is any six-character (or less) combination of letters and numbers which the person who runs the line printer recognizes as identifying one of the forms in stock. Your file will be SPOOLed out when the appropriate form has been loaded into the line printer.

If you specify –FORM but omit form-name, you will get the prompt: **ENTER FORM TYPE:.**

This example shows how to use the FORM option:

```
OK, spool $s2 -FORM WHITE
GO
YOUR SPOOL FILE IS PRT074   (LONG)   REV 15.00
```

RENAMING AND DELETING FILES AND SUB-UFDS

The CNAME and DELETE commands allow you to rename and delete both files and sub-UFDs.

The CNAME command allows you to change the name of a file or a sub-UFD. The format of the CNAME command is:

CNAME old-filename new-filename

If you give a name for **new-filename** which already exists in your current UFD, PRIMOS displays the message:

ALREADY EXISTS

If you misspell the **old-filename**, PRIMOS displays the message:

NOT FOUND
ER!

The DELETE command allows you to delete files and empty sub-UFDs. The format of the DELETE command is:

DELETE $\left\{ \begin{array}{l} \textbf{filename} \\ \textbf{sub-ufd-name} \end{array} \right\}$

You *cannot* delete a sub-UFD which contains files. To delete such a sub-UFD, you must first delete all the files in it. This procedure prevents you from accidentally wiping out a large quantity of work in one unintended command; if you try to delete a sub-UFD which still contains files, you will get the message:

SORT

The SORT command sorts a copy of a file, line by line, in alphabetic order. Refer to the PRIMOS commands guide for full information.

THE DATE COMMAND

You can find out the current date and time by giving PRIMOS the DATE command. The format of the DATE command is:

DATE

THE CLOSE COMMAND

In order for you to use a file, PRIMOS must OPEN it for you (which it always does automatically). The CLOSE command is used to close a file which may have been left open unintentionally—when you left EDITOR, RUNOFF, FUTIL, etc., via the BREAK key or a CONTROL-P. The format for the CLOSE command is:

CLOSE filename

COPYING FILE WITH FUTIL

FUTIL is a subsystem that manipulates files. Refer to either the PRIMOS Commands guide or the PRIMOS Programmer's Companion for full information.

INTRODUCTION

RUNOFF is Prime's text-processing system. It can turn a file like this:

```
 "My dear Fortunato, you are luckily met.  How
remarkably well you are looking
today. But I have received a pipe of what passes
for Amontillado, and I have my
doubts."
 "How?" said he.  "Amontillado?  A pipe?
Impossible!  And in the
middle of the carnival!"
```

into an output file like this:

```
"My dear Fortunato,  you  are  luckily  met.   How
remarkably well you  are looking today.  But I have
received  a  pipe  of what passes for Amontillado,
and I have my doubts."

"How?"   said  he.   "Amontillado?   A   pipe?
Impossible!  And  in the middle of the carnival!"
```

This example shows what RUNOFF can do even if you don't include any explicit commands:

1. It breaks text into paragraphs. (In the absence of other commands, RUNOFF assumes that a line beginning with a space is a new paragraph.)

2. It packs irregular lines of text into uniform lines with a justified right margin.

Unlike EDITOR, where you must explicitly state everything that you want done, RUNOFF does a lot of work for you automatically unless you tell it not to. RUNOFF's defaults include setting up a standard page size, right-justifying your text, para-

graphing a standard way. This means that you don't have to know a lot about RUNOFF in order to use it.

It only takes a few RUNOFF commands to format your output with headers, footers, and indentation. You don't need any commands for paragraphing, pagination and right-justification. These happen automatically.

RUNOFF allows you to change output formats and see them without your having to retype any text whatsoever-all you do is change a command or two. RUNOFF can generate indexes, tables of contents, do decimalized headings, write ~~a series of form letters~~ ~~inserting a different name on each~~, and combine separate files of text and tables into one master output file.

Using RUNOFF is a two-step process, and for this reason, this section is divided in two parts: First, a basic explanation of how to create a RUNOFF source input file, using EDITOR. Next, how to turn this source file into a processed output file.

This section will provide enough information to get you started with RUNOFF. Then, in Section 6, MORE RUNOFF, you can learn the additional commands to aid you in setting up your formats and processing your files. The RUNOFF REFERENCE SECTION contains a complete alphabetized list of the RUNOFF commands, with full explanations of how to use them, and the RUNOFF Summary in Appendix A supplies an easy-to-use table of RUNOFF command formats.

CREATING THE SOURCE FILE

In order to process text, RUNOFF requires both text to process and commands specifying how to process this text.

You have already learned how to make a file of text using EDITOR. Now you will learn how to give the commands that tell RUNOFF what to do with this file.

These commands are called RUNOFF commands. You put RUNOFF commands directly into your text file, using EDITOR, transforming it into a RUNOFF source file.

When you have finished creating your source file, you are ready to give the RUNOFF command to PRIMOS. The RUNOFF command tells PRIMOS to start the RUNOFF program, and gives your source filename. RUNOFF reads this source file, and processes the text according to the commands. The results are placed into a RUNOFF output file. This output file contains your formatted text, and can be SLISTed, SPOOLed, and/or EDITed. This process is illustrated in Figure 5-1.

RUNOFF COMMANDS

RUNOFF commands control the way that RUNOFF processes the text of the source file into the output file. Before we explain the basic RUNOFF commands, here's a brief discussion of RUNOFF's command conventions.

A RUNOFF command consists of a period (.) followed by a command word, and possibly one or more parameters:
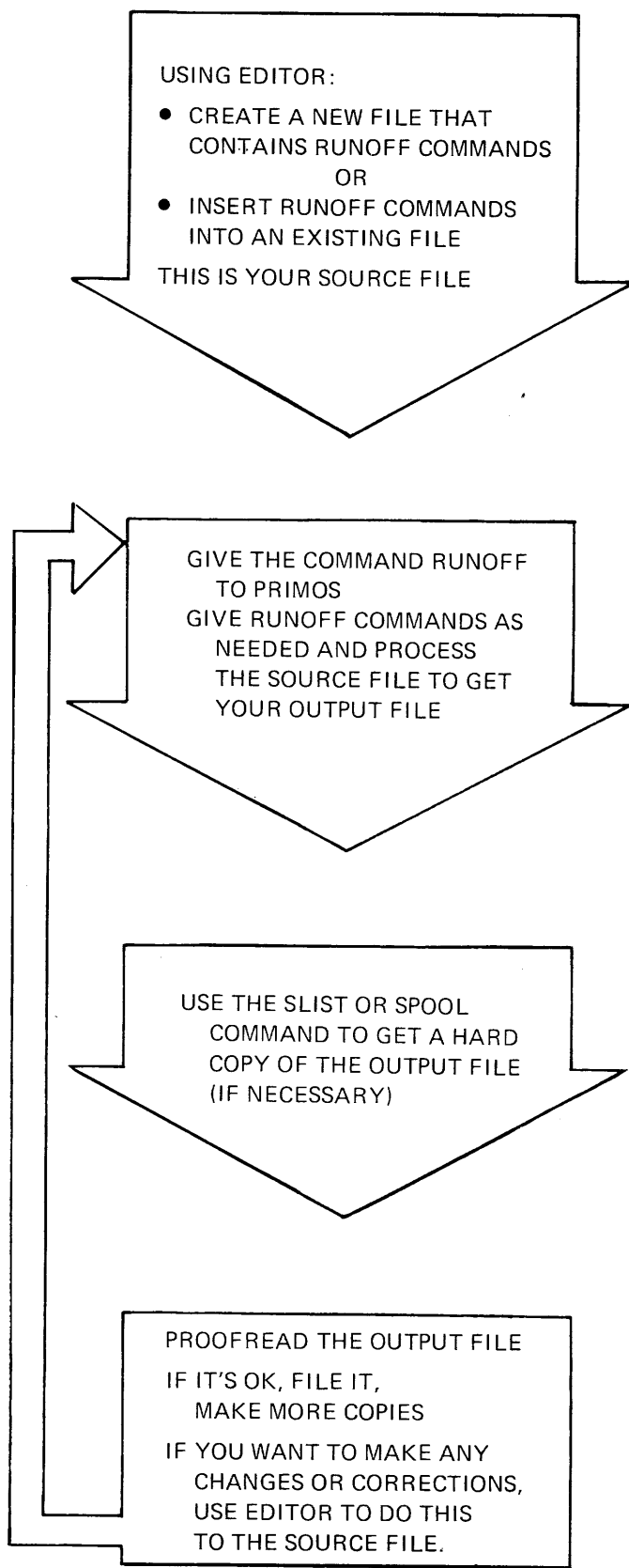
.COMMAND parameter

USING EDITOR:

● CREATE A NEW FILE THAT
  CONTAINS RUNOFF COMMANDS
            OR
● INSERT RUNOFF COMMANDS
  INTO AN EXISTING FILE

THIS IS YOUR SOURCE FILE

GIVE THE COMMAND RUNOFF
  TO PRIMOS
GIVE RUNOFF COMMANDS AS
  NEEDED AND PROCESS
  THE SOURCE FILE TO GET
  YOUR OUTPUT FILE

USE THE SLIST OR SPOOL
  COMMAND TO GET A HARD
  COPY OF THE OUTPUT FILE
  (IF NECESSARY)

PROOFREAD THE OUTPUT FILE

IF IT'S OK, FILE IT,
  MAKE MORE COPIES

IF YOU WANT TO MAKE ANY
  CHANGES OR CORRECTIONS,
  USE EDITOR TO DO THIS
  TO THE SOURCE FILE.

Figure 5-1 Using RUNOFF

The Period: The period signifies that the line contains a command, which is to be obeyed, as opposed to text, which is to be processed. The period must be in column one of the line.

Command Words: A command word is a word which specifies an action. In this manual, the minimum acceptable abbreviation of a RUNOFF command is shown in rust colored letters. For example: **.PARAGRAPH** could be abbreviated in any of the following ways: **.P, .PAR, .PARAGRAPH.** Command words may be in either upper or lower case (or a combination).

Parameters: In RUNOFF, a parameter is either the name of a file, a number specifying a line, column, page number, number of lines or columns, or a piece of text, depending on the command. Numerical parameters are represented in the formats by:

- The letter **i** for a general number, usually of pages
- The letter **m** for a number of spaces
- The letter **n** for a number of lines.

Some commands do not use parameters, some require them, and in others they are optional. In this manual, parameters which are enclosed in brackets – e.g., **[filename]** – are optional. If you omit them, RUNOFF will use the appropriate default value.

RUNOFF commands fall into the following categories:

1. *Page-formatting* – controls the physical layout of pages.
2. *Line-formatting* – controls the processing of words and lines of the input text into lines on the formatted pages.
3. *Output* – specifies how to process the output file.
4. *Characters &symbols* – defines special characters and their meanings.
5. *Features* – indexing, tables of contents, decimalization.

Because of RUNOFF's built-in defaults, a text file containing no RUNOFF commands whatsoever would be processed onto 8-1/2 by 11 inch pages as right-justified text. With the addition of only a few commands, you can do paragraphing, indention, headers and footers, tabbing, and other basic formatting.

PAGE-FORMATTING COMMANDS

Page-formatting commands define the desired margins and physical size of the page, plus headers, footers and page numbering.

Margins and page size

The default page size and margins are illustrated in Figure 5-2; if you do not specify anything, you get this 8-1/2 by 11 inch page, containing 54 lines of 71 character spaces each. The page margins are 7 spaces on the left and right sides, 7 lines on top, and 5 lines on the bottom. (The commands to change these defaults are described in Section 6—MORE RUNOFF.)
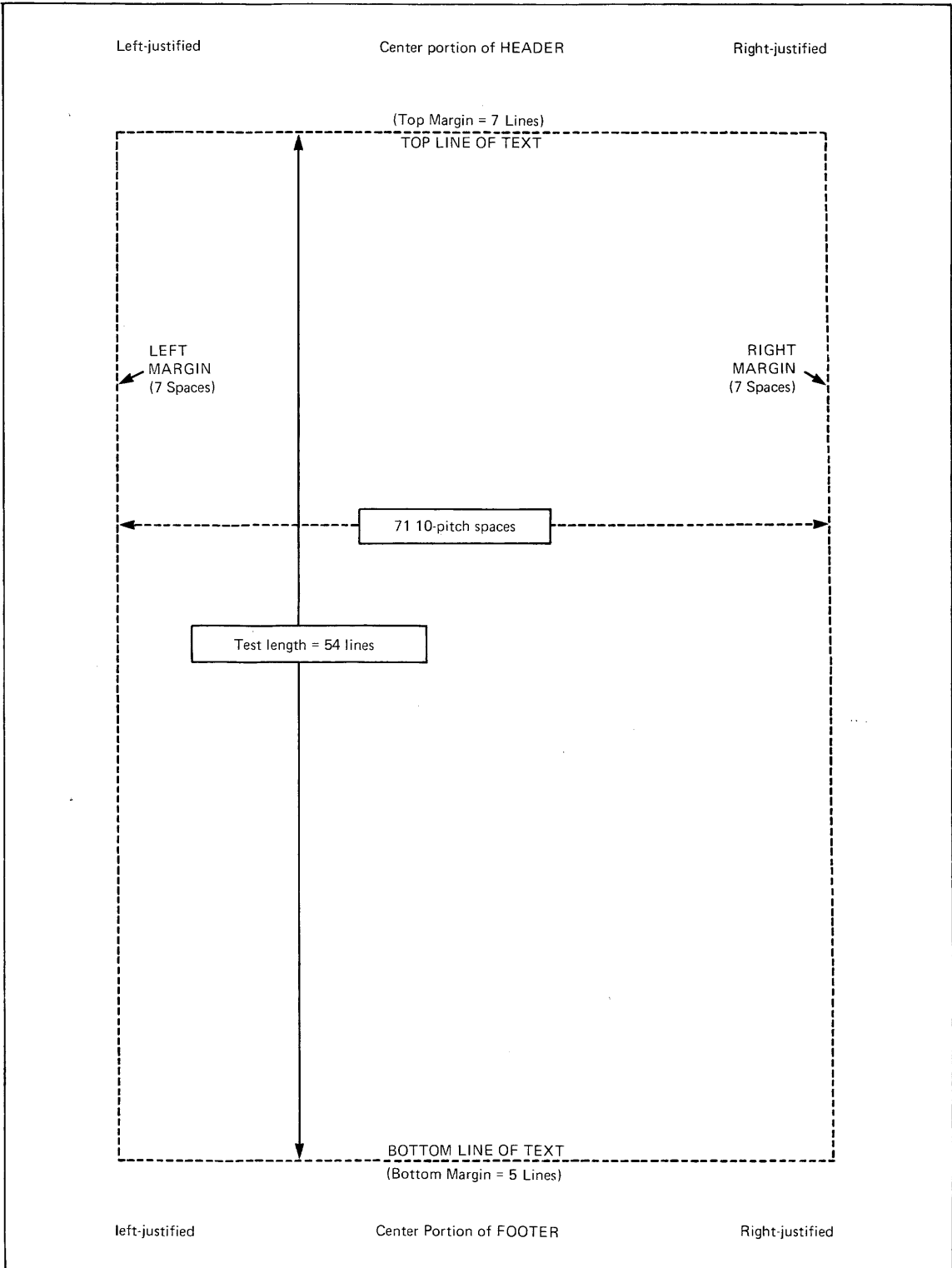
Left-justified   Center portion of HEADER   Right-justified

(Top Margin = 7 Lines)

TOP LINE OF TEXT

LEFT
MARGIN
(7 Spaces)

RIGHT
MARGIN
(7 Spaces)

71 10-pitch spaces

Test length = 54 lines

BOTTOM LINE OF TEXT

(Bottom Margin = 5 Lines)

left-justified   Center Portion of FOOTER   Right-justified

Figure 5-2 RUNOFF Default Page

Headers & Footers

You can set up one-line headers and/or footers centered in the top and bottom margins respectively, by the commands:

      .HEADER/left-text/center-text/right-text/

      .FOOTER/left-text/center-text/right-text/

The headers and footers will start on the first new page after the command. Headers and footers may contain left-justified, centered, and/or right-justified portions. All four delimiters must be in each HEADER or FOOTER command, even if a text portion is left blank.

We have used the slash (/) as the delimiter in the description of all header and footer formats, but you may use *any* character that does not appear in the text portions as the delimiter. This is demonstrated by the examples in the RUNOFF REFERENCE SECTION. The placement of headers and footer is indicated on the default page illustration, Figure 5-2.

Page number

You can insert the current page number into the header or footer by using the pound sign # with the current page number. For example, given this command:

      .HEADER/UNIVERSAL MFG CORP/NEW PRODUCTS/Page #/

the twelfth page of the output file would have this header:

      UNIVERSAL MFG CORP   **NEW PRODUCTS**   **Page 12**

You only need to specify a header or footer once. RUNOFF will print them on all subsequent pages and update the page number automatically. These commands must appear before the first text line of a page in order to be printed on that page.

LINE-FORMATTING COMMANDS

Now you've set up your page format. The next thing to do is to specify the general way you want RUNOFF to process each word or line of text from the source file to the output file. This is done with line-formatting commands.

There are two types of line-formatting commands—general and local. General line-formatting commands specify a manner of processing which is to stay in effect until you specify otherwise. Local line-formatting commands affect only the text at the point where the command occurs.

General line-formatting commands

General line-formatting commands, when given, stay in effect until you explicitly change them. RUNOFF has five commands of this type—SPACE, TAB, and the three modes FILL, ADJUST, and NFILL.

The modes define how RUNOFF will move text from the source file to the output file. You can choose between:

- **FILL** mode – Fill each output line with words from the input file.

- **ADJUST** mode – Right-justify each filled line by adjusting interword spacing.

- **NFILL** mode – Do not fill; transfer text line by line.

The command formats for these modes are:

    .FILL
    .ADJUST   *(default)*
    .NFILL

Since ADJUST mode is the default, you will automatically get right-justified output, unless you specify otherwise.

FILL Mode: FILL mode produces a "ragged right" margin. Both ADJUST and FILL modes *ignore* all RUNOFF tab characters unless the source file line begins with one; they also ignore all extra spaces between words. FILL puts *exactly* one space between words, unless the word is followed by a period (.), semicolon (;), exclamation point (!), question mark (?), or a right parenthesis which was preceded by any of these except the semicolon, in which cases you will get two spaces.

NOTE
_____

The last line of a paragraph is never FILLed.

_____

ADJUST mode: In ADJUST mode, once a line is filled, RUNOFF distributes extra spaces across the line, wherever there are spaces, until the line touches both left and right margins. This is also called "right-justified".

NFILL mode: In NFILL mode, RUNOFF transfers each line of text, *exactly* as it appears in the source file, to the output file. All tabs are obeyed; all multi-column spaces are maintained. NFILL mode is therefore useful in processing tables.

The following examples show how RUNOFF processes a sample source file. The source file:

```
.paragraph 5 1
"My dear Fortunato, you are luckily met.  How
remarkably well you are looking
today. But I have received a pipe of what passes
for Amontillado, and I have my
```

```
doubts."
.paragraph
"How?" said he.  "Amontillado?  A pipe?
Impossible!  And in the
middle of the carnival!"
```

is processed, in ADJUST mode, as right-justified text:

```
        "My dear Fortunato, you are luckily  met.
How  remarkably  well  you  are looking today.
But I have received a pipe of what passes  for
Amontillado, and I have my doubts."

        "How?"  said he.  "Amontillado?  A  pipe?
Impossible!    And  in  the  middle  of  the
carnival!"
```

In FILL mode, the lines are filled with words, but *not* right-justified.

```
        "My dear Fortunato, you are luckily met.
How remarkably well you are looking today.
But I have received a pipe of what passes for
Amontillado, and I have my doubts."

        "How?"  said he.  "Amontillado?  A pipe?
Impossible!  And in the middle of the
carnival!"
```

RUNOFF always uses whichever of these three modes has been most recently specified. If you do not specify a mode, the default, ADJUST, is used.

The other two general line-formatting commands are SPACE and TAB.

The SPACE command: The SPACE command defines the spacing between lines of print—single-space, double-space, triple-space, and so forth. The format is:

.SPACE [n]

If **n** is omitted or 0, the default value of 1 (single-spacing) is used.

The TAB command: The TAB command defines the tab character and tab stop columns relative to the current left margin for a file. As we mentioned earlier, RUNOFF and EDITOR perform tabbing differently. EDITOR inserts the appropriate number of spaces into your file whenever the current EDITOR tab symbol is entered. RUNOFF, however, ignores these spaces when processing in FILL or ADJUST mode. You must explicitly tab in RUNOFF, requiring you to define both your tab character as well as your settings before you may use them. The format of the TAB command is:

.TAB character tab-1 tab-2 . . .tab-8

The eight tab stops must be column numbers, with the higher ones to the right. These settings are always relative to the current left margin. RUNOFF processes the tab character in NFILL mode, or if an input line begins with a tab character. In either case, all characters on an input line are processed. In FILL and ADJUST modes, tab characters are not processed unless the input line begins with one; instead, they will appear in print.

In the following example, the tab character is defined as the @ character and the tab stops are set in columns 7, 27, and 47.

```
.tab @ 7 27 47
.nfill
@In NFILL MODE@All @tab
characters are@recognized
.adjust
@In ADJUST @and FILL@Modes
tab @characters@ are only recognized
if @one begins @ a line
```

processes to:

```
        In NFILL MODE       All                 tab
characters are          recognized
        In ADJUST           and FILL            Modes
tab @characters@ are only recognized if @one begins
@ a line
```

Notice how in ADJUST and FILL Modes, the @ character is not interpreted as a tab character. This is because it was not at the beginning of a line in the source file. The tab character and stops can be redefined at any point in the source file. RUNOFF uses the most recent values.

Tabs are primarily useful for formatting tables.

Local line-formatting commands

Local line-formatting commands have an effect only when you explicitly give them, although they may also set a numerical value for a parameter which will be remembered. Local line-formatting commands insert either a text line or a specified number of blank lines.

.> text	Center the following text.
.+ text	Enter exactly as written even if you are in .ADJUST or .NFILL modes.
./text/text/text/	Apportion text in left, center and right fields (like a header or footer)
.* text	Omit text line (usually a comment) from the output file

The .>**text** format signals RUNOFF to center text on the line. This command is particularly useful for titles and captions.

The .+ **text** format tells RUNOFF to insert this line verbatim—i.e., *exactly* as it appears here. This command permits you to override FILL or ADJUST mode for a single line.

The ./**left-text/center-text/right-text/** format tells RUNOFF to apportion the pieces of text as left-justified, centered, and right-justified portions of text. You may omit any of the text portions, but must still give all four delimiters. The slash (/) is the only permissible delimiter here. This means the / may not be used as a text character. Headers and footers are special cases of apportioning in the top and bottom margins.

The .*****text** format tells RUNOFF to ignore this line completely. This permits you to insert comments within your source file, to document what you have done at a given point.

Here's an example of these four commands: The source file:

```
.*This file contains RUNOFF line-insertion commands.
.>EXAMPLES OF LINE INSERTION
.+This             line inserted    literally.
./Left piece/I'm the Middle/Right On/
```

processes to:

```
                EXAMPLES OF LINE INSERTION
This            line inserted    literally.
Left piece          I'm the Middle          Right On
```

BLANK LINES AND PARAGRAPH SPACING COMMANDS

The commands which insert blank lines are BREAK, SKIP, PARAGRAPH and EJECT. The BREAK command tells RUNOFF to stop filling the current output line at once. The line is ended, and *not* adjusted, no matter what mode you are in. The next text in the source file is put onto a new output line. This action of breaking off the transferring of text to a line is called a BREAK. Many RUNOFF commands, because of what they do, cause a break in the source file. This action of a break without there being a BREAK command is called an implicit BREAK. All the line-insertion commands cause an implicit BREAK before they are processed. The format of the BREAK command is:

.**BREAK**

The SKIP command does an implicit BREAK and then skips n printing lines. The format of the SKIP command is:

.**SKIP [n]**

If **n** is omitted, RUNOFF will skip 1 line.

For example, the source file:

```
          These names of virtue, with
          their precepts, were:
          .skip 2
          .>1. TEMPERANCE.
          .skip 1
          Eat not to dullness; drink not to elevation.
          .skip 2
          .>2. SILENCE.
          .skip 1
          Speak not but what benefit others
          other yourself; avoid trifling conversation.
          .skip 2
          .>3. ORDER.
          .skip 1
          Let all your things have their places; let
          each part of your business have its time.
```

processes to:

```
          These names of virtue,  with  their  precepts,
          were:
                            1. TEMPERANCE.

          Eat  not to dullness;  drink not to elevation.


                            2. SILENCE.

          Speak not  but  what  benefit   others   other
          yourself;  avoid trifling conversation.


                            3. ORDER.

          Let  all  your  things have their places;  let
          each part of your business have its time.
```

The PARAGRAPH command tells RUNOFF to do an implicit BREAK, and then indent m spaces after skipping n printing lines. The format for the PARAGRAPH command is:

.PARAGRAPH [m] [n]

The default values for **m** and **n** are 0 spaces, 1 line. If no values are given, RUNOFF will use the most recently specified values. The value of m can be positive or negative; for a full discussion, see Section 8, SAMPLE SESSIONS.

Here's an example of the PARAGRAPH command. The source file:

```
          "...You were not to be found, and I was
          fearful of losing a bargain."
          .paragraph 5 1
          "Amontillado!"
          .paragraph
          "I have my doubts."
```

```
.paragraph
"I have my doubts."
.paragraph
"Amontillado!"
.paragraph
"And I must satisfy them."
.paragraph
"Amontillado!"
.paragraph
"As you are engaged, I am on my way to Luchresi.
If anyone has a critical turn it is he..."
```

processes to:

```
        "I have my doubts."

        "Amontillado!"

        "And I must satisfy them."

        "Amontillado!"

        "As you are  engaged,  I  am  on  my  way  to
Luchresi.  If  anyone  has  a critical turn it is
he..."
```

NOTE
_____

In both FILL and ADJUST modes, if the first character of an input line is a space, RUNOFF acts as if you gave a PARAGRAPH command here. This is known as implicit paragraphing.

_____

The EJECT command signals the end of a page, in the same way that BREAK signals the end of a line. The format of the EJECT command is:

.EJECT

The EJECT command does an implicit BREAK, and skips to the beginning of a new page. There are several commands which do an implicit EJECT (which includes doing an implicit BREAK.)

OUTPUT COMMANDS – THE TTY COMMAND

The TTY command causes RUNOFF to display the processed output file on your terminal. If you have a printing terminal, you will get a "hard-copy" of the output file. The format of the TTY command is:

.TTY

You must specify TTY if you want the output displayed as it is processed. It can appear anywhere in your source file.

Now that you've learned the basic RUNOFF commands, you're ready to process your source file by running RUNOFF.

Here's a sample RUNOFF source file which uses all the commands that you have learned.

```
.*This file contains the monthly report of the
.*Universal Mfg. Corp.
.tty
.header/Monthly Report/UNIVERSAL MFG CORP/Page #/
.footer//In-House Only//
.tab @ 10 22 40
.space 2
.skip 1
.>UNIVERSAL MANUFACTURING CORPORATION
.>Report for the month of: July 1984
.space 1
.+Summary
.fill
.paragraph 5 1
Although sales this month have not met up with previous
expectations, our staff in Product Development have
come up with some real rousers which, we are
pleased to announce, should guarantee an upward-spiraling
sales trend for the year to come.
.paragraph
These new products include:
.skip 1
./Euphorics/Domestic Androids/Microwave Freezers/
./Cloned Pets/Reversible Umbrellas/Metric Boots/
./Smart Cars/Wide-beam Lasers/Pet Moon Rocks/
./Portable Windmills/Ultimate Mousetraps/Air Sensors/
./Thinking Caps/Ersatz Sawdust/Presidential Mugs/
./Biorhythmic Sheet Music/"Sadness"/Featherless Chickens/
.skip 1
.paragraph
The schedule of releases is:
.skip 1
.nfill
@MONTH @PHASE @PRODUCTS
.skip 1
@September @Rumours @Androids, freezers, rocks
@@Press Kit @Boots, cars, pets
.skip 1
@October@Follow-up@Freezers, rocks, cars
@@Public demo@Cars, umbrellas
.skip 1
.adjust
.paragraph
In addition, Universal Manufacturing is introducing an entire
new line of products---complete tailored-to-order
universes in customer's choice of Einstein,
de Sitter and Lobeshevsky curvature tensors.
.break
.+*****WE WILL MAKE 'NESTED' CUSTOM ORDERS****
Our only stipulation will be that we must be part of any and
all orders.
```

RUNNING RUNOFF (PROCESSING YOUR SOURCE FILE)

Once you've put the desired RUNOFF commands into your source file, you are ready
to process the file through RUNOFF. The sequence for processing a source file is:

1. If you have finished EDITing your source file by inserting RUNOFF
   commands, but are still in EDITOR, save this file by giving EDITOR
   the FILE command. (Make sure you give a new filename, unless you
   are willing to delete the previous file of the same name.

   file form.letter
   OK,

2. Give PRIMOS the RUNOFF command. This tells PRIMOS to get the
   specified file and process it using RUNOFF. The format of the
   RUNOFF command is:

   RUNOFF filename

   You may give the name of the file that you just finished EDITing, or
   the name of any other file in your current directory that contains
   RUNOFF commands. PRIMOS will respond with the word GO, and
   start RUNOFF. RUNOFF will announce itself by displaying the word
   "RUNOFF" plus the version number of RUNOFF which is on your sys-
   tem:

   ```
   OK, runoff s2
   GO
   RUNOFF REV 15
   COMMANDS
   ```

3. If you did not specify a filename when you gave the RUNOFF com-
   mand, RUNOFF will *immediately* ask for a source file:

   INPUT FILE:

   At this point, give the name of the file that you want processed. If you
   misspell the filename—i.e., RUNOFF cannot find any file by that
   name in your current directory—you will be returned to PRIMOS with
   the ER! prompt, and will have to give the RUNOFF command all over
   again.

4. Once RUNOFF finds the specified source file, it enters command
   mode, which it indicates by displaying the word "COMMANDS". You
   may now give RUNOFF commands directly from your terminal. Com-
   mands entered in command mode need not be preceded by periods.

   RUNOFF signals that it is ready to accept a command by displaying
   the command mode prompt, which is the dollar sign ($). When you

are done giving additional RUNOFF commands, and are ready for your source file to be processed, hit the ⌈RETURN⌉ key. This tells RUNOFF to begin processing.

RUNOFF acknowledges by displaying the word "PROCESSING . . .". This part of the sequence looks like:

> **COMMANDS**
> **$header/UNIVERSAL MFG CORP/NEW PRODUCTS/Page # /**
> **$spaces 2**
> **PROCESSING . . .**

5. In order to process your source file, RUNOFF needs to know the name of the output file it is about to create. If you have not given an output filename in the source file, RUNOFF asks for one:

> **ENTER OUTPUT FILE TREENAME:**

You may specify an existing name, if you want to overwrite a previous version. RUNOFF asks you to verify that you want to delete the previous version by querying:

> **OK TO DELETE OLD filename?**

If you answer anything but YES, Y, YE, OK, etc., RUNOFF says:

> **PLEASE SPECIFY NEW FILENAME:**

6. RUNOFF will process the source file and create an output file. This output file is placed in your current directory. If you give the TTY command, it is displayed at your terminal. You can also inspect it via EDITOR, SLIST, or SPOOL. If there were no command errors in your source file, RUNOFF returns you to PRIMOS with an OK prompt. (RUNOFF command errors are explained later.)

Here's a quick summary of the processing sequence:

1. Input, edit and file a source text file which includes RUNOFF commands, using EDITOR.
2. Give PRIMOS the RUNOFF command.
3. If you did not give the name of the source file, do so in response to the prompt INPUT FILE.
4. Give RUNOFF commands from terminal, if desired. Hit the ⌈RETURN⌉ immediately after the $ prompt to begin processing.
5. Give name of output file, if not previously done via RUNOFF's FILE command.

6. Inspect output file. If there were no command errors, proofread text. If there were, return to source file and make command corrections, via EDITOR.

The sequence of processing might look like this on your terminal:

```
runoff s2
GO
RUNOFF REV 15
COMMANDS
$ tty
$ space 2
$ [ RETURN ]
PROCESSING...
```

RUNOFF COMMAND ERRORS

RUNOFF can detect errors which you make in giving RUNOFF commands. RUNOFF *cannot* detect errors in your source text; this is processed exactly as it appears in your source file. It's up to you to catch errors in spelling, punctuation, grammar, etcetera.

Command errors can occur either in your source file or when you are giving commands in RUNOFF's command mode. The errors made in command mode are detected instantly; those in the source file are detected when the line containing the error is processed.

A common source of errors is unintentional characters or words in your source file which you inadvertently entered in EDITOR by trying to give commands while you were in INPUT mode. If you did not delete these unwanted lines, you will get unexpected errors when you try to process the file through RUNOFF. Another common source is forgetting the periods in front of commands in your source file.

RUNOFF has two kinds of command errors: illegal commands and unrecognized commands.

Illegal commands

Illegal commands are commands which RUNOFF can understand but not carry out. For example, the commands .TAB @ 7 30 12 or .PARAGRAPH 300 can be interpreted, but the values of the parameters are invalid.

Unrecognized commands

Unrecognized commands are commands which RUNOFF does not recognize. The typical reasons for this are that you either misspelled a RUNOFF command word, or used a *non*-RUNOFF command by mistake – e.g., ATTACH, WHERE, DUNLOAD.

RUNOFF calls to your attention any unrecognized commands by printing out the command word followed by the message: **UNRECOGNIZED** at your terminal.

For both illegal and unrecognized commands, RUNOFF inserts, at the current line of the output file between a pair of blank lines, the line from your source file which caused the error:

```
          Pickeral Frogs are more aquatic than  Leopard
     Frogs.  They  like to stay in the neighborhood of
     some large stream or body of water.


paragraph 200 1


     Around the  Great  Lakes  Region,  Pickeral  Frogs
     represent  perhaps  40
     sections.
```

When RUNOFF detects a command error, after printing out the bad command and the error message, it returns to RUNOFF command mode. Your choices at this point are:

- Type QUIT, and correct the error in the source file via EDIT.
- Type the correct command, plus a carriage-return to continue the processing.
- Type a carriage-return to continue processing without correcting the error.

In order to get an error-free output file, you will have to make the appropriate corrections in the source file, by means of EDITOR. The only reason you might have for continuing to process a source file once RUNOFF detects an error is to locate any further errors.

An output file which contains error messages will be useful *only* for determining the results of your errors.

If RUNOFF detects any errors in your source file, it returns you to PRIMOS by this prompt:

```
     *****ERROR(S)*****
     ER!
```

instead of the regular "OK," prompt.

DONE: THE PROCESSED OUTPUT FILE

Now you're done. You've created a source file of text and RUNOFF commands, and processed this file through RUNOFF to get an output file of formatted text.

All that remains for you to do is proofread your RUNOFF output — to see if the format suits you and to make sure there aren't any spelling or formatting errors, etc. If you do find things you wish to change:

1. Give the ED command, plus the name of the source file.

2. Make appropriate changes in the source file.

3. FILE this, under the old name, or a new one.

4. Give the RUNOFF command, with the name of the corrected source file.

5. Check the new version of the output file.

6. Repeat steps 1-5 until you are satisfied.

7. Make a "hard" copy of the output file, either on the line printer via the SPOOL command, or on a printing terminal by giving the TTY command to RUNOFF or using the SLIST command.

8. Keep whichever of the source and/or output files you want; delete those you do not want.

9. Check your current directory with a LISTF command to make sure you have only those files you want.

10. You can now log out, or proceed to a new task.
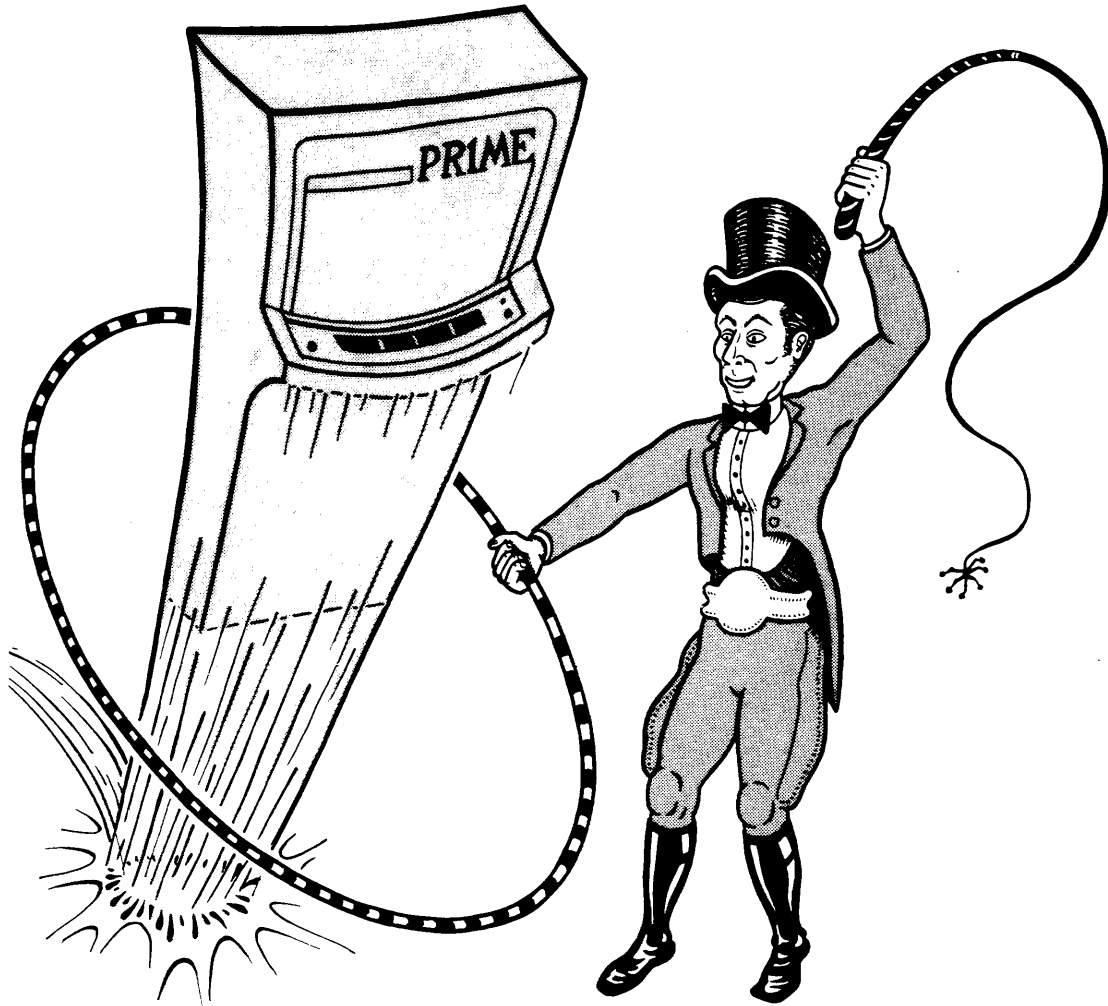
INTRODUCTION

Before you read through this section, you should be familiar enough with the material in Section 5, THE ESSENTIALS OF RUNOFF, to use those basic RUNOFF commands without any trouble.

This section explains the remaining RUNOFF commands, excluding those which deal with tables of contents and decimalization (these are explained in Section 7). The commands are grouped by function, and are explained sufficiently for you to be able to use them. If you need more information about a particular command, consult the RUNOFF Reference Section. The remaining commands fall into these categories:

- *Page-formatting* – specifies page size, margins, indentations, text columns, and even and odd headers and footers.

- *Output options* – controls the manner in which you get output.

- *Special characters, symbols and conventions* – controls RUNOFF's reserved characters, definable symbols, and underlining.

- *Art and file-insertion* – reserves space for art and tables, and insert other source files into the output file.

- *Indexing.*

PAGE-FORMATTING COMMANDS

RUNOFF's page-formatting commands allow you to define any and all of the following:

- Length and width of physical page.

- Top, bottom, side and inter-column margins.

- Left and right indention.

- Number of text columns per page.

- Different headers and footers for even and odd numbered pages.

Of course, you don't have to give any of these commands if you want to use the default format of the page (Figure 6-1).

Left-justified   Center portion of HEADER   Right-justified

(Top Margin = 7 Lines)

TOP LINE OF TEXT

LEFT
MARGIN
(7 Spaces)

RIGHT
MARGIN
(7 Spaces)

71 10-pitch spaces

Test length = 54 lines

BOTTOM LINE OF TEXT

(Bottom Margin = 5 Lines)

left-justified   Center Portion of FOOTER   Right-justified
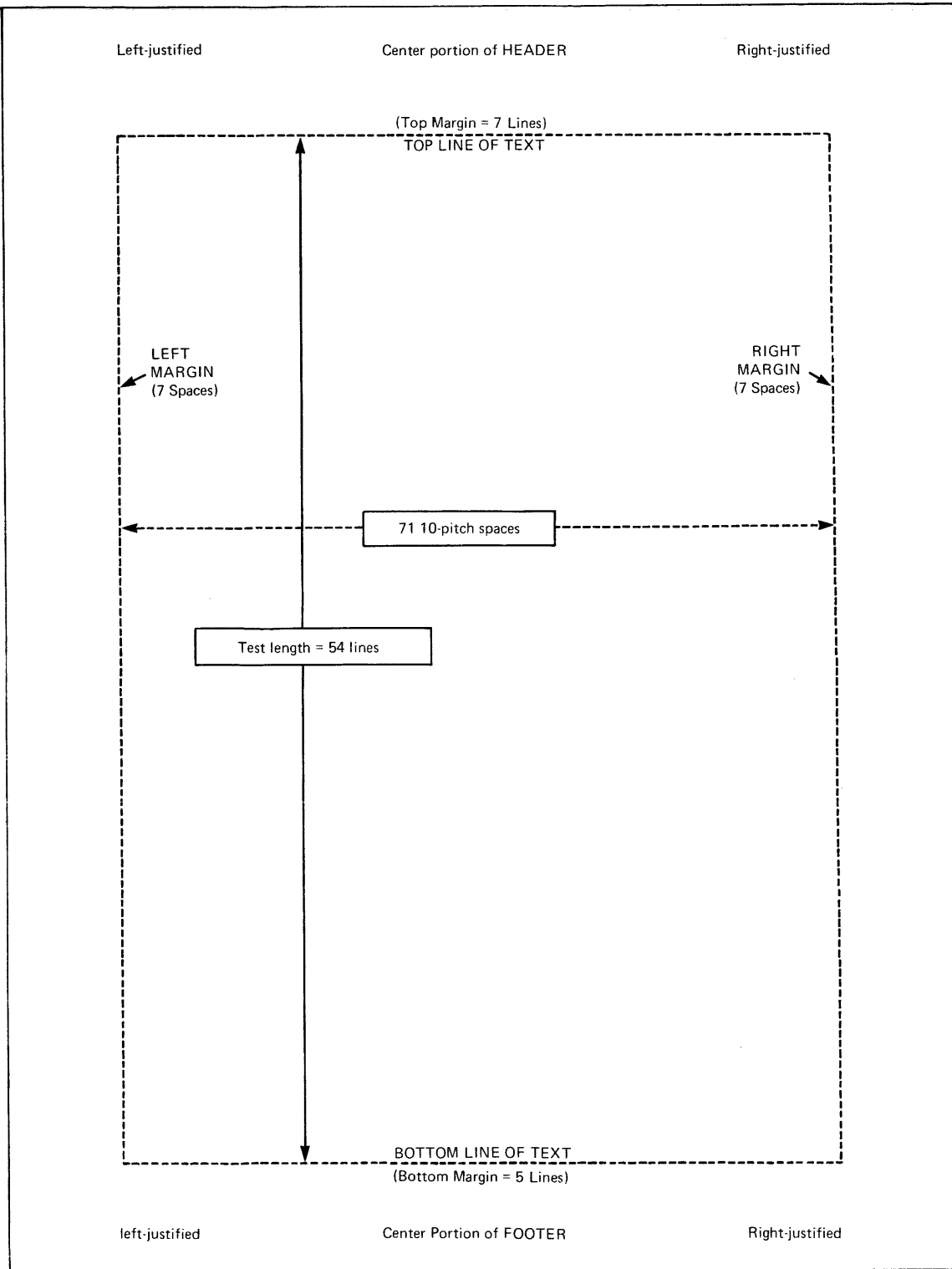
Figure 6-1 RUNOFF Default Page

Page size commands

You can define the physical size of the page with these two commands:

.LENGTH [n]
.WIDTH [m]

The LENGTH command defines the length of the physical page, including the top and bottom margins, as **n** lines. If no value of n is specified, RUNOFF uses the default value of 66 lines. (At the standard of 6 lines = 1 inch, 66 lines = 11 inches.) The maximum length is 132 lines (22 inches).

The WIDTH command defines the width of the physical page, including the left and right margins, as **m** spaces. If no value of m is specified, RUNOFF uses the default value of 85 spaces. RUNOFF works in pica spaces, which are 10 per inch. This is known as "10-pitch type". Although you may be able to vary the type-pitch on certain hard copy printers, RUNOFF always acts as if it were printing 10-pitch type. The maximum allowable page width is 170 spaces (17 inches).

Both the WIDTH and LENGTH commands cause an implicit BREAK and EJECT. The default page size is 8½ by 11 inches.

Margin commands

You can define any of the margins on the page by means of these commands:

.TMARGIN [n]
.BMARGIN [n]
.SMARGIN [m]
.CMARGIN [m]

The TMARGIN command sets the top margin to **n** lines from the top of the physical page. If n is omitted, the default value of 7 lines is used.

The BMARGIN command sets the bottom margin to **n** lines from the bottom of the physical page. If n is omitted, the default value of 5 lines is used.

The SMARGIN command sets the side margins (i.e., left and right) to **m** spaces from each side of the physical page. If m is omitted, the default value of 7 spaces is used.

The CMARGIN command can only be used when you have at least 2 columns of text on the page. (See the COLUMNS command.) This command sets the inter-column margin — i.e., the number of spaces between columns — to **m** spaces. If m is omitted, the default value of 5 spaces is used.

All of the margin commands cause both an implicit BREAK and EJECT. the default margin settings, if none of these commands are given, are:

1. Top = 7 lines

2. Bottom = 5 lines

3. Sides = 7 spaces

4. Inter-column = 5 spaces (if two or more columns of text are specified)

Indention

The indention command allows you to move the left and right margins individually and temporarily. Indention is always relative to the most recent setting. (Remember that your tab settings will always be *relative* to the current left margin). The indention commands are:

        .INDENT [m]
        .RINDENT [m]
        .UNDENT [m]
        .RUNDENT [m]

The INDENT command moves the current left margin **m** spaces to the *right*. If m is omitted or zero, RUNOFF indents by the default value of 5 spaces.

The RINDENT command moves the current right margin **m** spaces to the *left*. If m is omitted or zero, RUNOFF uses the default value of 5 spaces.

The UNDENT command moves the current left margin by **m** spaces to the *left*. If m is omitted or zero, RUNOFF resets the left margin back to the value specified in the most recent SMARGIN command, or, if none was given, to the default margin of 7 spaces from the side of the page.

The RUNDENT command moves the current right margin by **m** spaces to the *right*. If m is omitted or zero, RUNOFF resets the right margin back to the value specified in the most recent SMARGIN command, or, if none was given, to the default margin of 7 spaces from the side of the page.

Columns of text

The COLUMNS command allows you to have more than one column of text on a page. The format of the COLUMNS command is:

        .COLUMNS [i]

The default setting is one column of text per page. If you use COLUMNS to define more than one column of text, the default margin between these columns is five spaces. This can be changed via the CMARGIN command. The COLUMNS command causes both an implicit BREAK and EJECT.

Here's an example of 2-column text:

            an   ancient   sage   I address myself to   all
boasted, that, tho' he could not   ministers   who   have   the
fiddle, he knew how to make a   management   of   extensive
great city of a little one. The   dominions, which from their very
science   that   I,   a   modern   greatness are become troublesome
simpleton,   am   about   to   to   govern,   because   the
communicate,   is   the   very   multiplicity of their   affairs
reverse.   leaves no time for fiddling.

Page headers and footers

You can specify *different* headers and footers for the even and odd-numbered pages by using the following commands:

.EFOOTER/left-text/center-text/right-text/
.EHEADER/left-text/center-text/right-text/
.OFOOTER/left-text/center-text/right-text/
.OHEADER/left-text/center-text/right-text/

The rules for these commands are the same as for the HEADER and FOOTER commands:

1. Any character not in the text portions may be used as the delimiters.

2. Text portions may be omitted.

3. All four delimiters must appear in a given command.

4. Each command affects all pages after it is given.

OUTPUT OPTIONS

Output-option commands determine what RUNOFF will do with the output file. These choices include:

- Specifying a name for the output file, or having no output file.
- Processing only selected pages of the output file.
- Numbering the lines of the output.
- Pausing and/or perforating between each page of output.
- Suppressing command error messages.

Output-option commands are usually given in RUNOFF's command mode.

Naming the output file

The FILE command specifies the name of the output file, to which all subsequent text is processed. This means that the FILE command *must* come before any text-formatting commands and/or text in the source file. The format of the FILE command is:

.FILE [filename]

No output file

The NFILE command tells RUNOFF that you do not want any output file saved. The format of this command is:

.NFILE

RUNOFF processes the source file, but does not save the results. This command has little use, except when you are also giving the TTY command.

Processing selected pages

The FROM and TO commands permit you to select specific range of pages to be processed from a longer source file. The format of these commands is:

.FROM [i]
.TO [j]

You may give either, both or none of the FROM and TO commands.

If you give the FROM command, page **i** of the processed output will be the first page displayed and/or filed. If you do not give the FROM command, RUNOFF begins with page 1.

If you give the TO command, page **j** will be the last page of the processed output to be displayed and/or filed. If you do not give the TO command, RUNOFF will continue until the entire source file has been processed.

If you have reset the page numbering by means of the PAGEN command, RUNOFF compares the specified number(s) against this page count; otherwise, RUNOFF compares them against the sequential page count. (The page number need *not* physically appear on the pages to use FROM or TO.)

For example:

```
OK, runoff s2
GO
RUNOFF REV 15
COMMANDS
$ from 2
$ to 6
$ tty
$
PROCESSING...
```

Numbering the lines of output

The SOURCE command tells RUNOFF to put line numbers which correspond to lines in the source file by the side of the output file lines. The format of the SOURCE command is:

.SOURCE [n]

When the SOURCE command is given, RUNOFF begins generating a list of line numbers one space to the right of the right margin of the output file. Each number corresponds to a non-blank text line from the source file; command lines from the text file, as they do not appear in the output file, are not numbered. If you are producing multi-column text, each column will have corresponding line numbers to the right of it.

Each line number is **n** greater than the previous one. If the value of n is 1 or omitted, the line numbers increase by 1. If you specify a value of zero for n, RUNOFF stops inserting line numbers.

BLOCKS OF TEXT, ARTWORK AND INSERTED FILES

RUNOFF has commands which allow you to reserve groups of lines on pages for blocks of text or artwork, plus commands which allow you to insert text from other source files.

Blocks of text

There are two RUNOFF commands that reserve space on the page for text which cannot be split between two pages. These commands are:

> .NEED [n]
> .WIDOW [n]

The NEED command indicates that **n** lines must appear in a continuous group on the page. When RUNOFF processes a NEED command, it immediately checks to see if there are n lines left on the page before the bottom margin. If so, then RUNOFF proceeds as usual. If there are less than n lines remaining, RUNOFF does an immediate BREAK and EJECT, putting n lines of text at the beginning of a new column or page. If n is zero or omitted, the default value of 1 is used.

The WIDOW command permits you to specify a minimum number of lines in a paragraph which may appear at the bottom of a page.

While the NEED command is local, WIDOW is general. Once you give the WIDOW command, RUNOFF checks the text for the bottom **n**+1 lines of each page of output before finishing the processing of the page. If there is a BREAK, SKIP or PARAGRAPH command in the corresponding source text, RUNOFF does an EJECT just before such commands, and continue processing the text on a new page.

Artwork

The PICTURE command allows you to reserve blank lines in a continuous block. This is particularly useful if you plan to insert items into your document which cannot be produced by RUNOFF. This includes artwork, drawings, tables set in different type, and examples. The format of the PICTURE command is:

> .PICTURE [n]

The **n** in this command always refers to physical lines — not n times the current value of SPACE. When RUNOFF encounters a PICTURE command, it checks if n lines remain on the page. If so, RUNOFF fills the current output line, if you are in FILL mode, or fills and adjusts the line if you are in ADJUST mode. It then does a BREAK and SKIPs n physical lines.

If n physical lines are *not* available on the page, RUNOFF continues to process text, and SKIPs n physical lines at the top of the new page before continuing to process text. If n is larger than the total number of physical lines between the top and bottom margins, RUNOFF skips additional lines and/or pages until all n lines are skipped.

You can specify more than one PICTURE request at a time. RUNOFF keeps track of up to ten values of n at the same time, and attempts to put one per page in the order that each PICTURE was specified. Each page is filled with text below the reserved space.

Inserting files

RUNOFF has two special commands which allow you to use additional source files to create a single output file. The commands which insert other source files are INSERT and FLOAT. The INSERT command inserts an entire file at the place where the command is given, like NEED. FLOAT inserts an external file where the space is available, like PICTURE. INSERT and FLOAT are explained fully in the RUNOFF REFERENCE SECTION.

SPECIAL CHARACTERS, SYMBOLS AND CONVENTIONS

Like EDITOR, RUNOFF has a number of special characters with specific functions. There are commands to change the value of these reserved characters. RUNOFF's special characters, plus their defaults and meanings, are:

Name	Default	Meaning
BLANK	CONTROL-@	Required blank
ERASE	"	Erase previous character (Command mode only)
HYPHEN	RUBOUT	Permissible hyphenation point
KILL	?	Erase (Kill) entire line (Command mode only)
PAGEN	#	Current page number (headers and footers only)
SYCHAR	%	Delimits symbol-names in text. Enter literal SYCHAR by inputting two in a row (%%)
TAB	none	Move to the next "tab stop", (relative to the left margin)

The commands to change the value of reserved characters are BLANK, ERASE, HYPHEN, KILL, SYCHAR, and TAB. The page number character *cannot* be changed.

The BLANK command resets the value of the blank character. Each blank character indicates a required space in the source file and indicates words which must not be broken. These blanks are neither suppressed nor padded during FILLing and ADJUSTing. The formats for the BLANK, ERASE and KILL commands are:

> **.BLANK character**
> **.ERASE character**
> **.KILL character**

RUNOFF's default settings for the ERASE and KILL characters are the same as those used by PRIMOS and EDITOR — unless, of course, they have been changed on your particular computer. The default value of the erase character is the double-quote ("); the default value of the kill character is the question-mark (?). These two characters can be used only in RUNOFF's command mode. Otherwise, literal double-quotes and question-marks in your source text would be interpreted as erase and kill commands. You can give the value-changing commands BLANK and KILL within your source file.

NOTE

Once you have changed either of these two characters, you cannot return them to the default setting within the RUNOFF session. They will be reset when you leave RUNOFF.

The format for the HYPHEN command is:

.HYPHEN character

This command defines the value of RUNOFF's phantom hyphen. The phantom hyphen can be used to indicate places where RUNOFF may hyphenate a given word of text if the word is too long to fit on an output line. See the RUNOFF REFERENCE SECTION for more information.

The SYCHAR command defines the delimiter for symbol-names, which are explained in the following paragraphs. The format for the SYCHAR command is:

.SYCHAR character

The default value of **character** is the percent sign (%).

Defining and using symbols

There may be occasions when you want to leave a "hole" in a document which you intend to fill in later. This "hole" could be a page number which refers to a section you have yet to write, the address and name of a customer, the date, or many other things.

RUNOFF allows you to leave such "holes" in your source file, and gives you a way of labeling each of these so that you can define the contents of the missing material elsewhere in your source file, or in command mode. You can label each place to be filled in by means of symbols.

Each symbol has a name and a value. You insert the name enclosed in percent signs, at the appropriate place in your source file. You define the value of the symbol either in command mode or within the source file prior to the use of the symbol-name. RUNOFF inserts the specified value of a symbol in the output file whenever its name appears in the source file.

You define the name and value of each symbol by means of the DEFINE command. The format of this command is:

.DEFINE symbol-name value

Whenever RUNOFF finds **symbol-name** enclosed in a pair of percent signs in your source file, it replaces it in the output file, with the corresponding value as defined by a DEFINE command. The value may be text, a parameter, or even a command.

For example, the source file statements:

    .DEFINE Name Arthur Ingrid
    .+DEAR %Name%

will be processed as:

    Dear Arthur Ingrid,

The rules for defining symbols are as follows:

1. RUNOFF distinguishes between upper and lower case letters in symbol-names. This means that, TITLE, Title, and title are different symbol-names.

2. The name of a symbol may be of any length — however, RUNOFF looks only at the first six letters. This allows you to use longer names for your own reference, but note, for example, that the names: Expedite and Expedition would appear to be the same, (Expedi).

3. You *cannot* use commas, parentheses or spaces in a symbol-name.

4. The symbol-value may contain up to 30 characters.

5. You may have up to sixty different symbols defined at a given time. If you need additional symbols, you may redefine an existing symbol-name whose old value is not longer needed, via the DEFINE command. You can undefine one or all symbols without giving new values with the UNDEFINE command. (See the RUNOFF REFERENCE SECTION for details.)

6. You may redefine the delimiting character with the SYCHAR command.

To insert a literal percent-sign in the text – e.g., 95% pure – type two percent-signs together: i.e., 95%% pure.

Special conventions

RUNOFF has special conventions to cover two particular text-formatting circumstances: beginning a line with a period, and underlining.

Initial text periods: Since a period at the beginning of a line signifies a RUNOFF command, you need a special way to indicate that you want a text line to begin with a period. You do this by typing two periods in a row; RUNOFF processes this pair as a single text period which begins a line of text in the output file.

So, for example, the lines:

```
..INSERT FOUR
..PARA
```

process to:

```
.INSERT FOUR
.PARA
```

Underlining: RUNOFF permits you to underline any portion of a line of text. A pair of left braces ({{) indicates the beginning of text to be underlined; a pair of right braces (}}) indicates the end.

MORE RUNOFF 6

For example:

```
{{Underline me}}
```

processes to:

Underline me

RUNOFF allows you to underline, in addition to standard words, sentences, lines and paragraphs:

- Punctuation marks
- Parts of words
- Centered text
- Headers, footers, and apportioned text
- Blanks
- Decimal headings

You *cannot* underline RUNOFF commands. This means you must be careful not to enclose any commands within the double braces when underlining multiple lines of text.

Headers, footes, and apportioned text can be underlined by portion:

/{{**text**}}/**text**/

or completely — the entire line:

/{{**text/text/text**}}/

For the second case, the outer delimiters must enclose the double braces.

For example:

```
.HEAD/{{Monthly Report/UNIVERSAL MFG CORP/Page#}}/
```

processes to:

Monthly Report   UNIVERSAL MFG CORP   Page   1

You can underline blanks in any of the following ways:

1. In any mode, use the blank character within the braces, e.g.,

FDR 3104   6–11   *June 1978*

.blank #

{{# # # # #}}

2. In NFILL mode, use regular or reserved blanks, e.g.,

{{   }}   or   {{# # # # #}}

3. In either mode, use the underscore character (__).

If you want to enter a pair of left or right braces literally into your output file, i.e., {{ or }}, type a phantom hyphen between the two braces in your source file.

For example, the source file:

```
.hyphen &
{&{I'm enclosed in literal double braces}&}
```

processes to:

```
{{I'm enclosed in literal double braces}}
```

The default for the phantom hyphen is the rubout key.

NOTE

1. If your terminal *cannot* format underlined text properly because it cannot backspace, a separate line containing the underline characters will be printed below the text to be underlined.

2. For Diablo printers, the codes control-left-parenthesis and control-right-parenthesis generate the left and right braces, respectively.

3. If you have underlined a decimal heading, the decimal label number will also be underlined, unless an extra space is between the decimal heading command and the text of the heading.

INDEXING

RUNOFF has two commands which permit you to build a file of index notes: the IX-FILE command, which defines the name of the file, and the INDEX command, which indicates items in your source file to be noted. The format of the IXFILE command is:

.IXFILE filename

This defines the name of the index file. The format of the INDEX command is:

.INDEX string

If you have given the IXFILE command, RUNOFF compiles a list of all **strings** given in the INDEX commands, together with the page numbers of the output page that RUNOFF was processing when each INDEX command was given. Note that this is not a proper index – you still have to alphabetize this list, and condense multiple entries to a single occurrence of string plus all its page references.

If you have given at least one INDEX command in your source file, but not specified IXFILE, RUNOFF asks for the name of the index file with the prompt:

INDEX FILE

You may give a filename, or else hit the ⌐RETURN⌐ . The latter tells RUNOFF that you do not want an index file this time; RUNOFF will ignore all further INDEX commands in the file. (You can specify no index file via the NIXFILE command.) See the RUNOFF REFERENCE SECTION for more information on these commands.

Pausing and/or perforating

RUNOFF has two commands which have an effect between pages of output. The format of these commands is:

.PAUSE
.PERFORATE [n]

The PAUSE command tells RUNOFF to pause at the end of each page of the output file until you type a character at the terminal. This permits you to adjust the paper in a hard-copy terminal, which is particularly useful if you want to run a file off on separate sheets of paper, instead of on the continuous sheets which many terminals use. Typing any character signals RUNOFF to process another page; we recommend a non-printing character (such as ⌐RETURN⌐), which does not leave a mark on the paper.

The PERFORATE command causes RUNOFF to print a line of hyphens, which are meant to act as perforation marks, at the place where the physical bottom of the paper should be. This helps to indicate where new pages begin on long rolls of paper output. If you give any value for **n** that is a positive integer — e.g., 1, 2, or 378 — RUNOFF will print one hyphen at each end of that line, instead of a full line of hyphens. The full line is the default.

The PAUSE and PERFORATE modes can be turned off with the commands:

.NPAUSE
.NPERFORATE

respectively.

Suppressing RUNOFF command error messages

As a rule, you will want to know where in your source file you have made RUNOFF command errors. There will be times when you want RUNOFF to process your source file without stopping to display error messages. The command:

.ERRGO

tells RUNOFF to suppress all error messages as well as the associated actions, and instead process the entire file. The output file will include the results of any errors that occur.

This method of using RUNOFF is primarily useful when you are processing a file as a phantom user — i.e., you have told the computer to RUNOFF a file during a period of time when you are not going to be at the terminal, and therefore cannot correct and continue processing as errors occur.

The command:

     .NERRGO

returns RUNOFF to its default mode of displaying error messages and then returning to command mode.

WHAT IS DECIMALIZATION?

If you have ever written an outline for a paper, you will recall that this outline divided the contents of the paper into sections and subsections, and that each piece could be uniquely identified. For example, in the outline:

BREEDING FROGS FOR PLEASURE AND PROFIT

   I. INTRODUCTION
      A. History
         1. Ancient
         2. Recent
         3. Rain of Frogs – Myth or Reality?
      B. Why frogs
  II. HOW TO RAISE FROGS
      A. Location
      B. Housing
      C. Schedule
      D. Weather

The information on Recent History is contained in the second division of Part A in Section I.

Decimalization is another way to divide and label the various portions of your documents. In decimalization, you use numbers — i.e., 1, 2, 5, 32, etc. — to mark successive sections, and the numbers indicating different levels of organization are separated by decimal points (hence the name "decimalization").

Using decimalization, the above outline would become:

1. INTRODUCTION
1.1 History
1.1.1 Ancient
1.1.2 Recent
1.1.3 Rain of Frogs – Myth or Reality?
1.2 Why Frogs

2. HOW TO RAISE FROGS
2.1 Location
2.2 Housing
2.3 Schedule
2.4 Weather

Each paragraph and heading is uniquely identified by a number and decimal point combination called a decimal label.

1	is a label for a first-level block of text
2.1	is a label for a second-level block of text
3.6.2	is a label for a third-level block of text
4.7.6.2	is a label for a fourth-level block of text

The major function of decimal labels is to permit you to identify any portion of a document without reference to a page number. This is particularly useful when you want to insert cross-references in a document for which the final page numbers are not known.

Many people like to use decimal labels in their documents because it clarifies the relationships between various sections. This method is common in military documentation and engineering specifications.

Figure 7-1 shows a sample page of decimalized text. Notice that you can put decimal labels both on headings and paragraphs.

USING RUNOFF TO DO DECIMALIZATION

If you were typing your document "by hand" instead of using RUNOFF, and wanted to decimalize it (i.e., insert decimal headings), you would have to keep track of all your various level numbers, plus the values, and possibly the appropriate indentions as well. With RUNOFF's decimalization features, all you have to do is enter RUNOFF commands where you want a decimal label. RUNOFF automatically inserts the proper values.

RUNOFF also enables you to make changes in your decimalized document without any additional work. You can add or delete paragraphs, move sections around, combine chapters, rearrange the entire source file if you wish — so long as you've got the right commands in the new source file, RUNOFF does the new decimal labels correctly. In addition, once you have inserted the decimalization commands into your source file, you can have RUNOFF generate a paginated table of contents based on the decimal labels.

You can also use RUNOFF's decimalization commands to format your document, but suppress printing of actual numbers.

FROG RAISING FOR PLEASURE AND PROFIT

1 HISTORY OF THE FROG INDUSTRY

1.1 What Has Held Frog Raising Back?

Frog farming is perhaps America's most needed, yet least developed industry. What has prevented its normal progress? Why aren't frog farms as common as poultry farms? (All indications show that frogs are just as popular as food as chickens.) The frog industry has been a "victim of circumstances," and here are the things that have held it back:

1.1.1 SKEPTICISM:

It is difficult for people to accept any new thing. The possibilities in frog raising amazed many persons, and they thought "it just couldn't be." These people required a leader to go first—to "show them" what could be done—then they willingly follow.

1.1.2 JEALOUSY:

Frog raisers who were successful guarded their secrets. They seemed to want to be the only one raising frogs in their section. Today most frog raisers know that cooperations with their fellow frogs raisers brings NEW knowledge to themselves too.

1.1.3 LACK OF PROPER INSTRUCTIONS:

Until recently there was no way to learn how to raise frogs. You just had to "experiment" for yourself without help, assistance, or even interest from anyone else. After you finish this book, consider whether or not you ever could be successful without the knowledge contained herein.

1.1.4 FEEDING PROBLEMS:

Some persons put frogs in ponds and expect them to "make their own food." They knew nothing about what they ate, nor how to raise the frogs. How could they expect to succeed?

1.1.5 CANNIBALISM:

Amateurs who simply put a few pairs of breeders in a pond never seemed to get any young. Their frogs ate them as fast as they hatched. Today cannibalism is no longer a problem. You will learn in future lessons how simple it is to avoid it.

Figure 7-1. Decimalized Text

Specifying decimal labels is a two-step process:

1. Define the format parameters at beginning of source file — numbers of lines to skip and amount of indentation for each level of decimal headings.

2. Insert commands throughout the text, wherever you want decimal labels.

You can insert decimalization commands either as you create the source file, or edit them into an existing source file. When you give the RUNOFF command, RUNOFF creates, increments and formats the decimal labels and accompanying text the way you have indicated.

FORMAT PARAMETERS

Format parameters are the values you can set for each level of decimal heading, controlling the indentions and line-skipping. Figure 7-2 illustrates the four parameters you can set, which are:

- **before-skip** – the number of lines to skip before the label is printed

- **after-skip** – the number of lines to skip after the label is printed

- **head-indent** – the number of spaces to indent the label from the current left margin

- **text-indent** – the number of spaces to indent the text which follows the label, from the current left margin (including head-indent)

---

head-indent   2.7 MAINTENANCE INSTRUCTIONS FOR POWER CELL

                    With proper care, the power cell on the light saber
                    (LSB/6832) shall last for 325.6 years. This section
       text-indent includes instructions applicable to the power cell.

       before-skip

              2.7.1 Testing Level of Charge

                after-skip

                    Pressing Area B (See Fig. VI-4A) on the
                    handle activates the charge check. Read
                    charge level by comparing green charge dot to
                    ground and minimum marks. (Charge level for
                    activated light saber may be obtained by
                    monitoring audio hum frequency.)

Figure 7-2. The four decimalization parameters

You can set these values individually for each level, or generally for all levels of decimal labels. If you do not explicitly set any of the four parameters, RUNOFF uses the following defaults:

- Skip two lines before and one line after each label.

- Indent labels three additional spaces; indent text zero spaces more.

Figure 7-3 illustrates a page of text with default decimalization settings.

FROG RAISING FOR PLEASURE AND PROFIT

1 HISTORY OF THE FROG INDUSTRY

1.1 What Has Held Frog Raising Back?

Frog farming is perhaps America's most needed, yet least developed
industry.   What has prevented its normal progress? Why aren't frog
farms as common as poultry farms? (All indications show that frogs
are just as popular as food as chickens.) The frog industry has
been a "victim of circumstances," and here are the things that have
held it back:

1.1.1 SKEPTICISM AND JEALOUSY

1.1.1.1 Skepticism:

It is difficult for people to accept any new thing. The
possibilities in frog raising amazed many persons, and they
thought "it just couldn't be." These people required a leader
to go first—to "show them" what could be done—then they
willingly follow.

1.1.1.2 Jealousy:

Frog raisers who were successful guarded their secrets. They
seemed to want to be the only one raising frogs in their
section.   Today most frog raisers know that cooperations with
their fellow frogs raisers brings NEW knowledge to themselves
too.

1.1.2 LACK OF PROPER INSTRUCTIONS:

Until recently there was no way to learn how to raise frogs. You
just had to "experiment" for yourself without help, assistance,
or even interest from anyone else. After you finish this book,
consider whether or not you ever could be successful without the
knowledge contained herein.

1.1.3 FEEDING PROBLEMS:

Some persons put frogs in ponds and expect them to "make their
own food."   They knew nothing about what they ate, nor how to
raise the frogs. How could they expect to succeed?

Figure 7-3. Default decimalizaton settings

FORMAT COMMANDS

The two commands which set values for the format parameters are DSKIP and DINDENT.

The DSKIP command specifies the number of lines to be skipped before and after each label of **level-number.** The format of the DSKIP command is:

.DSKIP $\left\{ \begin{array}{c} \textbf{level-number} \\ * \end{array} \right\}$ **before-skip [after-skip]**

If the value of **level-number** is an asterisk (*), RUNOFF assumes that the command applies to labels of the next lower level. If the value of level-number is zero, the DSKIP command sets the skip values for all levels of labels except those that are explicitly defined following such a command in your text. For example, if you expect to have four levels of labels in your text, and want to skip five lines before each label and two lines after, then the following DSKIP sequences are equivalent:

$$\left\{ \begin{array}{ccc} \textbf{.dskip} & \textbf{1} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{2} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{3} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{4} & \textbf{5} & \textbf{2} \end{array} \right\} = \left\{ \begin{array}{ccc} \textbf{.dskip} & \textbf{1} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{*} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{*} & \textbf{5} & \textbf{2} \\ \textbf{.dskip} & \textbf{*} & \textbf{5} & \textbf{2} \end{array} \right\} = \quad \textbf{.dskip 0 5 2}$$

Unless the value of **after-skip** is zero, RUNOFF does a BREAK after printing the decimal label. If the value of after-skip is zero, RUNOFF does not do this BREAK; text follows the label on the same line. This permits you to begin paragraphs with a decimal label. If the value of **before-skip** or after-skip is –1, RUNOFF EJECTS to a new page.

The DINDENT command specifies the amount to indent labels and text from the current left margin. The format of the DINDENT command is:

.DINDENT level-number label-indent [text-indent]

If the value of **level-number** is an asterisk (*), RUNOFF assumes that the command applies to labels of the next lower level than the current one. For example, if you are currently on level 2, the command DINDENT * 5 8 applies to level 3.

If the value of level-number is zero, the DINDENT command sets the indent values for all level of labels in your text except level 1, for which **label-indent** is always 0 unless specifically overridden by a level 1 DINDENT command.

```
.dindent 1 0 0
.dindent 2 5 0
.dindent 3 8 2
```

would produce, given an initial left margin of 10:

- Level 1 labels indented 10+0=10 spaces,
  following text indented 10+0=10 spaces.

- Level 2 labels indented 10+5=15 spaces,
  following text indented 15+0=15 spaces.
- Level 3 labels indented 15+8=23 spaces,
  following text indented 23+2=25 spaces.

Unless you have explicitly set the indentions for level 1 (e.g., DINDENT 1 5 3), RUNOFF will not indent level 1 labels or text. The command DINDENT 0 5 2, for example, will cause decimalization indenting to begin with level 2.

WAYS TO GENERATE DECIMAL LABELS

You can generate a decimal label:

- On the *same* level as the previous label.
- One or more levels *up* from the level of previous label
- One level *down* from the level of the previous label

On its line, each label can be followed by a one-line heading, the text, or nothing. You can also generate a heading or block of text, without the decimal label, on the same or lower level as the previous label.

The commands which generate decimal labels are DNEXT and DDOWN.

The commands which generate headings or text with the specified spacing and indention but without decimal labels are DDSUPPRESS and DNSUPPRESS.

You can explicitly set both the current level-number and the value of the current decimal label via the DUP, DLEVEL and DRESET commands.

The heading in all the commands is optional. If the value of after-skip is zero at the current level, the source text will continue on the same line; otherwise, it will cause a BREAK.

The DNEXT command generates a new label on the current level. The format of the DNEXT command is:

.**DNEXT** [**heading**]

For example, if the most recent label and **heading** was 2.2 Life Cycle, the command .dnext Mating Season produces the label and heading:

2.3 Mating Season

The DDOWN command generates a new label one level below the current level. The format of the DDOWN command is:

.**DDOWN** [**heading**]

For example, if the most recent label and **heading** was 2.3 MATING SEASON, the command .ddown Tropical Climates produces the label and heading:

2.3.1 Tropical Climates

LEVEL-RESETTING COMMANDS

The DUP and DLEVEL commands reset the level of the labels you are generating. Their major use is to go "up" — return to a higher level from a lower (for example, to return to level 2 from level 5). The DUP command is a "relative" command — it moves you a given number of levels from where you are. The DLEVEL command is an "absolute" command — you specify what level you want, without regard to the level you are on at the time.

In other words, if your most recent label is level 5, and you want to return to level 2, you can get there either by saying "go up 3 levels" or "go to Level 2."

The DUP command resets the level value "up" by a specified number of levels; the next DNEXT command you give will generate a label at the new level. The format of the DUP command is:

.DUP [n-levels]

If **n-levels** is zero or omitted, the default value of one is used; the current level is reset up 1 level (i.e., from 2 to 1, from 3 to 2, from 4 to 3, etc.)

If n-levels is greater than the current level, the level is reset to level 1 (the top level).

This command does an implicit BREAK and resets the indent and skip values to those of the new level.

```
.dnext      A LEVEL THREE LABEL
text...
.dup
.dnext      a level two label
```

The DLEVEL command resets the current level to a new level. Text following a DLEVEL command will be indented to the setting for the new level. A DNEXT command following a DLEVEL command will, of course, generate a label on the new level. The DLEVEL command also does an implicit BREAK. The format of the DLEVEL command is:

.DLEVEL [level]

If the value of **level** is zero or omitted, the default value of one is used — the level value is reset to one (first level).

The DRESET command allows you to specify the next number value for a specific level in the next label. This command is almost always used for the current level. For example, if the most recent label is 2, you can specify that you want the next Level One label to be 6. This is particularly useful if you must leave sections out of a document — or if you want to begin a document, say, on Chapter 6. The format of the DRESET command is:

.DRESET [level] [value]

The DRESET command resets a specified **level** to a new **value**. (Actually, the number at that level is set to one less than value.) Since the DNEXT command generates a label

whose value on the current level is one more than the current value, the next value on the current level will be:

value – 1 + 1 = value

However, this means that if the current level is not the same as level, you will get an incorrect label when you give the DNEXT command.

Example:

      Your most recent label was 1.2.7
      Your current level is 3
      You want the next label to be 5 (level 1, value 5)

This is what you *don't* want to do:

      .dreset 1 5
      .dnext

Doing that would give you the label 4.2.8. You want to do it one of these ways:

.dreset 1 5		**.dreset 1 5**		**.dup 2**		**.dlevel 1**
.dup 2	*or*	**.dlevel 1**	*or*	**.dlevel 1 5**	*or*	**.dreset 1 5**
.dnext		**.dnext**		**.dnext**		**.dnext**

If omitted when the DRESET command is given, the default value for level and value is one.

GENERATING UNLABELED HEADINGS AND PARAGRAPHS

The DNSUPPRESS and DDSUPPRESS commands have the same indenting and skipping effects as DNEXT and DDOWN; they do not generate decimal labels, but do alter the current count. If no heading is specified, the text follows after the appropriate number of skipped lines.

The DNSUPPRESS command generates an unlabeled **heading** on the current level. The format of the DNSUPPRESS command is:

      .DNSUPPRESS [heading]

The DDSUPPRESS command generates an unlabeled **heading** one level down from the previous level. The format of the DDSUPPRESS command is:

      .DDSUPPRESS [heading]

For example,

      .dnsuppress PORPOISE-HUMAN VOCODERS
      .ddsuppress Basic Vocabulary and Expressions

processes to

PORPOISE-HUMAN VOCODERS
Basic Vocabulary and Expressions

MISCELLANEOUS INFORMATION YOU SHOULD KNOW

Decimal Range

RUNOFF produces labels with up to 8 levels. The largest value you can have on any one level is 99. The largest decimal value, therefore, is 99.99.99.99.99.99.99.99 (most documents seldom require more than four levels).

Space between two labels not separated by text

If there is no text between two labels (same or different levels), RUNOFF will still skip the number of printing lines specified by after-skip following the first label, and then the number of lines specified by before-skip preceding the second label.

Widow prevention

RUNOFF will *not* print a label at the bottom of a page unless there is room for at least one line of text below it.

Keeping track of indenting

When using decimalization, the command UNDENT or UNDENT 0 will undent to the left margin setting for the current level *instead* of the page margin defined by SMARGIN.

Be sure to UNDENT any additional indentations set by INDENT before changing levels or starting new paragraphs.·

Underlining labels and headings

If you are underlining the heading after a decimal label:

- One space between the DNEXT/DOWN command and the left brackets will underline *both* label *and* heading; e.g.:

      .dn {{LABEL AND HEADING UNDERLINED}}

      4.2.1 LABEL AND HEADING UNDERLINED

- Two spaces between the DNEXT/DDOWN command and the left brackets will underline *only* the heading; e.g.:

      .dn {{HEADING ONLY}}

  becomes

      4.2.1 HEADING ONLY

GENERATING A TABLE OF CONTENTS

If your source file contains decimalization commands, RUNOFF can automatically generate a table of contents from the labels and put it into a specified file. You generate a table of contents file by including the TOFC command in your source file.

With this command, RUNOFF produces a table of contents entry each time it encounters a DNEXT, DDOWN, DNSUPPRESS, or DDSUPPRESS and/or TTOFC command in the source file. The table of contents entry consists of the decimal heading number, the value for heading (if any), periods out to the right margin, and the page number on which the heading appears. If the heading is too long to fit on one line, it will be split at a space and continued, indented, on the following line.

Example:

The format of the TOFC command is:

.TOFC   filename [**level**]

If a value is given for **level**, RUNOFF includes all labels through this level. For example, if you give a value of 3, your table of contents lists all labels (and headings) of levels 1, 2, and 3.

If **filename** already exists, you will get the message:

OK TO DELETE OLD filename?

Any response except YES, YE, Y, OK or O will produce the message:

NEW NAME:

Format of the table of contents file

When you give the TOFC command, RUNOFF creates a file containing the following RUNOFF commands:

.TMARGIN currnt-value	Specify top margin
.BMARGIN current-value	Specify bottom margin
.SMARGIN current-value	Specify side margins
.WIDTH current-value	Specify paper width
.LENGTH current-value	Specify paper length

.BLANK current-value	Specify Special Blank Character
.NFILL	Enter NFILL mode
.COLUMN 1	Single Column
.DSKIP 0 0 0	Skip no lines before or after entry, except,
.DSKIP 1 1 0	Skip one line before level 1 entries.
.DINDENT 0 0 3	Indent 3 after all levels, except, level 1
.DINDENT 1 0 2	Indent 2 after level 1 entry
.DINDENT 2 0 4	Indent 4 after level 2 entry
.DINDENT 3 0 6	Indent 6 after level 3 entry
.DLEVEL 1	Go to decimal heading level 1
.DRESET 1	Reset to heading value 1
.EJECT	Eject to new page
.SKIP 2	Skip 2 lines
.> {{Table of Contents}}	
.SKIP 2	Skip 2 lines

The settings for the various **current-values** are the values most recently defined in the source file prior to the TOFC command, or, for those commands not given, the appropriate default values. This guarantees that your table of contents will look the same as the rest of the output.

In other words, if your source file contains the sequence:

```
.tmargin 10
.bmargin 2
.smargin 15
.tofc contents
```

Your table-of-contents source file would have these margin settings and the default page size; the file contents would have this sequence:

```
.tmargin 10
.bmargin 2
.smargin 15
.width 85      (Default value)
.length 66     (Default value)
    .
    .
    .
```

The TOFC command *must* follow the page-formatting commands in your source file in order for RUNOFF to pass the values on.

The DSKIP and DINDENT commands in filename determine the SKIP and INDENT values for the various levels in the table of contents. You can, of course, change any and all of these values before processing the TOFC file with RUNOFF.

The DLEVEL and DRESET commands re-initialize the current level to first-level, and the current value to one. This is done so you can INSERT the table of contents source file into your main source file, for processing at the same time. Without these commands, the table of contents file would be processed with the most recent left margin and level values.

The EJECT command makes sure the table of contents starts on a new page; you then get the underlined title, Table of Contents, two blank printing lines, and then the table of contents itself.

Remember: If the TOFC command is *not* in the main source file, you can redefine any and all of these settings by changing/inserting RUNOFF commands at the top of the file with EDITOR—and a finished table of contents source file can be INSERTed at any point in your main source file.

The TTOFC command

The TTOFC (To Table of Contents) command lets you enter text strings which are not headings into the table of contents. The format of the TTOFC command is:

>   .TTOFC string

The value of **string** is inserted at the appropriate place in the table of contents file. String is indented to the current level setting; underscored characters are still underscored.

The DLIMIT command

The DLIMIT command lets you reset the level of decimal labels (and pertinent headings) to be entered into the table of contents file. The format of the DLIMIT command is:

>   .DLIMIT [limit]

limit defines the highest level of label to be entered into the table of contents file; for example, if the value of limit is 3, then only level one, two and three labels will be entered. You can change the value of limit as often as you like.

Turning off contents generation

To omit certain decimal labels and associated headings from the table of contents file, use the two commands:

>   .TOFC 0
>   .TOFC 1

When RUNOFF encounters TOFC 0 (off), it will stop processing all following decimal labels and headings to the TOFC file until it encounters TOFC 1 (on). RUNOFF will

ignore the TOFC 0 and TOFC 1 commands if you are not currently making a table of contents.

RUNOFF DECIMALIZATION COMMAND SUMMARY

A summary of all RUNOFF decimalization commands follows:

.DDOWN [heading]	Go down one level and generate a label.
.DDSUPPRESS [heading]	Go down one level but do not print a label.
.DINDENT level label-indent text-indent	Set indent increment for **level** to **label-indent** and **text-indent** spaces.
.DLEVEL level	Define the **level** at which numbering continues.
.DNEXT [heading]	Generate a label (and **heading**) on the current decimal level.
.DNSUPPRESS [heading]	Generate a **heading** on the current decimal level, but do not generate a label.
.DREST level value	Change the current level number to **level** and set the next number on this level to **value.**
.DSKIP level before-skip after-skip	For each label on specified **level,** skip **before-skip** lines before the label and **after-skip** lines after it.
.DUP [n-levels]	Decrement the level number by **n-levels**.
.TOFC filename [limit]	Generate a table of contents in **filename**. If the **limit** is specified, only labels down to that level are recorded in the contents.
.TOFC	Close the current table of contents.
.TOFC 0	Temporarily turn off the generation of the table of contents and at .TOFC 1, turn it on again.
.DLIMIT [limit]	Reset maximum level of label to be entered in TOFC file.
.TTOFC string	Enter string in TOFC file.

INTRODUCTION

This section explains and demonstrates the more advanced features of EDITOR and RUNOFF. The features covered are:

- Document formatting
- How to do hanging indents
- Form letters
- Using GMODIFY

Your suggestions and contributions to future revisions of this section are welcome.

DOCUMENT FORMATTING

The following is a suggested method for formatting multi-section documents.

Due to the six-character limit on symbol-names, the following abbreviations or conventions have been used:

- **SUBJCT** — for the name of the document
- **DOCNO** — for the number of the document
- **VERSNO** — for the number of the version or revision
- **PUBDAT** — for the date of publication
- **TITLE** — for the section title
- **SECNO** — for the number of the section

The file GLOBAL.INFO contains the information pertinent to all sections of a given document. Use EDITOR or CHANGE the letters W, X, Y and Z to the appropriate values. (Or input the entire file.) You can also define one or more groups of standard tab stops here.

Here is one possible format for GLOBAL.INFO:

```
.*GLOBAL.INFO--contains general formatting information
.define SUBJECT W
.define DOCNO X
.define VERSNO Y
.define PUBDAT Z
.define FIVES 5 10 15 20 25 30 35
.define TENS 10 20 30 40 50
.define TABLE1 5 15 18 33 56
```

GLOBAL.INFO gets inserted into the file SECTION.INFO. The file SECTION.INFO sets up headers, footers, and output filename. This file gets inserted at the beginning of each section. You could type it in each time, but it is quicker and more practical to IN-SERT it.

```
.*SECTION.INFO--sets up section format
.insert GLOBAL.INFO
.eheader/SECTION %SECNO%/%SUBJCT%//
.efooter/VERSION %VERSNO%/Page %SECNO% - #//
.oheader//DOC %DOCNO%/%TITLE%/
.ofooter//Page %SECNO% - #/%PUBDAT%/
.file $SECTION.%SECNO%
.sk2
.>SECTION %SECNO%
.sk
.>%TITLE%
```

Then, start each section like this:

```
.*This file contains SECTIONxx
.define SECNO x
.define TITLE text of title
.insert SECTION.INFO
.skip 10
.tab @ %FIVES%
text
```

Then, if you have set up GLOBAL.INFO properly, running off a section gives you pages formatted as shown in Figure 8-1. Figure 8-2 shows the file arrangement for this method of document formatting.

HOW TO DO HANGING INDENTS

RUNOFF permits you to specify either positive or negative indention at the beginning of paragraphs. Positive indention is the more traditional — the first word of text is in-

dented **m** spaces to the right of the current left margin , like so:

```
        You might ask, "Why do you recommend not
less than  five   pairs   of   breeders    to
beginners?   We urge you to get as many pairs
as possible to  start  out  with  because  it
gives you a better chance to succeed.
```

Negative, or hanging indention, means the first word of text begins m spaces to the left of the current left margin — specified as **−m** spaces. For example:

```
You might ask, "Why  do  you  recommend  not
        less  than  five  pairs  of  breeders to
        beginners?  We urge you to get  as  many
        pairs  as  possible  to  start  out with
        because it gives you a better chance  to
        succeed.
```

Hanging indents (a common nickname for *negative* indention) are often used for formatting lists. To start a new block of text without beginning a new paragraph, use the SKIP command. When preparing to do hanging indents, be sure to:

1. INDENT the left margin appropriately, keeping in mind that each paragraph will begin m spaces to the left of this new margin. (Also RINDENT the right margin, if desired.)
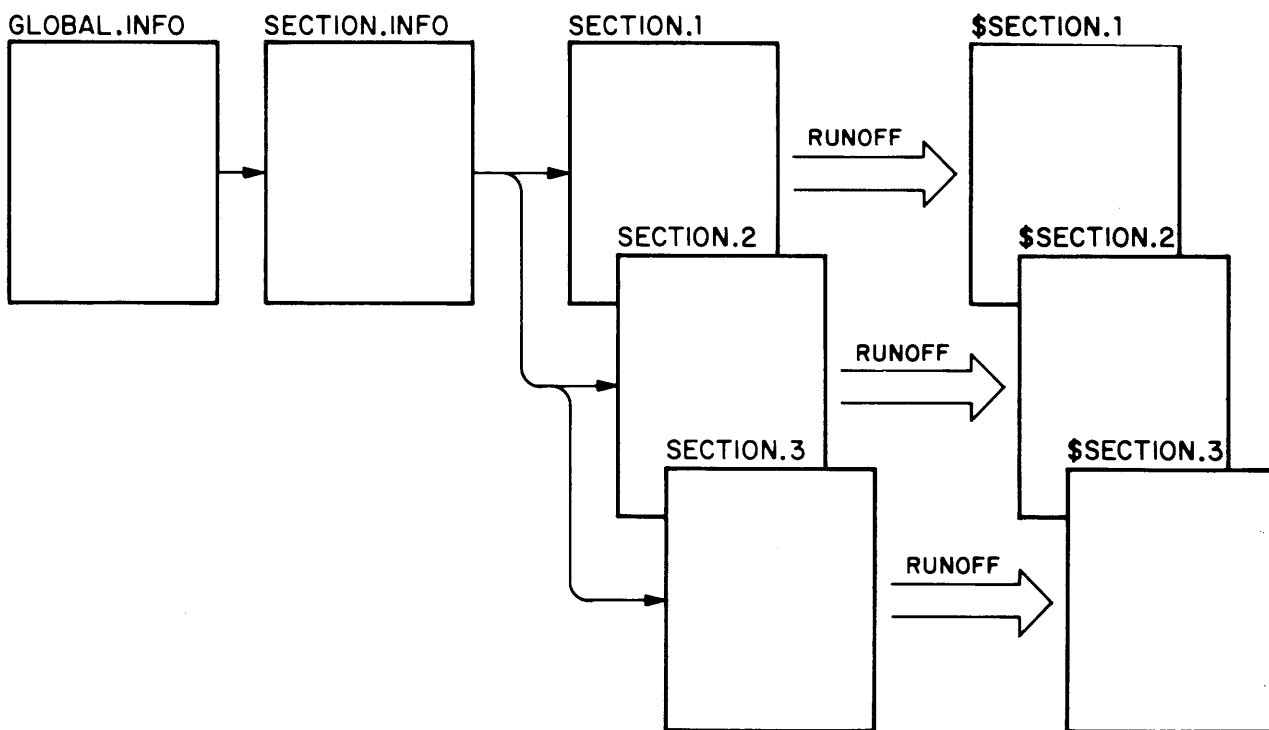
Figure 8-1. File arrangement for document formatting

2. Set the negative indention with the PARAGRAPH command.

3. Reset the left margin (and right, if changed) after the last entry in the list, using UNDENT.

4. Reset the PARAGRAPH values (if resetting to the default values, you can give the NPARAGRAPH command, otherwise you must give the reset values in the next PARAGRAPH command).

Also consider using the BLANK character between the item number, if any, and the first word. The BLANK character prevents RUNOFF from inserting justification spaces between such numbers and the text which would give the appearance of an uneven text margin. (The default for BLANK is control-@.) For example, the source file:

```
.blank
.p 5 1
The benefits of having good water vegetation
by your frog pond are:
.in 8
.p -3 1
1.It gives oxygen to water.  Without enough
oxygen, only a small number
of frogs could be raised in a small pond.
.p
2.Keeps the water fresh.  Water in which proper
plants grow remains fresh regardless of age. It
smells sweet and clear.
.p
3.Helps to keep the water cool in summer.
.p
4.Provides cover for frogs and water life
while in the water.  Many larval
insects thrive in the properly planted pond.
.un
.p 5 1
Bank vegetation also serves many functions
in the frog pond...
```

processes to:

```
    The benefits of having good water  vegetation
by your frog pond are:


    1. It gives oxygen to water.  Without  enough
       oxygen, only a small number of frogs could
       be raised in a small pond.


    2. Keeps the water  fresh.   Water  in  which
       proper  plants   grow   remains    fresh
       regardless of age.  It  smells  sweet  and
       clear.
```

3. Helps to keep the water cool in summer.

4. Provides cover for frogs and water life
   while in the water. Many larval insects
   thrive in the properly planted pond.

    Bank vegetation also serves many functions in
the frog pond...

Negative indention can also be used to format two-column tables when the second column contains justifiable text.

For example,

```
.*This file contains a negative indent table
.blank #
Following are listed some of the controllable conditions and their
effect on the tadpoles:
.sk
.indent 23
.p -18 0
DEEP#WATER########Tadpoles living in deep water
ponds, where they cannot come into contact
with a sloping bank, will usually remain in the
tadpole stage twice as long.
.sk
.p
SUDDEN#WATER######Cold water retards the
.p
TEMPERATURE#######development of and growth
.p
CHANGES###########of tadpoles, just as it does with frogs.
.sk
.p
UNBALANCED#DIET###Contrary to other forms of life,
overfeeding causes tadpoles to grow fast, but
keeps them in the tadpole stage for a much longer
period.
```

processes to:

Following are listed some of the controllable
conditions and their effect on the tadpoles:

    DEEP WATER   Tadpoles living in deep
                              water ponds, where they
                              cannot come into contact
                              with a sloping bank, will
                              usually remain in the
                              tadpole stage twice as
                              long.

```
SUDDEN WATER       Cold water retards the
TEMPERATURE        development of and growth
CHANGES            of tadpoles,  just  as   it
                   does with frogs.


UNBALANCED DIET    Contrary to other forms  of
                   life,   overfeeding  causes
                   tadpoles to grow fast,  but
                   keeps  them  in the tadpole
                   stage for  a  much   longer
                   period.
```

To simulate justification of initial lines in each paragraph, use EDITOR to insert blank characters as needed. (If no paragraph has more than one line in the first column, you can use the command PARAGRAPH –xx 1 and will not have to watch for this.)

INSERTING ADDRESSES ON FORM LETTERS

Suppose you want to RUNOFF a group of form letters, each with a different name and address, such as this:

```
                   The Raniburger Corporation
                   666 Ouroboros Drive
                   Arkham, Mass. 02546
                   April 3, 1978

Arthur Trent
13 Diablo Drive
Wallingford, Pa. 19380

Dear Arthur Trent,

     It is our pleasure to inform you, Arthur
Trent, that you are a FINAL CONTESTANT in the
RANIBURGER BAKE-OFF!

     Raniburger is prepared to fly you, at your
expense, to our BIG BAKE-OFF, which will be
broadcast live from the Hotel Cthulhu in downtown
Arkham.  Winners will be selected by our panel of
distinguished judges, which includes Drs.  Ann E.
Lidda and Chuck Render of the Serpentine Health
Spas, and Grima Viper of Saruman Bakeries.  Prizes
include a two-week vacation for two at Loch Ness,
plus a year's supply of Raniburgers.


     So, congratulations, Arthur Trent, and we
hope to see you soon!

                   Sincerely,

                   The Raniburger Corporation
```

Rather than create and RUNOFF a new source file for each address, here's a way to do all these letters giving the RUNOFF command only once, assuming you have a uniform address file, such as:

Arthur Trent
13 Diablo Drive
Wallingford, Pa. 19380

Mildred Saurus
5507 Burney Blvd.
Phoenix, Arizona 85031

The method described in this section requires that all addresses have exactly the same number of lines. You can overcome this limitation by having one file for three-line addresses, a second for four-line addresses, etc., or by making all addresses the maximum number — making sure that you include enough blank lines to fill the shorter addresses to this length.

Suppose, for example, you have a file LIST containing three-line addresses, separated by blank lines, like the one above. You also have a source file LETTER, which has the symbol-names SYMB-1, SYMB-2, and SYMB-3 where you want the three lines of an address to appear (plus SYMB-1 wherever you want the name to appear in the text) such as:

```
.*This file contains a sample form letter
.*RANIBURGER Finalists Form Letter
.tab @ 20
.nfill
@The Raniburger Corporation
@666 Ouroboros Drive
@Arkham, Mass. 02546
@%DATE.2%
.sk
%SYMB-1%
%SYMB-2%
%SYMB-3%
.sk2
Dear %SYMB-1%,
.fill
.p 5 1
It is our pleasure to inform you, %SYMB-1%,
that you are a FINAL CONTESTANT in the RANIBURGER BAKE-OFF!
.p
Raniburger is prepared to fly you, at your expense,
to our BIG BAKE-OFF, which will be broadcast live from
the Hotel Cthulhu in downtown Arkham.
Winners will be selected by our panel of distinguished
judges, which includes
Drs. Ann E. Lidda and Chuck Render of the Serpentine
Health Spas, and Grima Viper of Saruman Bakeries.
Prizes include a two-week vacation for two
```

```
at Loch Ness, plus  a year's supply of Raniburgers.
.p
So, congratulations, %SYMB-1%, and we hope to see you soon!
.sk
@Sincerely,
.sk
@The Raniburger Corporation
.sk2
.>"I Never Ate A Frog I Didn't Like"
.eject
```

What you want to do is convert the address lines into the symbol-values assigned by SYMB-1, SYMB-2, and SYMB-3 by RUNOFF DEFINE commands. You also want to have the command INSERT LETTER follow each address. To make these changes in LIST, use EDITOR as follows:

1. Use the CHANGE//string/ format of the CHANGE command to insert the characters .def SYMB-n at the front of each address line.

2. Insert the line .insert LETTER after each address.

For example:

```
OK, ed list
GO
EDIT
next
Arthur Trent
c//.def SYMB-1 /
.def SYMB-1 Arthur Trent
n
13 Diablo Drive
c//.def SYMB-2 /
.def SYMB-2 13 Diablo Drive
n
Wallingford, Pa.19380
c//.def SYMB-3 /
.def SYMB-3 Wallingford, Pa.19380
i .insert LETTER
t,p6
.NULL.
.def SYMB-1 Arthur Trent
.def SYMB-2 13 Diablo Drive
.def SYMB-3 Wallingford, Pa.19380
.insert LETTER


n
Mildred Saurus
c//.def SYMB-1 /
.def SYMB-1 Mildred Saurus
        .
```

```
          .
          .
          .
    file main


    OK,
```

A quick and easy way to make these changes on the entire file LIST is to use EDITOR's
* ("Repeat") command. Typing this command line:

c//.def SYMB-1 /;n;c//.def SYMB-2 /;n;c//.def SYMB-3 /;i .ins LETTER;n2;*

will alter the entire LIST file properly. Simply position the pointer at the first line of
the first address, and type the above command line. You may want to give the BRIEF
command first, to eliminate unnecessary and possibly time-consuming verification
printouts. It is important that you do *not* omit the spaces between the symbol-names
and the closing delimiters (e.g., SYMB-1).

FILE the results of this editing with a new name (to avoid writing over the actual list).
The edited file, here called MAIN, should look like this:

```
    .def SYMB-1 Arthur Trent
    .def SYMB-2 13 Diablo Drive
    .def SYMB-3 Wallingford, Pa.19380
    .insert LETTER


    .def SYMB-1 Mildred Saurus
    .def SYMB-2 5507 Burney Blvd
    .def SYMB-3 Phoenix, Arizona 85031
    .insert LETTER


    .def SYMB-1 Genghis  Sylvia Khan
    .def SYMB-2 3 The Circle
    .def SYMB-3 Edison, NJ 08817
    .insert LETTER


    .def SYMB-1 Harry Seldon
    .def SYMB-2 2 Foundation Place
    .def SYMB-3 Trantnor, Ohio 45067
    .insert LETTER
```

Now, simply give the command:

RUNOFF MAIN

plus FILE $MAIN and/or TTY, and you will get all your form letters, one for each ad-
dress, with no more effort.

NOTE

If you want each letter on a separate sheet of paper, rather than on the standard continuous folded paper, give RUNOFF's PAUSE command at the beginning so you can insert paper for each letter.

Figure 8-2 shows what your output will look like. Figure 8-3 shows the arrangement of files for doing form letters.


```
           The Raniburger Corporation



                        May 24, 1978

Arthur Trent
13 Diablo Drive
Wallingford, Pa. 19380


Dear Arthur Trent,

        It is our pleasure to inform you, Arthur
Trent, that you are a FINAL CONTESTANT in the
RANIBURGER BAKE-OFF!

        Raniburger is prepared to fly you, at your
expense, to our BIG BAKE-OFF, which will be
broadcast live from the Hotel Cthulhu in downtown
Arkham.  Winners will be selected by our panel of
distinguished judges, which includes Drs.  Ann E.
Lidda and Chuck Render of the Serpentine Health
Spas, and Grima Viper of Saruman Bakeries.  Prizes
include a two-week vacation for two at Loch Ness,
plus a year's supply of Raniburgers.

        So, congratulations, Arthur Trent, and we
hope to see you soon!




                        Sincerely,

                        The Raniburger Corporation




                "I Never Ate A Frog I Didn't Like"
```

Figure 8-2. Form letter

Figure 8-3. Form letter file arrangement

HOW TO USE EDITOR'S GMODIFY COMMAND

EDITOR's GMODIFY command is a string-oriented editing routine which permits you to edit the current line on a character-by-character/column-by-column basis.

GMODIFY works by moving a column pointer back and forth across the current line, and copying specified characters into a temporary one-line text buffer Unlike EDITOR's line pointer, the column pointer is positioned BETWEEN columns. The column pointer always is initially located before the first column.

The temporary buffer is built up from left to right. You can also append, insert, overlay and retype characters into the buffer.

You use GMODIFY by following the command word with a series of sub-commands (explained below). Subcommands do the following tasks:

- Copy characters from the current line into the buffer by moving the column pointer either a specified number of columns, or until a specified character is found.

- Move the column pointer along the current line, in either direction, without copying characters into the buffer, either a specified number of columns or until a specified character is found.

- Insert new text strings into the buffer.

If the current line is a null line, EDITOR does not perform the GMODIFY command; instead, you get the message

.NULL.
BAD GM

If you have given an invalid or illegal subcommand, EDITOR will ignore the rest of the current command line, and the current line will not be replaced by the buffer contents.

Each subcommand, depending on the subcommand and the circumstances, consists of a single letter, and may be followed by:

- A number, indicating a number of columns.

- A character, indicating a character in the current line.

- A text string enclosed in delimiters, to be inserted into the buffer as specified.

Sub-commands may be separated by any number of spaces (or no spaces at all) but may not be separated by commas.

In the following explanation of GMODIFY's subcommands, the **c** represents a character in the current line, **n** stands for a number of columns/characters. The slash character (/) represents a delimiter — any character not contained in string may be used as the delimiter, except for an erase or kill character or a carriage-return. If the string in a sub-command is followed immediately by a carriage-return, you may omit the closing delimiter.

GMODIFY's subcommands are:

Subcommand	Meaning
A/string/	Append **string** as follows: Copy current line from current column pointer position to end of line, then append string. (Note: This sub-command is equivalent to the sub-commands F I/string/)
Bn	Back: Starting at the current column, move the column pointer back **n** spaces. (But not past the beginning of the line)
Cc	Copy: Starting at the current column, copy characters from the current line to the buffer, up to but not including the character **c**. The column pointer will remain in front of the column containing c.
Dc	Starting at the current column, move the column pointer to the right across the current line without copying characters to the buffer, up to but not including the character **c**.
En	Starting at the current column, move the column pointer **n** spaces to the right without copying characters to the buffer.

F   Finish: starting at the current column, copy characters from the current line into the buffer until the end of the current line is reached.

I/string/   Insert **string** at the end of the buffer

Mn   Move: starting at the current column, copy **n** characters from the current line to the buffer. The column pointer is left n characters to the right of where it began.

N   Negate: Reverse the sense of c in the next C or D sub-command. For example, the sub-command NC0 will copy characters into the buffer until a character which is not 0 is found. The N can be separated from the C or D by any number of other sub-commands; it will only affect the first C or D that follows it.

O/string/   Overlays **string** on top of a string of characters of equal length from the current line. The column pointer will be left at the last-copied character from the current line; ·if string is longer than the current line, GMODIFY assumes that it is overlaying on spaces.
As in the EDITOR OVERLAY command, a space in string leaves the original character unchanged, and a wild card symbol (!) in string provokes a space.

R/string/   Retype: Starting with the current column, **string** is copied to the buffer, and the column pointer is moved an equal number of columns across the current line. The R sub-command is similar to O except it does not recognize either the space or wild card symbol as special characters.

S   Start: reposition the column pointer to the beginning of the current line.

Here are a few examples using GMODIFY:

p	
OneTwo	*Start with this*
gmodify e3m3sm3	*Skip 3, Copy 3, Beginning Copy 3*
TwoOne	*You have flipped the two halves on the line*
g e3m3sm3	*Do it again*
OneTwo	*Line is restored to original order*
g dTfscT	*Same operation with different commands*
TwoOne	
g e3m3sm3	*Restore original line*
OneTwo	
g m3sm3sm3	*Copy beginning 3 times*
OneOneOne	
g a/Two	*Append "Two"*
OneOneOneTwo	

Using MOVE and the string buffers

The MOVE command permits you to move a text line from one string buffer to another, or to load a string to a specified buffer. The format of the MOVE command is:

$$\text{MOVE} \quad \text{buffer-1} \quad \left\{ \begin{array}{l} \textbf{buffer-2} \\ \textbf{/string/} \end{array} \right\}$$

The MOVE command moves one line of text to **buffer-1** from **buffer-2** or **string**. Any character which does not appear in string may be used as the delimiter, including comma and semicolon. If MOVE is the last command on the line, the closing delimiter may be omitted.

The names of the string buffers are:

- **INLIN** – the current line in the EDITOR file
- **EDLIN** – the line you are typing at your terminal which contains either commands or input.
- **STR.1** – string buffer (can also be called STRA)
- **STR.2** – string buffer (can also be called STRB)
- **STR.3** – string buffer (can also be called STRC)

Upper and lower case letters are equivalent in string buffer names.

NOTE
_____

EDITOR comes with the PRIME-supplied default of 3 STR.x string buffers. Your System Administrator may have configured additional ones.

_____

Using the XEQ command with the string buffers

Once you understand how to use the string buffers, you can move command strings into them and then execute them via EDITOR's XEQ command. This is particularly useful if you need to perform an operation many times in a given session, or periodically from day to day. Once you have a series of commands working, you don't even have to retype them — simply file for future use.

You put a command string into a string buffer via the MOVE command. You can also put it into a file at this point via UNLOAD. Remember: you *cannot* input explicit semicolons unless you first have changed the SEMICO character via SYMBOL SEMICO character. (You can use an alternate symbol and then CHANGE all occurrences.) If you have saved a command string in an external file, simply LOAD it in, and then MOVE it. Don't forget to back up one line between LOADing and MOVEing, and to DELETE the LOADed and MOVEd line from the actual fine.

Properly used, these techniques can be powerful timesavers. However, pay close attention when you do — accidentally starting on a null line, forgetting to delete the LOADed and MOVEd command string, or mis-naming a string buffer can prevent your mini-programs from working properly.

INTRODUCTION

This section contains complete information on all the EDITOR commands. The commands are listed in alphabetical order.

EXPLANATION OF THE COMMAND FORMAT

EDITOR's command format is:

> **COMMAND parameter**

The word in capital letters is the command word.

Command words

The letters shown in rust in the command words indicate the required abbreviation. You must type at least these letters; you may type as many more as you wish. The following are all acceptable ways to input a command whose format is **APPEND: A, APP, APPE, APPEND.**

Also, you may type the letters of command words in any combination of upper and lower case letters. All of the following are equally valid: **A, a, APpend, append, Ap-Pend, appEND.**

Parameters

As a rule, the parameter in a command format will be:

- The word **filename**, representing a filename.
- The letter **n**, representing a number.
- The word **character**, representing a single character.
- Any of the words **string, text,** or **newline,** representing a piece of text.

If the parameter is a number represented by the letter n, you don't have to type a space between the command word and the number. For example, the following are all valid:

> **Print5**
> **Pr5**
> **PRINT 15**
> **p–5**
> **p   –5**

Note that there cannot be a space between the minus sign and a number.

If the parameter is a filename or a character, you must have at least one space between the command word and the parameter:

> **load memo**
> **kill   &**

If the parameter is a text string (indicated by text, string, newline), EDITOR assumes that there is exactly one space between the command word (or abbreviation) and the text string:

> **find DEPARTMENT**
> **append and so forth**
> **insert Dear Sir or Madame,**

Any space after the first space is considered part of string. So, for example, the commands:

> **append⟦S⟧⟦S⟧⟦S⟧henceforth**
> **find⟦S⟧⟦S⟧⟦S⟧henceforth**

would be using the string "⟦S⟧⟦S⟧henceforth" which begins with two blank spaces.

THE COMMANDS

APPEND string

The APPEND command attaches the specified text to the end of the current line. One blank separates the command word APPEND (or whatever abbreviation) from **string** you wish to append. All further blanks are treated as text. If you want to have a space between the last word of the current line and the first word you are appending, you *must* type two spaces between the command word and the first word of appended text. If there is *no* blank between APPEND and string, EDITOR gives the error message:

> **BAD APPEND**

and you should try again. The string to be appended is terminated by either a ⟨RETURN⟩ or a semicolon (;). You can use commas (,) in the string. You can append semicolons—the semicolon character has been freed for use as a text character via the SYMBOL COMMAND. They must be inserted by using the CHANGE, MODIFY, or GMODIFY command. No trailing blanks — i.e., blanks between the last nonblank character of appended text and the ⟨RETURN⟩ or ; — are inserted.

BOTTOM

The BOTTOM command positions the pointer at the bottom of the EDITOR work file. A PRINT command at this point would show the null line .NULL. any new lines inserted now would go under the last line of text in the file. Any attempt to find the NEXT line would result in the word:

BOTTOM

BRIEF

The BRIEF command suppresses the verification output produced by the following commands: APPEND, CHANGE, FIND, GMODIFY, LOCATE, MODIFY, NEXT.

BRIEF mode allows experienced users to work faster. This is particularly useful on terminals which print at a slow speed.

When in BRIEF Mode, use the PRINT command to check the results of your most recent command: To re-activate the verification responses, use the command VERIFY (which is the default).

CHANGE/string-1/string-2/ [G] [n]

The CHANGE command replaces **string-1** with **string-2.**

The first character after the command word CHANGE (or abbreviation) is used as the delimiter in the command. Any character including the semicolon (;) and the space may be used as a delimiter instead of the slash.

If the letter **G** (for General) is specified, CHANGE changes *every* occurrence of string-1 on a line. If you *don't* specify G, only the *first* incidence of string-1 is changed.

If the value of **n** is 1, 0 or unspecified, EDITOR only makes changes on the current line. If a value other than 0 or 1 is specified, EDITOR inspects and makes changes on n lines starting at the current line, and leaves the pointer positioned at the nth line. If there are fewer than n lines in the file below the current line, the pointer is left at the null line below the last line, and the message BOTTOM is printed.

EDITOR will print all changed lines, plus the last line examined.

NOTE

1. Remember to go to the TOP before making changes on the file as a whole.

2. You can omit the closing delimiter (/) if you end the command with a ⌈ RETURN ⌉

3. You can specify the semicolon (;) as a text character within the delimiters — i.e., if you used "@" every place in your file where you wanted to use ";", then the command sequence

   TOP, CHANGE/@/;/g9999

   would change the @'s to ;'s. (Make sure n is greater than the number of lines in your file.)

4. You can use CHANGE to insert characters at the beginning of a line
   with the sequence CHANGE //string/.

DELETE [n]

The DELETE command deletes **n** lines, *including* the current line, from the EDITOR work file, and leaves the pointer at the place where the last deleted line was. (The line .NULL. will be maintained, in case you wish to insert a new line; this null line will disappear as soon as a new command moves the pointer away.)

If n is omitted, the default value of 1 is used, and *only* the current line is deleted. The value of n may be positive or negative, indicating deletion of the current line plus lines below or above the current line. Since n always includes the current line, D1 = D = D -1.

Helpful Hints: To avoid wiping out file lines completely, use the DUNLOAD command, and delete the DUNLOAD–created file(s) later. If the deleted lines were in the original file, you can QUIT without FILEing and start EDITing a new copy, if you have to. Of course, you will have to do all changes from the previous session on this new copy.

DELETE TO string

The command DELETE TO string deletes from the EDITOR work file all lines, starting with the current line, up to but *not* including the first following line which contains **string**. There *must* be a blank between TO and string. See the Helpful Hints in DELETE for an alternative.

WARNING

Be sure string is there and spelled *exactly* as specified or you will delete *everything* in your file below the current line.

DUNLOAD filename [n]

The DUNLOAD command creates a new file with the indicated **filename**, copies **n** lines from the EDITOR work file, beginning with the current line, into this new file, and then deletes these n lines from the work file. Be careful not to specify a filename currently in use unless you want the old file wiped out. If filename is not specified, you will get the error message:

BAD DUNLOA

If n is not specified, the default value of 1 is used and one line is DUNLOADed. DUNLOAD leaves the pointer positioned at a null line where the deleted lines used to be; this null line disappears as soon as the pointer is moved. The DUNLOAD command is useful for moving lines of text to different places; DUNLOAD can also be used instead of DELETE if you want to make sure you don't accidentally delete large blocks of text. (But don't forget to delete the DUNLOAD–created files from your UFD when you're through with them.)

DUNLOAD filename TO string

The DUNLOAD TO command copies lines from the EDITOR work file into a new file named **filename** and then deletes these lines from the original file. Lines are copied starting with the current line and continuing down until a line containing **string** is found, or until BOTTOM is reached. There must be a blank between TO and string. If filename is not specified, you will get a file named TO.

DUNLOAD TO leaves the pointer positioned at a null line where the deleted lines used to be; this null line disappears as soon as the pointer is moved.

The DUNLOAD TO command is useful for moving lines of text to different places; DUNLOAD TO can also be used instead of DELETE TO if you want to make sure you don't acidentally delete large chunks of text. (But don't forget to delete the DUNLOAD TO–created files from your UFD when you're through with them.)

Remember:

1. Be sure the string is there and spelled *exactly* as specified or you will *delete everything* in your file below the current line.

2. *Don't* specify a filename currently in use unless you want the old file wiped out.

ERASE character

The ERASE command allows you to change the previous value of the erase character to the **character** specified; this change will be in effect either until you give a new ERASE command within the EDITOR session or until you QUIT from EDITOR. The erase character rubs out the immediately preceding character of terminal input.

When you leave EDITOR, PRIMOS resets the erase character to the default. Unless your System Administrator has changed the system defaults, the EDITOR default erase character is the double–quote (") character.

The ERASE command is useful if you intend to use the default character often as a text character. The erase character of the moment can always be entered as text by preceding it with an up–arrow (↑).

To check the current value of the erase character use the PSYMBOL command.

FILE [filename]

The FILE command turns the EDITOR work file, (which is so far only temporary) into a permanent file in your UFD (or sub–UFD), and returns you from EDITOR back to PRIMOS.

WARNING

Since the work file has *no existence* outside of EDITOR, you must FILE if you want to save work.

The rules for using the FILE command are:

1. If you have been creating a new file, you must specify **filename.** (The error message FILENAME MUST BE SPECIFIED will occur if you don't.) You *cannot* have two files with the same name in the same UFD! If you give a filename which already exists in your UFD, EDITOR will *delete* the old file by that name from your current UFD and put the work file in its place.

2. The same warning holds true for old files. If you have been working on an old file and you specify the old filename, or say FILE without a filename, your old copy is deleted, and only your new version kept. Giving a new filename keeps both the old and new versions — but be sure not to accidentally wipe out some other old file by using its name.

3. If you do not wish to save your work from a given session — i.e., want to save your old version, if any — but don't want what you just did in the work file copy — type QUIT instead of FILE. If there is anything in your current file, EDITOR will inquire: FILE MODIFIED, OK TO QUIT? to double–check with you. A YES response QUITS you back to PRIMOS; NO provokes PLEASE FILE, at which point you give the FILE command.

4. Rules for making filenames

    A. Filenames can be up to thirty–two characters long.

    B. The first character may be any character except a digit.

    C. Filenames can contain only the following characters: A through Z, 0 through 9, – & $ * __ . / Characters *not* permitted in filenames include: imbedded blanks, special characters such as: , ? ! @ ;

    D. Upper and lower case letters are treated as upper case by PRIMOS. (Letters entered in lower–case will be converted to upper–case.)

FIND string

The FIND command finds the first line below the current line which begins with **string**, and makes that line the current line. If no line beginning with string can be found, the pointer stops at the end of the file, and the word BOTTOM is printed. The FIND command distinguishes between upper and lower case letters in string. If you are unable to FIND old lines in your file, but can FIND newly inserted ones and your current display is entirely in upper case letters, the CASE control on your terminal may be in the wrong position.

FIND(n) string

You can also FIND a **string** starting on other than column 1 of the line, by specifying the number of the column within parentheses directly after the command word.

GMODIFY

EDITOR'S GMODIFY command is a string–oriented editing routine which permits you to edit the current line on a character–by–character/column–by–column basis.

If you have given an invalid or illegal subcommand, EDITOR will ignore the rest of the current command line, and the current line will NOT be replaced by the buffer contents.

Subcommands may be separated by any number of spaces (or no spaces) but may NOT be separated by commas.

See Section 8, SAMPLE SESSIONS, for complete information on the GMODIFY command.

$$\text{INPUT} \quad \left\{ \begin{array}{l} \textbf{(ASR)} \\ \textbf{(PTR)} \\ \textbf{(TTY)} \end{array} \right\}$$

The INPUT command reads text from the specified input devices:

- ASR – Teletype paper tape reader
- PTR – High–speed paper tape reader
- TTY – Terminal (default)

The opening parenthesis is required. An INPUT command without an argument puts you in input mode.

INSERT newline

The INSERT commands inserts **newline** following the current line; the inserted line then becomes the current line. The first space after the command word separates it from newline.

You may *not* use semicolons in newline. A semicolon or ⌈ RETURN ⌉ will signal the end of the line being inserted. Unless SYMBOL has been used to make semicolon (;) a valid text character – in which case the character that has replaced ; *cannot* be inserted.

KILL character

The KILL command changes the kill character from whatever it previously was to the specified **character.** There must be **exactly** one space between the command word KILL and the new kill character. You can check the current value of the kill character with the SYMBOL command.

LINESZ n

The LINESZ command allows the line size of the EDITOR to be changed. In no case can it be made larger than the maximum size (initial size at start–up), but within that limit, it can be set or reset as much as desired. The parameter **n** is the number of characters in the line. If EDLIN, STRA, STRB, STRC, or INLIN contain more characters than the new maximum they is shortened appropriately. If the file contains a line longer than the new size, the message BAD LINE IN FILE is printed and the line truncated (a

few characters may be lost). When a line is typed that is too long, the ? is printed, the BELL is rung, and the line is truncated. In INPUT mode, the remainder of the typed line becomes a new line. The character which caused the overflow, however, is lost.

This command is useful **only** when you are creating a new file, and should be used before you enter any text. The default value of n is 144, which is the maximum value.

LOAD filename

The LOAD command copies the contents of **filename** into the EDITOR work file just below the current line. The pointer will then be just below the end of the LOADed text, positioned at a null line.

LOAD does not affect the contents of the original file filename in any way; it simply copies the contents of filename into the work file.

Remember: LOADED text will not go in your permanent files in your UFD unless you FILE at the end of the EDITing session.

LOCATE string

The LOCATE command finds the first line below the current line which contains **string** and makes it the current line. If no line containing string is found, BOTTOM will be printed and the pointer left at the end of the file.

The first space after LOCATE separates string from the command word. All other blanks will be considered part of string. You may use the wild card (!) and match–n–spaces (#) symbols in string.

MODE COUNT [start] [increment] [width] $\left\{ \begin{array}{l} \textbf{PRINT} \\ \textbf{BLANK} \\ \textbf{SUPPRESS} \end{array} \right\}$

MODE COUNT allows you to increment a counter symbol with every use and replaces it in the text by the current value, whenever the counter symbol occurs in the commands: APPEND, INSERT, OVERLAY, RETYPE, GMODIFY (with A, I, O, R subcommands).

The meanings of the COUNT parameters are:

- **start** – initial value for the counter >0 (default $= 1$)
- **increment** – initial increment $\neq 0$ (default $= 1$)
- **width** – field width (number of digits) $1 < $ width < 5 (default $= 5$)
- **PRINT** – Print leading zeroes (default)
- **SUPPRESS** – Do **not** print leading zeroes
- **BLANK** – Replace leading zeroes with blanks

The counter character may be redefined via the SYMBOL command. Its default is the @.

MODE NCOUNT frees the count character as a normal printing character. This command does not affect the values of the counter, increment, width, or the PRINT, SUPPRESS or BLANK parameters. MODE NCOUNT is the default.

MODE $\left\{ \begin{array}{l} \textbf{NUMBER} \\ \textbf{NNUMBER} \end{array} \right\}$.

The command **MODE NUMBER** causes EDITOR to add the line number (as printed by WHERE) in front of each line of the file when displayed either by verification responses or by the PRINT command. The line number is *not* part of the actual file – it is a number EDITOR uses for its own reference. You cannot FIND or LOCATE line numbers; however, you can determine a line's number via WHERE, and go directly to a line via POINT.

The command **MODE NNUMBER** turns off the display of line numbers. MODE NNUMBER is the default mode.

MODE $\left\{ \begin{array}{l} \textbf{COLUMN} \\ \textbf{NCOLUMN} \end{array} \right\}$

The command **MODE COLUMN** causes a column header display to be printed every time you enter input mode during an EDITOR session. The command **MODE NCOLUMN** turns off the column header display. NCOLUMN is the default mode.

MODE $\left\{ \begin{array}{l} \textbf{PROMPT} \\ \textbf{NPROMPT} \end{array} \right\}$

The command **MODE PROMPT** causes EDITOR to start printing prompt characters. **MODE NPROMPT** is the default mode, and is reset every time you leave EDITOR.

The default prompts are the ampersand (&) for INPUT mode and the dollar sign ($) for EDIT mode.

MODE $\left\{ \begin{array}{l} \textbf{PRALL} \\ \textbf{PRUPPER} \\ \textbf{PRLOWER} \end{array} \right\}$

The case modes allow you to indicate upper and lower case letters on terminals which have only upper case display.

MODE PRALL (default) is for terminals with both upper and lower case letters; it accepts and prints mixed case text. In MODE PRALL, an upper case only terminal will output mixed case text as unflagged upper case text even though the file really contains mixed case.

MODEs PRUPPER and **PRLOWER** accept and output case–flagged upper–case letters (i.e., upper–case letters with the actual case flagged). PRUPPER assumes that each line of input and output begins with upper–case letters; PRLOWER assumes the lines begin with lower–case letters. Upper–case is signalled by an up–arrow–U (↑U) before a series

of letters which are meant to be upper–case; lower– case by up–arrow–L (↑L) before lower case.

MODIFY/string–1/string–2/ [G] [n]

The MODIFY command changes **string–1** to **string–2** like CHANGE, but does not affect the spacing of characters in the original line. MODIFY operates in the following manner:

1. Locates string–1 in current line.

2. Starting at the beginning of string–1, replaces with blanks as many characters as there are in string–1 or string–2, whichever is longer.

3. Copies string–2 onto the current line beginning at the first newly–made blank.

The first character after the command word MODIFY (or abbreviation) is used as the delimiter in the command. Any character including the semicolon (;) and the space may be used as a delimiter instead of the slash.

If the letter **G** (for General) is specified, MODIFY alters *every* occurrence of string–1 on a line. If you *don't* specify G, only the *first* incidence of string–1 is modified. If the value of **n** is 1,0 or unspecified, EDITOR will only MODIFY on the current line. If a value other than 0 or 1 is specified, EDITOR will inspect and MODIFY n lines starting at the current line, and leave the pointer positioned at the nth line. If there are less than n lines in the file below the current line, the pointer will be left below the last line (at .NULL.) and the message BOTTOM will be printed. EDITOR will print out all modified lines, plus the last line examined.

NOTE

1. Remember to go to the TOP before MODIFYing the file as a whole.

2. You can omit the closing delimiter if you end the command with a
   ⌈ RETURN ⌉ .

3. You can specify the semicolon (;) as a text character within the delimiters — i.e., if you used "@" every place in your file where you wanted to use ";", then the command sequence

   TOP, MODIFY/@/;/g9999

   would change all the @'s to ;'s. (Make sure n is greater than the number of lines in your file.)

MOVE   buffer–1   ⎰ **buffer– 2** ⎱
                  ⎱ **/string/** ⎰

The MOVE command moves one line of text into **buffer–1** from **buffer–2** or **string**. The Prime–supplied buffers are:

- **EDLIN** (line you are typing at terminal)

- **INLIN** (current line in EDITOR file)

- **STR.1** (also called STRA)
- **STR.2** (also called STRB)
- **STR.3** (also called STRC)

The delimiters around string can be any non–alphabetic character including comma (,) or semicolon (;), which do not appear in string. If MOVE is the last command on the line, the closing delimiter may be omitted.

The command: MOVE buffer–1// clears buffer–1. See Section 8 for more information.

NEXT [n]

The NEXT command moves the pointer **n** lines. Positive values of n move the pointer to the bottom, negative values to the top. If n is 0 or unspecified, the default value of 1 is used.

If a NEXT command would move you to or beyond TOP or BOTTOM, the pointer stops at the appropriate null line.

NFIND string

The NFIND command finds the first line below the current line which does not begin with **string,** and makes that line the new current line.

There must be *EXACTLY* one space between NFIND and string. If NFIND can't find a line which doesn't begin with string, the pointer is left at BOTTOM.

Like FIND, you can NFIND beginning on a column other than column 1:

NFIND(n) string

Remember: There must be no spaces between NFIND and **(n)**, and exactly one space between (n) and string. The parentheses ( ) are required.

OUTPUT $\left\{ \begin{array}{l} \textbf{(DISPLAY)} \\ \textbf{(TTY)} \end{array} \right\}$

While all PRINT commands will print on the user terminal (TTY), the command **OUTPUT DISPLAY** permits you to send verification output to a DISPLAY terminal. **OUTPUT TTY** is the default; if no device is specified, TTY is assumed.

(Using the DISPLAY command option calls for a 9600 Baud display terminal to be connected to Port 3 of the System Option Controller. If you don't have one, this command is of no use to you.)

OVERLAY string

The OVERLAY command superimposes the indicated **string** over the current line, beginning with column 1 as follows:

1. Any character in string EXCEPT !, /, "/ ?, or ; will replace the character in the corresponding column of the current line. (Note: ↑" and ↑? are each considered to be a single text character.)

2. An exclamation (!) in string forces a space in the corresponding column.

3. A space ( ) in string leaves the original character unchanged.

4. The tab (\) causes tabbing to the next tab stop.

5. ", ?, and ; have their usual control functions of Erase, Kill, and end-of-command-string.

PAUSE

The PAUSE command allows you to return to PRIMOS level without ending your EDITOR session. PAUSE preserves EDITOR's work file with all your input and/or editing, and holds the pointer at the current line.

PAUSE is useful, among other things, if you wish to check your UFD for potentially duplicate filenames before doing a FILE. To return to EDITOR, type START — and a carriage-return. You are now back in EDITOR, where you left off.

NOTE

You cannot return to a specific EDITOR session via START if:

1. You type ED again

2. You do anything other than an internal PRIMOS command (LISTF, CREATE, DELETE, CNAME, ATTACH).

3. You log out.

POINT n

The POINT command positions the pointer at line **n** (makes line n the current line).

The value of n must be > 0. If n is greater than the number of lines in the file, the pointer is left at the bottom.

The line numbers are not actually part of your work file; EDITOR generates them for its own reference. You can determine line numbers specifically via WHERE or generally via MODE NUMBER.

PRINT [n]

The PRINT command prints **n** lines, including the current line, and makes the last PRINTed line the new current line.

If n is negative, EDITOR first backs up n lines, beginning the count with the current line, and then prints one line.

The space between PRINT and n is *optional*. A PRINT immediately after the following commands yields .NULL. ; BOTTOM, DELETE, DUNLOAD, LOAD, TOP.

QUIT

The QUIT command tells EDITOR you want to return to PRIMOS level and do not want to save the input or changes you have done. If you have created/modified the file during the session, EDITOR will respond with:

FILE MODIFIED, OK TO QUIT?

This message asks whether EDITOR may throw away the work file.

A YES (or Y, YE, O, OK, or null line `RETURN`) response QUITs you; you will get back an upper-case OK response, meaning you're at PRIMOS level. Any response except YES, YE, Y, O, or OK produces a PLEASE FILE (see FILE); doing a FILE, with or without a filename (depending on the circumstances), automatically QUITs you.

If you did not create or modify a file, saying QUIT returns you to PRIMOS with an OK.

RETYPE string

The RETYPE command deletes the current line and replaces it with the text in **string.** Exactly one space must be between the command word RETYPE and string. The string is terminated by either a semicolon (;) or a `RETURN` .

RETYPE followed by one space before a carriage–return will act as a DELETE, erasing the current line and leaving the pointer at the NULL line. RETYPE followed immediately by a carriage–return yields the error message: BAD R.

SYMBOL name character

The SYMBOL command changes the character value of a special symbol name to the new **character.** The special symbols and their initial (system default) values are:

Special Symbol Name	Character
KILL	?
ERASE	''
WILD	!
BLANKS	#
TAB	\
ESCAPE	↑
CPROMPT	$
DPROMPT	&
COUNTE	@
SEMICOLON	;

You *cannot* use any of the following as a special symbol character:

- Multiple characters (e.e., ABC)

- comma (,)

- space ( )

- asterisk (*)

- any character currently in use as a Special Symbol except the CPROMPT or DPROMPT

You can check the current value of your symbols with the PSYMBOL command.

```
psymbol
KILL    ?
ERASE   "
BLANKS  #
WILD    !
TAB     \
ESCAPE  ^^
CPROMP  $
DPROMP  &
COUNTE  @
SEMICO  ;
symbol kill {
symbol tab *
BAD SYMBOL
symbol tab ?
symbol escape %
 symbol wild ~
psymbol
KILL    {
ERASE   "
BLANKS  #
WILD    ~
TAB     ?
ESCAPE  %
CPROMP  $
DPROMP  &
COUNTE  @
SEMICO  ;
```

EDITOR automatically resets the symbols back to their default values whenever you QUIT or FILE.

The Kill and Erase characters can also be changed via the KILL and ERASE commands.

NOTE

The SYMBOL command is helpful if you often use semicolons, quotes, or the question mark in your text.

TOP

The TOP command repositions the pointer at the top of the file, just above the first line of text. The contents of the current line are .NULL. An INSERT after TOP puts text above the first line of text.

It is often a good idea to go to TOP before doing a FIND, a LOCATE, or a multi–line CHANGE, or MODIFY.

UNLOAD filename [n]

The UNLOAD command copies **n** lines beginning at the current line from the file begin EDITed into a new file named **filename.** If n is 0 or omitted, it is assmed to be 1. A negative value for n UNLOADS the preceding n–1 lines and the current line, in the correct order.

The last line UNLOADed is the new current line. Make sure *no* file named filename previously exists, or, if one does, that you don't want it – EDITOR will delete your old file to write the new one.

UNLOAD filename TO string

The UNLOAD TO command copies lines in the work file into a new file named **filename;** lines are copied starting with the current line and continuing until a line containing **string** is found, or until BOTTOM is reached. Remember: *Don't* specify a filename currently in use unless you want the old file wiped out. There *must* be a blank space between TO and string.

VERIFY

The VERIFY command causes verification output — i.e., automatic printing of the new current line — whenever any of the following commands are given; APPEND, CHANGE, FIND, GMODIFY, LOCATE, MODIFY, NEXT, NFIND, OVERLAY, POINT.

Verification output can be suppressed via the BRIEF command. VERIFY is the default mode. If the OUTPUT (DISPLAY) command has been given, verification output goes to the remote display.

WHERE

The WHERE command prints the current line number.

XEQ buffer

The XEQ command executes the contents of **buffer** as a command line. The possible buffers are:

- **INLIN** (current line)
- **EDLIN** (line you are typing)
- **STR.1** (also called STRA)
- **STR.2** (also called STRB)
- **STR.3** (also called STRC)

It is possible to nest XEQ commands so that commands in string buffers can be executed as subroutines. A n+1-deep stack is maintained to allow the maximum proper nesting of XEQ commands in the n string buffers. Recursive XEQ's continue to work

as always but the stacking still allows "returning" should the recursion stop. In case of an error, BOTTOM or TOP, XEQing stops and the stack is cleared.

Note that the "*" command causes execution to start at the beginning of the CURRENT command line. All the rules for nesting *'s apply, but *'s may be used in an XEQ sequence so long as only one * is outstanding at any time. See Section 8 for more information.

*[n]

The Repeat (*) causes the previous command string to be repeated **n** times, or until TOP or BOTTOM is reached.

INTRODUCTION

This section contains complete information on all RUNOFF commands. The commands are listed in alphabetic order.

EXPLANATION OF THE COMMAND FORMAT

A RUNOFF command consists of a period (.) followed by a **command** word, and possible one or more **parameters**, as in:

.COMMAND parameter

The period

The period beginning a line identifies a command, which is to be obeyed, as opposed to text, which is to be processed. The period (.) must precede all RUNOFF commands that appear in your source file. In RUNOFF command mode, the period may be omitted.

Command words

A command word is a word which specifies an action. You may input command words in either upper or lower case (or a combination). In this manual, the letter shown in rust in a command word indicate the minimum acceptable abbreviation. For example:

.PARAGRAPH

Any of the following would be acceptable:

.P, .p, .para.,.PaRaGrAph, etc.

Parameters

A parameter, in RUNOFF, is either the name of a file, a number specifying a line, column, page number, or number of lines or columns, or a piece of text, depending on the command.

Numerical parameters are represented in the formats by:

- The letter **i** for a general number
- The letter **m** for a number of spaces
- The letter **n** for a number of lines

Some commands do not use parameters, some require them, and in others, they are optional. In this manual, parameters which are enclosed in brackets, e.g., **[filename]** are optional. If you omit them, RUNOFF will use the appropriate default value.

RUNOFF COMMANDS

- ⬚ RETURN ⬚

The NULL command (period carriage-return, or just carriage-return) tells RUNOFF to begin (or continue) processing the source text file.

.*comment

The asterisk (*) indicates a **comment** line. It signals RUNOFF to ignore this text line. The *comment permits you to insert explanatory comments into your source text files for later reference.

.+text

The + (plus sign) inserts subsequent **text** on the line verbatim (exactly as it appears in the source file) into the output file.

If previous lines were FILLed or ADJUSTed, any mode changes implied by the text on this line apply only to this line. The + causes an implicit BREAK.

.>text

The right-angle bracket (>) inserts subsequent **text** on this line verbatim, centered between the left and right margins. If previous lines were FILLed or ADJUSTed, any mode changes implied by the text apply only to this line. The > causes an implicit BREAK.

./left-text/center-text/right-text/

Apportion the text as **left**-justified, **center**ed, and **right**-justified portions. You may omit any of the text portions, but must still give all four delimiters. The slash (/) is the *only* permissible delimiter here. The / may not be used as a text character. For example, the source file statement:

```
./Left Wing/Middle of Road/Right Wing/
```

processes to:
```
Left Wing                Middle of Road                Right Wing
```

.ADJUST

The ADJUST commands tells RUNOFF to fill each line with words and adjust the spacing between words until each line is right justified. ADJUST causes an implicit BREAK.

The ADJUST command takes priority over any previous mode command. ADJUST is the default state. See Section 5 for more information.

.BLANK character

The BLANK command resets the value of the blank **character**. Each blank character indicates a required blank in the source file and indicates words which must not be broken; these blanks will then be neither suppressed nor padded during FILLing and ADJUSTing. For example, the source file:

```
.blank &
On November&12,&1923,  noted author H.G.&Wells visited New&York&City,
and travelled by boat to Staten&Island.
```

processes to:

```
On November 12, 1923, noted author  H.G. Wells
visited  New York City,  and travelled by boat
to Staten Island.
```

The default blank character is CONTROL-@

.BMARGIN [n]

The BMARGIN command sets the bottom margin to **n** lines. If n is zero or omitted, the bottom margin is reset to the default of five lines. If necessary, RUNOFF will recalculate placement of the footers. BMARGIN causes both an implicit BREAK and EJECT.

.BREAK

The BREAK command signals the end of a paragraph. RUNOFF will stop FILLing the current output line, and not ADJUST it.

Many commands cause an implicit BREAK, i.e., it is as if you said BREAK after them. See Section 5 for complete information.

.CMARGIN [m]

The CMARGIN command sets the inter-column spacing to **m** spaces. This command cannot be used if you have only one text column at the time. If m is zero or omitted,

the column margin is *reset* to its default value of five spaces. The CMARGIN command causes an implicit BREAK and EJECT.

.COLUMNS [i]

The COLUMNS command sets the number of text columns on a page to **i** columns. This command causes an implicit BREAK and EJECT. The default number of columns is 1. Here's an example of 2-column text.

If you want a combination of single and multi-column material on a given page, you must:

1. RUNOFF the different section as a dinstinct file.
2. Trim the margins off, using EDITOR.
3. Use RUNOFF's INSERT command to insert it, enclosed in a pair of NFILL/ADJUST commands.

.DDOWN [heading]

Go down one level and generate a label. (Decimalization command.)

.DDSUPPRESS [heading]

Go down one level but do not print label. (Decimalization command.)

.DLEVEL level

Define the **level** at which numbering continues as level. (Decimalization command.)

.DLIMIT [limit]

Reset maximum level of label to be entered in TOFC file (Table of Contents command.)

.DNEXT [heading]

Generate a label (and optional **heading**) on the current decimal label. (Decimalization command.)

.DNSUPPRESS [heading]

Generate a **heading** on the current decimal level, but do not generate a label. (Decimalization command.)

.DRESET level value

Reset the level number to **level** and the next number on this level to **value.** (Decimalization command.)

.DSKIP level [before-skip] [after-skip]

For each label on level level, skip **before-skip** lines before the label and **after-skip** lines after it. (Decimalization command.)

.DUP [n-levels]

Decrement the level number by **n-levels.** (Decimalization command.)

.DEFINE symbol-name value

The DEFINE command tells RUNOFF to find every use of the specified **symbol-name** enclosed in symbol characters, and replace it with the indicated **value.** For example, the command DEFINE Name Arthur Trent tells RUNOFF to replace *every* occurrence of the word %Name% by Arthur Trent when processing the source file into the output file.

The symbol-name may be up to six letters in length and *may not* contain any imbedded commas, parentheses or spaces. You may use two digit numbers as symbol names, e.g., 03, 15, etc., but not single digit numbers. A value may be up to thirty characters in length and may contain any characters.

Up to 60 symbols may be defined at any one time. If you previously defined symbols which you are not using, and need to now define more new ones, you can UNDEFINE some or all of the old symbols. (See UNDEFINE.) Also, you may re-DEFINE an existing symbol; the new definition will replace the old one.

.EFOOTER/left-text/center-text/right-text/

The EFOOTER command sets up the footer on even-numbered pages. Any character may be used as a delimiter. The delimiters (the slash, in above format) define the contents of the left, center, and right portions of the footer; their presence is required even if there is no text in a given portion, as the following examples demonstrate:

> .EFOOTER/left-text///
> .EFOOTER@@center-text@@
> .EFOOTER***right-text*

.EHEADER/left-text/center-text/right-text/

The EHEADER command sets up a header on even-numbered pages. Any character may be used as a delimiter. The delimiters (slashes in above format) define the contents of the left, center and right portions of the header; their presence is required even if there is no text in a given portion, as the following examples demonstrate:

> .EHEADER/left-text///
> .EHEADER@@center-text@@
> .EHEADER***left-text*

.EJECT

The EJECT command makes the next text line go on a new page. EJECT forces a new page even if there is more than one text column on the page at the time.

Certain commands, namely: BMARGIN, COLUMNS, CMARGIN, LENGTH, QUIT,

SMARGIN, TMARGIN and WIDTH, cause an EJECT implicitly when given; this prevents pages from having contrasting text formats (even if you want it). See the COLUMNS command for a way around this limitation.

.ERASE character

The ERASE command redefines the RUNOFF erase character, which is usable only in RUNOFF command mode. ERASE changes the current value of the erase character to the specified **character.** Once the erase character has been changed you cannot reset it until the end of your RUNOFF session.

.ERRGO

The ERRGO command suppresses the printing of error messages and prompts at the terminal while RUNOFF is processing the source file. Instead, RUNOFF will process the entire file, complete with the results of any errors. (This is particularly important when running RUNOFF as a phantom user, as otherwise a source file with errors will bomb out part way through the run.) The default is NERRGO (print error message and wait for prompt).

.FILE filename

The FILE command specified the name of the output file. All source text, following a FILE command, is processed into the given **filename.** You can give the FILE command more than once in a source file, if necessary.

Make sure you give the FILE command before any text or formatting commands! If you do not give the FILE command in RUNOFF command mode, or do not specify a filename at the time you give the command, RUNOFF asks you for one when it begins processing with this message:

ENTER OUTPUT FILE TREENAME

If you give a filename which already exists in your current (or specified) UFD, RUNOFF inquires:

OK TO DELETE OLD filename?

If you respond YES (or Y, OK, YE), RUNOFF deletes the existing file of that name, and assigns the name to the new output file. If you respond any other way, RUNOFF asks:

NEW NAME:

Give a filename not in your directory.

```
$
PROCESSING...
ENTER OUTPUT FILE TREENAME: $memo
OK TO DELETE OLD$memo? no
NEW NAME: $new.memo
```

The FILE command can be used at any point in your source file; if you want to process text into a different file halfway through the source, give the FILE command plus a new output filename. RUNOFF EJECTs before processing any output to the new file.

.FILL

Enter FILL and NADJUST mode. Text is filled, but not adjusted. This is true even if you were in ADJUST; the FILL command overrides any previous mode. You must specify ADJUST to enter ADJUST mode. This command causes a BREAK.

.FLOAT filename

Operates in the same manner as PICTURE, but inserts the contents of an external file rather than simply leaving space. **Filename** is the name of an external file containing text, a table, or illustration captions.

The external file may contain any legal combination of text and RUNOFF commands, but must be terminated by a RETURN statement.

You have the option of beginning the external file with a size control line, in the form:

> .*n

where **n** is the number of lines in the file (including lines reserved by FLOAT, INSERT, or PICTURE commands). Thus if the file is shorter than one page and fits on the page currently being processed, it is inserted immediately following the current line. If larger, the current page is completed, and the external file is inserted starting at the top of the next page.

If the size control line is omitted, RUNOFF assumes that the FLOAT file is one page or more in length, and inserts it starting at the top of the next page.

Parameters may be "passed" to the FLOAT file in the same way as with the INSERT command (See INSERT). FLOAT commands may be nested up to 10 levels.

.FOOTER /left-text/center-text/right-text/

The FOOTER command sets up the footer on all pages. Any characters may be used as the delimiter. The delimiters (slashes in the format) define the contents of the left, center, and right portions of the footer. Their presence is required even if there is no text in a given portion, as the following examples demonstrate:

> **.FOOTER/left-text///**
> **.FOOTER@@center-text@@**
> **.FOOTER/left-text//right-text/**
> **.FOOTER***right-text***

.FROM i

The FROM command defines the page number of the first page in the processed output file which you want to appear as processed output. This number is the page number as printed by the # sign, not the sequential page number. The sequential page number is the total page count, whereas you can reset the count at any time, to any value, using PAGEN. If **i** is zero, or the command is not given, RUNOFF starts printing at page 1.

The FROM and TO commands are particularly useful when you want to examine a few selected pages of processed output from a long file.

.HEADER /left-text/center-text/right-text/

The HEADER command sets up the header on all pages. Any character may be used as a delimiter. The delimiters define the contents of the left, center and right portions of the header; their presence is required even if there is no text in a given portion, as the following examples demonstrate:

        .HEADER/left-text///
        .HEADER@@center-text@@
        .HEADER/left-text//right-text/
        .HEADER***right-text*

.HYPHEN character

The HYPHEN command defines RUNOFF's phantom hyphen **character.** The phantom hyphen may be inserted between syllables of words within your source text to signal to RUNOFF that it may, if necessary, hyphenate a word at this point. The default value for the RUNOFF phantom hyphen is the rubout key (or its octal value of '377).

The phantom hyphen itself does not appear in the processed output in either case; for example, if the word

        antidisestab.↑377.lishment

would run over the right margin, RUNOFF would process the word as follows.

        text text text text antidisestab
        lishment text text text...

But, if the word fits within the output line, RUNOFF processes it as:

        text text antidisestablishment text
        text text text

.INDENT [m]

Indents the left margin **m** spaces to the *right* relative to the current left margin.

If m is 0, 5, or omitted, RUNOFF will indent 5. Negative values are *not* permitted; use UNDENT to move the indentation back.

.INDEX string

The INDEX command makes an entry into an index file defined by the IXFILE command. The entire **string,** up to but not including the carriage-return, is written to the index file with the current page number appended. (see .IXFILE). The .INDEX command should immediately follow the text containing the indexed phrase to insure that the proper page number is used. If the .INDEX command immediately follows a page eject, the previous page number is used.

.INSERT filename [(symbol-0, symbol-1...,symbol-9)]

The INSERT command processes and inserts an external source file **filename.** This file may contain both text and RUNOFF commands (including further INSERT commands). This command is used to insert alternate files within the current input file(s). Inserts may be nested up to 13 levels deep.

Defining **symbol-values** for INSERTed files: RUNOFF will automatically replace any symbol-name in an INSERTed file with symbol-values defined either in the INSERTed file or in the main source file.

If you know *exactly* what information you wish to pass along to an INSERTed file via symbols, RUNOFF provides a way to list pre-named symbol-values within the INSERT command. You may follow filename with a list of up to ten symbol-values separated by commas. No symbol-value may be more than six characters in length, and the entire list must be enclosed in parentheses. RUNOFF automatically assigns these ten symbol-values to the names %0%, %1%, %2%,...%9, in corresponding order. Unassigned symbol-names have. NULL. values. In other words, the command:

.insert MEMO (Jan05, 1984, Sales, Stock Report)

is equivalent to the commands:

```
.define 0 Jan05
.define 1 1984
.define 2 Sales
.define 3 503
.define 4 Stock
.define 5 Report
.define 6
.define 7
.define 8
.insert MEMO
```

MEMO would then look like this:

Date:	%0% %1%
To:	%2% Dept # %3%)
Subject:	%4% %5%

This then permits you to INSERT the same file several times using different parameter values each time, as in:

```
.blank &
.insert MEMO (Jan05,1984,Sales,503,Stock,Report)
.insert MEMO (Feb08,1984,Pets,444,Studio,nosy)
.insert MEMO (Mar15,1984,Acctg,666,Stock,Report)
```

There is an additional format for the INSERT command:

.INSERT [unit] [(symbol-1, symbol-2...,symbol-9)]

If filename is specified, that file will be opened, and input taken from it. If a unit number **unit** is specified, that unit is assumed to be open, and the input is taken from it. If no name or unit number is specified, RUNOFF assumes a unit has already been opened and the input is taken from it. In command mode, this last option is equivalent to typing a null line to begin processing. This command is useful, in conjunction with the RETURN command.

.IXFILE filename

The IXFILE command defines the name of an index file. If IXFILE is given, RUNOFF copies all subsequent INDEX statements in the source program into **filename** with page numbers appended. RUNOFF does not combine multiple entries or format the file; this must be done via EDITOR.

If you do not specify IXFILE, and your RUNOFF source file contains one or more IN-DEX entries, RUNOFF asks for the name of the index file:

INDEX FILE

You may then either specify filename, which becomes the index file, or enter a [RETURN] . The latter tells RUNOFF that you do not want an index or index file; RUNOFF ignores all subsequent INDEX entries in the file. (This is equivalent to doing a NIXFILE.)

If filename already exists, RUNOFF asks if it is OK to delete the old file filename; if you answer NO, RUNOFF then requests a new filename.

.KILL character

The KILL command re-defines the RUNOFF kill **character,** which can only be used in RUNOFF command mode. This command cannot be used in the source input file.

The default value of the RUNOFF kill character is the question mark (?). If you change this via the KILL command, you cannot reset it back to the default value during the RUNOFF session; however, PRIMOS automatically resets the KILL character value when you are finished.

.LENGTH [n]

The length command defines physical page length, including top and bottom margins, as **n** lines. RUNOFF will recalculate placement of headers and footers. The LENGTH command causes an implicit BREAK and EJECT. The default value of n is 66 lines. The maximum is 132. (At 6 lines = inch, 66 lines = 11 inches.)

.NADJUST

The NADJUST command stops RUNOFF from justifying the right margin of output text. If you were in ADJUST mode, giving the NADJUST command is equivalent to saying FILL; if you were already in FILL, NADJUST has no effect.

.NEED n

The NEED command specifies that a block of **n** printing lines (actual lines of text, not page lines) are needed for a body of text to follow.

.NFILL

The NFILL command tells RUNOFF to stop FILLing and ADJUSTing, and causes an implicit BREAK. in NFILL mode, the tab character is recognized. Wherever it appears NFILL takes priority over any previous mode. NFILL mode is particularly useful for formatting tables.

.NIXFILE

The NIXFILE command tells RUNOFF that despite the inclusion of INDEX commands in your RUNOFF source file, you do not want to generate an index file.

.NPARAGRAPH

The NPARAGRAPH command resets the paragraph values to their default of INDENT 0, SKIP 1. NPARAGRAPH does not signal a new paragraph. It just resets these values.

.NPAUSE

THE NPAUSE command turns off the between-page pause activated by the PAUSE command. NPAUSE is the default.

.NPERFORATE

The NPERFORATE command de-activates the perforation marks between pages, as activated by PERFORATE, and EJECTS to a new page.

.NTTY

The NTTY command tells RUNOFF not to print processed output at the terminal. NTTY is the default.

.OFOOTER /left-text/center-text/right-text/

The OFOOTER command sets up the footer on odd-numbered pages. Any character may be used as the delimiter. The delimiters define the contents of the left, center and

right portions of the Footer. Their presence is required even if there is no text in a given portion, as the following examples demonstrate:

> **.OFOOTER/left-text///**
> **.OFOOTER@@center-text@@**
> **.OFOOTER/left-text//right-text/**
> **.OFOOTER***right-text***

.OHEADER /left-text/center-text/right-text/

The OHEADER command sets up the header on odd-numbered pages. Any character may be used as a delimiter. The delimiters define the contents of the left, center and right portions of the header; their presence is required even if there is no text in a given portion, as the following examples demonstrate:

> **.OHEADER/left-text///**
> **.OHEADER@@center-text@@**
> **.OHEADER/left-text//right-text/**
> **.OHEADER***right-text***

.PAGEN i

The PAGEN command specifies a new starting page number. The next page to begin after this command will be numbered **i**. The number will be inserted wherever the **#** character is used in a HEADER or FOOTER command. For example:

> **.pagen 5**
> **.HEADER /text/text/Page # /**

makes the next page have a header with "Page 5".

.PARAGRAPH [m] [n]

The PARAGRAPH command signals the beginning of a new paragraph; RUNOFF does a BREAK in the current output line, and then INDENTs the first line of a new paragraph **m** from the current left margin after SKIPing **n** lines. For example, the source file:

```
...just in time to see it pop down a large
rabbit hole under the hedge.
.paragraph 5 1
In another moment down went Alice after it,
never once considering how in the world she was
to get out again.
.paragraph
The rabbit hole went straight down like a tunnel
for some way, and then dipped suddenly down...
```

processes to:

```
...just in time to see it pop down a  large  rabbit
hole under the hedge.

     In another moment down went  Alice  after  it,
never  once considering how in the world she was to
get out again.

     The rabbit hole  went  straight  down  like  a
tunnel  for  some  way,  and  then  dipped suddenly
down...
```

If you do not specify m or n, RUNOFF uses the most recently specified values. The default values for m and n are INDENT 0, SKIP 1. The NPARAGRAPH command resets m and n to these default values without actually starting a new paragraphing.

You can also signal a paragraph in FILL or ADJUST Modes by beginning an input line with a space. This is known as implicit paragraphing.

The command PARAGRAPH 0 0 is equivalent to BREAK.

For information on "hanging indents" (negative indention), see Section 8. SAMPLE SESSIONS.

.PAUSE $\begin{Bmatrix} \mathbf{[1]} \\ \mathbf{0} \end{Bmatrix}$

The PAUSE command causes RUNOFF to:

1. Pause when each new page of output is ready to be processed.
2. Ring the terminal bell.
3. Wait until a character is typed at the terminal before processing the new page.

The PAUSE command permits you to:

- On printing terminals, print each page of RUNOFF output on a separate piece of paper.
- On CRT terminals, inspect the output file page by page. (You can also use TERM -XOFF; see Section 4)

Although any character you type will start the printing output on the new page, you probably want to type a non-printing character so you won't have to erase it later. The default is NPAUSE.

The value 1 – "read character from terminal" – is the default value when PAUSE is used. 0 causes RUNOFF to look in a command file.

.PERFORATE [n]

The PERFORATE command prints a line of hyphens as perforation marks between pages, on the terminal and in the output file.

If a value consisting of a positive integer (e.g., 1, 2, 200, etc.) is used for **n**, the perforation line consists only of a hyphen at each end of the line. If you give the PERFORATE command again, with the value of n anything but a positive integer, the regular perforation marks are restored.

The NPERFORATE command turns off the perforation marks.

.PICTURE [n]

The PICTURE command reserves **n** physical page lines (6 lines = 1 inch) as opposed to printing lines, for later insertion of an illustration. The rules are:

1. If n physical lines remain on the current page, these lines are skipped, and if you're in FILL mode, the current line of output text is completed before the skip occurs – i.e., a break does not occur.

2. If n lines do not remain on the page, RUNOFF continues to process the output page, and then skips n lines beginning at the top of the next page (or column).

3. If n is larger than the number of available lines (excluding top and bottom margins) on a page, RUNOFF skips pages and lines until a total of n lines have been skipped.

RUNOFF keeps track of up to ten PICTURE requests at a time, i.e., up to ten requests for which space has not yet been skipped. RUNOFF attempts to satisfy each PICTURE request as soon as possible, putting one PICTURE per page and filling the rest of each page with text.

If you specify more PICTURE requests while there are ten outstanding requests, the additional requested space is added to that of the tenth PICTURE.

.PURGE

The PURGE command forces immediate satisfaction of all outstanding PICTURES and FLOATS on the current (and all nested) FLOAT levels. This is necessary when you do not want a nested FLOAT to extend beyond the end of a higher-level FLOAT.

.QUIT

The QUIT commands returns you from RUNOFF back to PRIMOS command level. If RUNOFF finds a QUIT in the source file, the current page of output is EJECTed before returning to PRIMOS.

Hitting the escape (or Break) key on your terminal has exactly the same effect as the QUIT command. You can restart RUNOFF by typing the START command, with or without a filename (i.e., START or START filename).

If you type QUIT while in RUNOFF command mode, you *may* lose the last page of processed output.

.RBAR [ON]

The RBAR command turns on and off a revision bar. A revision bar is a vertical line (or bar) printed just to the left of the text, to indicate that the text has been changed since some previous edition of the document.

The command **RBAR ON** turns on the revision bar. The bar appears on all subsequent lines until the RBAR command appears again with any parameter value other than ON – e.g., RBAR OFF, RBAR XXX, RBAR 5. In ADJUST mode, if the revision bar is turned off in the middle of a line, the line is marked by a bar. RBAR does not put a revision bar next to any blank line.

.RETURN [i]

The RETURN command returns you from an INSERT or FLOAT file to the previous input file.

If **i** is zero or omitted, the current file is closed if it was actually opened by RUNOFF. If i = 1, the current file is left open. In all cases, return is to the previous input level. if a RETURN is encountered in the primary input file, return is to command mode. If the value i = 1 is used, typing a null line or INSERT with no parameter causes processing to continue in the original file as if nothing had happened. This allows dynamic parameter changes during processing from the TTY. If RETURN is used (i=0) to command mode, then resumption is only possible if the file was not explicitly opened by RUNOFF. Otherwise, only INSERT filename or INSERT i can be used to resume file processing from a new file. The INSERT/RETURN combination is implemented using a file unit stack to insure proper nesting of input files, down to 13 levels. RETURN always returns 1 level and INSERT always goes down 1 level.

.RINDENT [m]

The RINDENT command indents the right margin, by moving it **m** spaces to the *left* of the current right margin. If m is 0 or omitted, the right margin is indented by the default value of 5 spaces. The RUNDENT command resets the right margin.

.RUNDENT [m]

The RUNDENT command resets (undents) the right margin, moving it **m** spaces to the *right* of the current margin.

If m is zero or omitted, the right margin is reset to the original right margin as specified by the SMARGIN command.

.SKIP [n]

Skip **n** printing lines – i.e., n times line spacing. (If double-spacing, skip 2n lines, etc.) SKIP causes an implicit BREAK.

.SMARGIN [m]

The SMARGIN command resets the current side margins (i.e., left and right) to **m** spaces from the edges of the page (as defined by WIDTH or its default value).

If m is 0 or omitted, the side margins are set to the default of 7 spaces. SMARGIN causes an implicit BREAK and/or EJECT.

.SOURCE [n]

The SOURCE command generates a list of line numbers one space to the right of the margin of the output file, each number corresponding to a non-blank text line from the source input file. Line numbers are four digits, enclosed in parentheses.

Each successive line number is n greater than the preceding number, if the value of n is 1 or omitted, the line numbers increase by 1; if n = 2, etc. If n = 0, the line numbering is terminated. The SOURCE command may be given in the source input file or in RUNOFF command mode.

Only source lines that generate output text are numbered; command lines, e.g., PARA, WIDTH, etc., do not appear in processed output, nor are they numbered. If an input line becomes several lines of processed text, only the first of these lines is numbered.

If you have multi-column output, each column will have corresponding source line numbers.

For files put into your output via INSERT or FLOAT, source numbering restarts at 1 for the duration of the external file; at the end of these files, the line number is reset to that of the line which followed the INSERT or FLOAT command.

.SPACE [n]

The SPACE command sets the spacing mode for printing output lines. **n** = 1 is single spacing, n = 2 is double spacing, etc. If n = 0, or omitted, the default of single spacing is used. If n is set to a value larger than the number of available lines (not including margins) per page, only one line is printed per page (column).

The SPACE command does an implicit BREAK.

.STOP

The STOP command is a conditional QUIT. If STOP is encountered in RUNOFF command mode or in the source input file, it is treated as a QUIT. However, if encountered in an inserted or floated file, it is treated as a .RETURN (with i = 0).

In *all* cases, an end-of-file (EOF) on an input file is treated exactly like a STOP command.

.SYCHAR character

The SYCHAR command defines the specified **character** as the delimiter for symbol names – i.e., these characters must enclose symbol-names in the source file. The default value of the SYCHAR character is the percent sign (%).

.TAB character tab-1 tab-2...tab-20

The TAB command defines the current tab character and stops (which are always relative to the current left margin) for RUNOFF. The tab symbol is set by **character,** which can be any character not currently defined by EDITOR or RUNOFF. (If the character has a special meaning in EDITOR, it is processed when you input it, and never actually appears in the source file.) There are no default tab stops.

The AT sign (@) is commonly used as the RUNOFF tab symbol.

CAUTION

1. Remember that the backslash is recognized as a tab symbol only by EDITOR, and is converted to tab spaces immediately upon input-spaces that RUNOFF will suppress in FILL or ADJUST mode.

2. In the TAB command, there must be a space between the command word, the character, and each tab setting. If no character is specified, or no tab stops specified, tabbing does not occur.

3. Tabs must be set in increasing order; otherwise you will receive the message: **ILLEGAL COMMAND.**

In NFILL mode, you may use the tab symbol anywhere on a line, and it will be interpreted correctly. However, in ADJUST and FILL modes, tab symbols are interpreted correctly only if the input line starts with a TAB symbol.

.TMARGIN [n]

The TMARGIN command sets the top margin of the page to **n** lines from the top. The placement of headers and footers is recalculated, if necessary. If the value of n or 0, 7, or omitted, the top margin is reset to the default value of seven lines. The TMARGIN command causes an implicit BREAK and EJECT.

.TO i

The TO command defines the page number of the last page in the output file to be processed. RUNOFF stops processing the source file as soon as page **i** is completed. The page number is the number as printed by #, rather than the sequential page number. The sequential page number is the total page count, whereas you can reset the # page count using PAGEN. The # symbol is replaced by the page number during processing, but RUNOFF knows a page number even if there is no # on the page's format.

If the TO command is not given, RUNOFF processes the entire source file. The FROM and TO commands are particularly useful when you only want to examine a few selected pages of a long source file.

Although these commands are usually entered in RUNOFF command mode, they may be located anywhere in the source file.

.TOFC filename [limit]

Generate a table of contents in a file called **filename.**. If **limit** is specified, only labels down to that level are recorded in the contents file.

.TOFC

Closes the current table of contents file.

.TOFC 0

Turns off the generation of the table of contents file. (A .TOFC 1, turn it on again.)

.TTOFC string

Enters **string** in table of contents file.

.TTY

The TTY command causes RUNOFF output to be printed at the user's terminal as well as written to an output file (if specified).

.UNDEFINE [symbol-name]

The UNDEFINE command removes **symbol-name** from the symbol table. If a symbol-name is not specified, i.e., the command UNDEFINE is given by itself – the entire symbol table is cleared.

.UNDENT [m]

The UNDENT command undents – i.e., moves the current left margin – **m** spaces to the *left*. If the value of m is zero or omitted, the left margin is reset to the original left margin specified by the side margin command, SMARGIN (or its default value of 7 spaces, if not explicitly given).

.WIDOW [n]

The WIDOW command prevents you from having widows of up to **n** lines on your output pages. A widow is one or more lines of text at the bottom of a page which are separated from the rest of the text by blank lines. The WIDOW command tells RUNOFF to check for widows at the bottom of all subsequent pages. If RUNOFF sees a line skip within n+1 lines of the bottom margin, it does an EJECT after that skip. The default value for n is zero.

.WIDTH [m]

The WIDTH command defines the physical page width, including both left and right margins as **m** spaces. The default page width is 85 spaces; the maximum allowable in RUNOFF is 170 spaces. (At 10 spaces = 1 inch; 85 spaces = 8-1/2 inches; 170 spaces = 17 inches.) The WIDTH command causes an implicit BREAK and EJECT.

EDITOR SUBSYSTEM

ED [filename]

Edits existing or new file. Carriage return inputs a line from the terminal. For more information refer to the NEW USER'S GUIDE TO EDITOR AND RUNOFF.

Defaults:

Erase character	"
Kill character	?
Tab character	\
Default tab settings	6, 12, 30
Command /input line terminators	; or carriage return

EDITOR SUBCOMMANDS

APPEND string
Appends **string** to end of current line.

BOTTOM
Moves pointer to bottom of work file.

BRIEF
Suppresses verification output.

CHANGE/string-1/string-2/[G] [n]
Replaces **string-1** with **string-2,** on **n** lines once or generally, **[G].**

DELETE [n]
Deletes **n** lines.

DELETE TO string
Deletes lines from file until **string** is found.

DUNLOAD filename [n]
Copies and deletes **n** lines from file.

DUNLOAD filename TO string
Copies and deletes line until **string** is found.

ERASE character
Makes **character** new erase character.

FILE [filename]
Files work file under **filename** or current filename.

FIND string
Finds first line below current containing **string.**

FIND(n) string
Finds first line below current containing **string** in column **n.**

GMODIFY
Alters current line as specified. See RUNOFF guide, Section 8.

Subcommands:

A/string/	Copy current line and append **string**
B n	Move pointer back **n** spaces
C character	Copy up to but not including **character**
D character	Skip up to but not including **character**
E n	Skip next **n** characters
F	Copy to end of line
I/string/	Insert **string** starting at current column
M n	Copy **n** characters
n	Test for "not character" in next C or D command
O/string/	Overlay **string** starting at current column
R/string/	Retype, using **string,** from current column
S	Move pointer to column 1

INPUT device
Takes input from either (ASR), (PTR), or (TTY) *(default)* device.

INSERT newline
Inserts **newline** below current line.

KILL character
Makes **character** new kill character.

LINESZ [n]
Makes **n** maximum line length.

LOAD filename
Copies **filename** into workfile.

LOCATE string
Finds first line below current line containing **string.**

MODE COLUMN
Displays banner at start of INPUT Mode.

MODE NCOLUMN
De-activates column banner *(default).*

MODE COUNT initial increment width $\left\{ \begin{array}{l} \textbf{BLANKS} \\ \textbf{ZEROES} \\ \textbf{SUPPRESS} \end{array} \right\}$

Activates counter symbol with these values.

MODE NCOUNT
De-activates counter.

MODE NUMBER
Activates printing of line numbers.

MODE NNUMBER
De-activates printing of line numbers *(default).*

MODE PROMPT
Displays prompts for INPUT and EDIT Modes.

MODE NPROMPT
De-activates Mode prompts *(default).*

MODE PRALL
Do not flag character case *(default).*

MODE PRUPPER
Treat as case-flagged uppers, line default upper.

MODE PRLOWER
Treat as case-flagged uppers, line default lower.

MODIFY/string-1/string-2/[G] [n]
Changes **string-1** to **string-2** without changing line alignment.

MOVE buffer-1 $\left\{ \begin{array}{l} \textbf{buffer-2} \\ \textbf{/string/} \end{array} \right\}$

Moves **string** or contents of **buffer-2** into **buffer-1.**

NEXT n
Moves pointer **n** lines.

NFIND(n) string
Moves pointer to next line not containing **string.**

OUTPUT $\left\{ \begin{array}{l} \textbf{(DISPLAY)} \\ \textbf{(TTY)} \end{array} \right\}$

Sends verification output to specified device *(default =* **TTY).**

OVERLAY string
Superimposes **string** on current line.

PAUSE
Freezes EDIT session, goes to PRIMOS-level (return via START).

POINT n
Moves pointer to line **n.**

PRINT [n]
If n>0, prints **n** lines; if n<0, backs up n lines and prints one line.

PSYMBOL
Prints list of current reserved characters.

PTABSET ptab-1 ptab-2 . . .
These are current physical tab stops on device.

PUNCH [n] { (ASR) (PTP) }
Punches **n** lines on indicated device.

QUIT
Leaves EDITOR without saving results of session.

RETYPE string
Deletes current line, replace with **string.**

SYMBOL name character
Makes **character** current value of symbol **name.**

Special Symbol Name	Character
KILL	?
ERASE	"
WILD	!
BLANKS	#
TAB	\
ESCAPE	↑
CPROMPT	$
DPROMPT	&
SEMICOLON	;
COUNTER	@

TABSET tab-1 tab-2 . . . tab-8
Sets these logical tab stops for EDITOR tab symbol.

TOP
Moves pointer to top of file.

UNLOAD filename [n]
Copies **n** lines into filename.

UNLOAD filename TO string
Copies lines into **filename** until **string** is found.

VERIFY
Prints verification of current line (default).

WHERE
Prints current line number.

XEQ buffer
Executes contents of **buffer** as a command line.

*[n]
Repeats preceding commands on line either **n** times or until TOP/BOTTOM reached.

RUNOFF [filename]

Invokes the RUNOFF text formatter to process an existing source file.

Defaults:

Page size	=	8-1/2x11"
Spacing	=	6 lines per inch
Pitch	=	10 characters per inch
top margin	=	7 lines
text depth	=	54 lines
bottom margin	=	5 lines
side margins	=	7 characters
text width	=	71 characters

ADJUST mode
BLANK character is CONTROL-@
ERASE character is double-quote "
HYPHEN character is .RUBOUT.
NERRGO
PARAGRAPH 0 1
SYCHAR character is percentage sign %
TAB character is undefined

Reserved Symbols:

%DATE.1%, %DATE.2%, %0% through %9% used in INSERT & FLOAT commands.

RUNOFF Subcommands

.CR
Starts processing [command mode].

.*text
Comment line — will not appear in output file.

.+text   B
Enters **text** verbatim.

.>text   B
Centers **text** on line.

./left/center/right/   B
Apportions text.

.ADJUST   B
Right-justifies filled output lines (default).

.BLANK character
Defines **character** as the required-blank character (*default:* Control-@).

.BMARGIN [n]   B&E
Sets bottom margin. (*default*=5 lines).

.BREAK
Truncates current line and resumes on next line. No file or adjust.

.CMARGIN [m]   B&E
Sets inter-column margin to **m** spaces (*default*=5 spaces).

.COLUMNS [i]   B&E
Makes **i** columns of text (*default*=1 column).

.DEFINE symbol-name value
Substitutes **value** for all subsequent occurrences of **symbol-name.**

.DDOWN [heading]   B
Goes down 1 level and generates a decimal label.

.DDSUPPRESS [heading]   B
Goes down 1 level; does not print label.

.DINDENT [level] [label-indent] [text-indent]
Sets indents for an indicated **level**.

.DLEVEL [level]
Continues labels at **level**.

.DLIMIT [level]
Resets maximum **level** of label to be entered in TOFC file.

.DNEXT [heading]   B
Generates a label for **heading** on the current level.

.DNSUPPRESS [heading]   B
Starts new block on current level; does not print label.

.DRESET [level] [value]
Resets the level number to **level** and sets next **value.**

.DSKIP [level] [before-skip] [after-skip]
Sets skip amounts for **level.**

.DUP [n-levels]
Decrements the level number by **n-levels.**

.EFOOTER/left/center/right/
Puts this footer on subsequent even-numbered pages.

.EHEADER/left/center/right/
Puts this header on subsequent even-numbered pages.

.EJECT   **B&E**
Proceeds to top of next printing page.

.ERASE character
Defines **character** as the new erase character.

.ERRGO
Suppresses error messages and prompts during processing *(for phantom jobs)*.

.FILE [filename]
Defines output file.

.FILL   **B**
Fills but does not adjust text.

.FLOAT filename
Inserts external **filename** when space is available on page.

.FOOTER/left/center/right
Puts this footer on all subsequent pages.

.FROM [i]
Displays files output beginning on page **i**. *(default=page 1)*.

.HEADER/left/center/right/
Puts this header on all subsequent pages.

.HYPHEN character
Defines **character** as the phantom hyphen character *(default=.RUBOUT.)*.

.INDENT [m]
Indents left margin by **m** additional spaces *(default=5 spaces)*.

.INDEX string
Enters **string** and current page number into index file.

.INSERT filename [0,1,...9
                     value-0,...value-9   **]**
Inserts **filename** into output, passing optional values for symbols %0%,...%9%.

.IX filename
Defines **filename** as the file for IXFILE entries.

.KILL character
Defines **character** as the kill character *(default=?)*.

.LENGTH [n]   **B&E**
Defines length of physical page as **n** lines *(default=66 lines)*.

.NADJUST
Stops adjusting output lines.

.NEED [n]
n continuous lines needed. Does B&E if not enough space on current page.

.NERRGO
Stops processing, displays error messages.

.NFILE
Does not put output into a file.

.NFILL
Stops filling and adjusting output lines. Obeys *all* tab characters.

.NIXFILE
Stops generating index file; ignores all INDEX commands.

.NPARAGRAPH
Resets PARAGRAPH values to indent=0 spaces, skip=1 line.

.NPAUSE
Disables pause between output pages *(default)*.

.NPERFORATE
Disables print perforation marks between pages *(default)*.

.NTTY
Disables display of output at the terminal *(default)*.

.OFOOTER/left/center/right/
Puts this footer on all subsequent odd-numbered pages.

.OHEADER/left/center/right/
Puts this header on all subsequent even-numbered pages.

.PAGEN [i]
Resets current page number.

.PARAGRAPH [m] [n]   **B**
Begins new paragraph; indents **m** spaces, skips **n** lines. *(Default=0 spaces, 1 line)*.

.PAUSE
Pauses between each output page — waits for a character from terminal.

.PERFORATE [n]
Prints perforation marks between pages.

.PICTURE [n]   **B&E**
Reserves **n** blank physical lines for artwork.

.PURGE
Forces *immediate* satisfaction of PICTUREs and FLOATs.

.QUIT   **B&E**
Returns to command mode or PRIMOS/

.RBAR [ON]
ON = Turn on Revision Bar; any other value = turn it off.

.RETURN [i]
Returns to previous source file; if **i**>o, leaves file open.

.RINDENT [m]
Increases right margin by **m** additional spaces *(default = 5 spaces)*.

.RUNDENT [m]
Decreases right margin by **m** spaces *(default = original margin)*.

.SKIP [n]   **B**
Skips **n** printing lines.

.SMARGIN [m]   **B&E**
Resets side margins to **m** spaces each *(default = 7 spaces)*.

.SOURCE [n]
Prints source file line number on output file incremented by **n**.

.SPACE [n]
Sets line spacing to **n** (*default* = 1 space).

.STOP   **B&E**
Conditional QUIT.

.SYCHAR character
Defines **character** as the symbol-name-delimiter character (*default* = %).

.TAB character tab-1 tab-2...tab-20
Defines **character** as the tab character and tab stops (*no default*).

.TMARGIN [n]   **B&E**
Resets top margin (*default* = 7 lines).

.TO [i]
Defines page number of last processed page wanted.

.TOFC filename [level-limit]
Generates a table of contents in **filename. If level-limit** is specified only decimal headings down to that level will be included.

.TOFC
Closes the current table of contents.

.TOFC $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Temporarily turns off the generation of table of contents if **0;** if **1,** turns on again.

.TTOFC [text]
Enters **text** into table of contents at current level.

.TTY
Displays processed output at terminal.

.UNDEFINE [symbol]
Undefines **symbol** or all symbols.

.UNDENT [m]
Moves left margin **m** spaces to left (*default* = original margin).

.WIDOW [n]   **E**
Avoids widows of **n** lines by doing an EJECT if necessary.

.WIDTH [m]   **B&E**
Redefines physical page width to **m** spaces, including margins (*default* = 85 spaces).

INDEX

In this index, references to pages with command formats are shown in **bold face**. Only the first occurence of a command format is highlighted. Pages that have figures are shown in *italic*. Error messages are shown in rust colored letters.