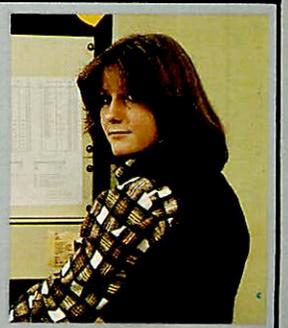
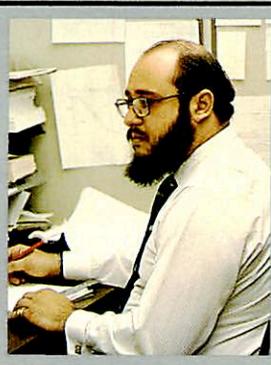
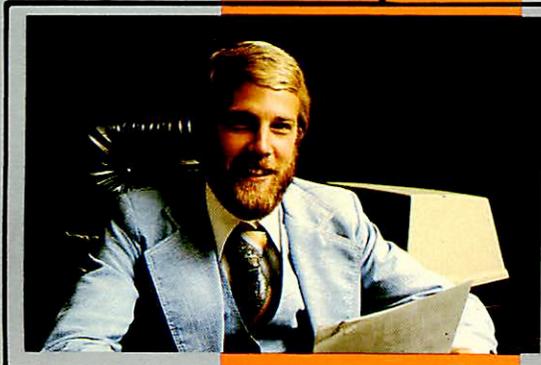
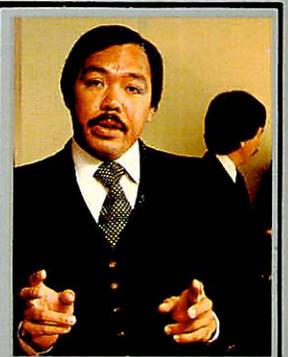
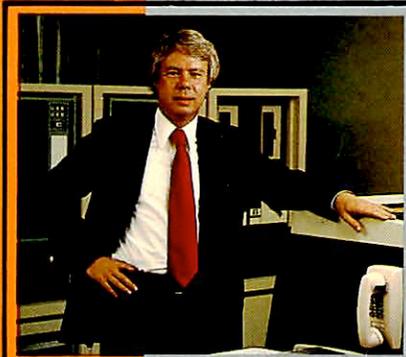
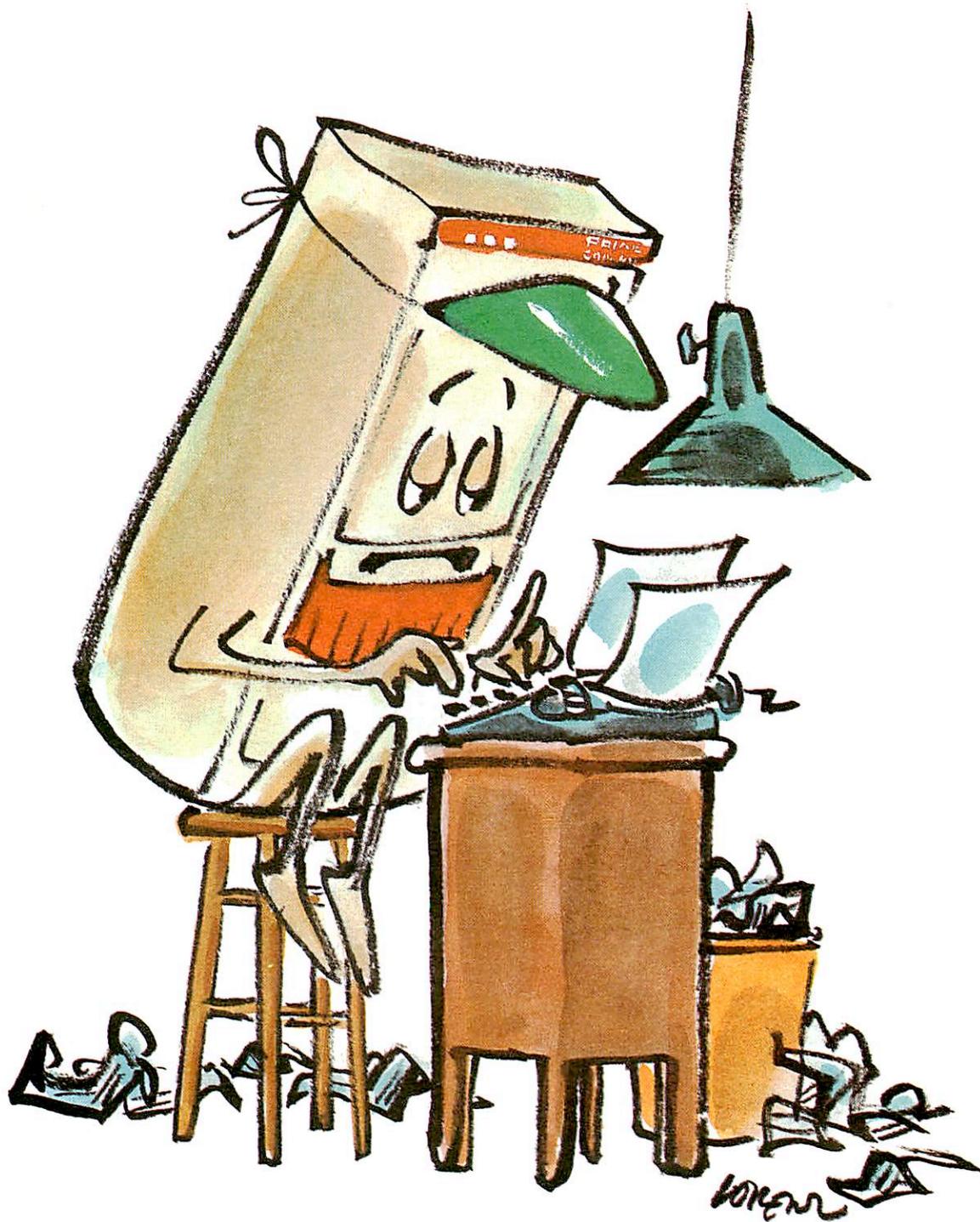


# PRIME Computer

## THE NEW USER'S GUIDE TO EDITOR AND RUNOFF



The New User's Guide to EDITOR and RUNOFF



Published by Prime Computer, Inc.  
Technical Publications Department  
500 Old Connecticut Path  
Framingham, Massachusetts 01701

Copyright © 1978, 1979 and 1980  
by Prime Computer, Inc.

Printed in USA. All rights reserved.

The information contained in this document is subject  
to change without notice and should not be construed  
as a commitment by Prime Computer Incorporated.  
Prime Computer assumes no responsibility for any  
errors that may appear in this document.

First printing, June 1978

Second printing, revisions, February 1980

**Production information:** This book was composed in 10  
and 11 point Melior by **Allied Systems**. The covers were  
printed in 6 colors by **MacDonald & Evans** with  
separations by **Spectrum**. The cover stock was 100#  
Warren LOE Gloss Cover. The text was printed in 6  
colors by **Federated Lithographers**. The text stock was  
50# Mohawk Vellum, Creme white. Layout and design  
was by **William Agush** of Prime Computer. The  
illustrations were by **Lee Lorenz**.

# The New User's Guide to EDITOR and RUNOFF

by Daniel P. Dern

With revisions and enhancements by  
Jacki Forbes and Peter Neilson

Illustrations by Lee Lorenz

---

# 1

## **INTRODUCTION**

Purpose: A Textbook for New Users	1-1
Organization	1-1
Terminals and Computers	1-2
The Computer	1-2
Computerese - Some Basic Terms	1-3
Conventions Used In This Manual	1-6
Terminals	1-8
Your Terminal Keyboard	1-8

# 2

## **USING PRIMOS**

What is PRIMOS?	2-1
User File Directory Names	2-1
How to Correct Typing Errors	2-1
Hitting the Carriage-Return Key	2-2
Getting Ready to Log In	2-2
Connecting the Terminal with the Computer	2-2
Logging In	2-3
Problems in Logging In	2-4
Other Reasons You Can't Log In	2-4
Finding Out What's in Your UFD	2-5
Logging Out	2-6
Notes on Logging Out	2-7

# 3

## **THE ESSENTIALS OF EDITOR**

Editing	3-1
What EDITOR Is	3-1
Conventions in EDITOR	3-2
How EDITOR Works	3-3
Input and Edit Modes	3-3
Entering EDITOR	3-3
Entering Text in Input Mode	3-4
Switching from Input to Edit Mode	3-5
How EDITOR Works on its File	3-5
Giving Commands in Edit Mode	3-6
Switching from Edit to Input Mode	3-6
Basic EDITOR Commands	3-7
EDITOR's Error Messages	3-7
The PRINT Command	3-8
The WHERE Command	3-9
Pointer-Moving Commands	3-9

String-Finding Commands	3-11
Line-Changing Commands	3-12
Ending an EDITOR Session	3-15
Miscellaneous Information	3-18
General Information	3-18
EDITOR's Other Commands	3-18
What's Next	3-20

## 4

### **MORE PRIMOS**

More PRIMOS Commands	4-1
Using Other UFDs-The ATTACH Command	4-1
Making Sub-UFDs Using the CREATE Command	4-3
Using the SLIST and SPOOL Commands	4-5
Renaming and Deleting Files and Sub-UFDs	4-8
The DATE Command	4-8
The CLOSE Command	4-8
Copying Files with FUTIL	4-9
The TERM Command	4-9

## 5

### **THE ESSENTIALS OF RUNOFF**

Introduction	5-1
Creating the Source File	5-2
RUNOFF Commands	5-2
Page-Formatting Commands	5-4
Line-Formatting Commands	5-6
Blank Lines and Paragraph Spacing Commands	5-9
Output Commands The TTY Command	5-11
Running RUNOFF	5-14
RUNOFF Command Errors	5-16
The Processed Output File	5-17

## 6

### **MORE RUNOFF**

Introduction	6-1
Page-Formatting Commands	6-1
Output Options	6-4
Blocks of Text, Artwork and Inserted Files	6-6
Special Characters, Symbols and Conventions	6-7
Indexing	6-11



## “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

# 7

## **RUNOFF DECIMALIZATION**

What is Decimalization?	7-1
Using RUNOFF to do Decimalization	7-2
Format Parameters	7-4
Levels	7-4
Format Commands	7-7
Ways to Generate Decimal Labels	7-8
Level-Resetting Commands	7-8
Generating Unlabeled Headings and Paragraphs	7-9
Miscellaneous Information	7-10
Generating a Table of Contents	7-11
RUNOFF Decimalization Command Summary	7-13

# 8

## **SAMPLE SESSIONS**

Introduction	8-1
Document Formatting	8-1
How To Do Hanging Indents	8-2
Inserting Addresses on Form Letters	8-7
Asterisk Constructions	8-10
The GMODIFY Command	8-12
The MOVE and XEQ Commands	8-15

# 9

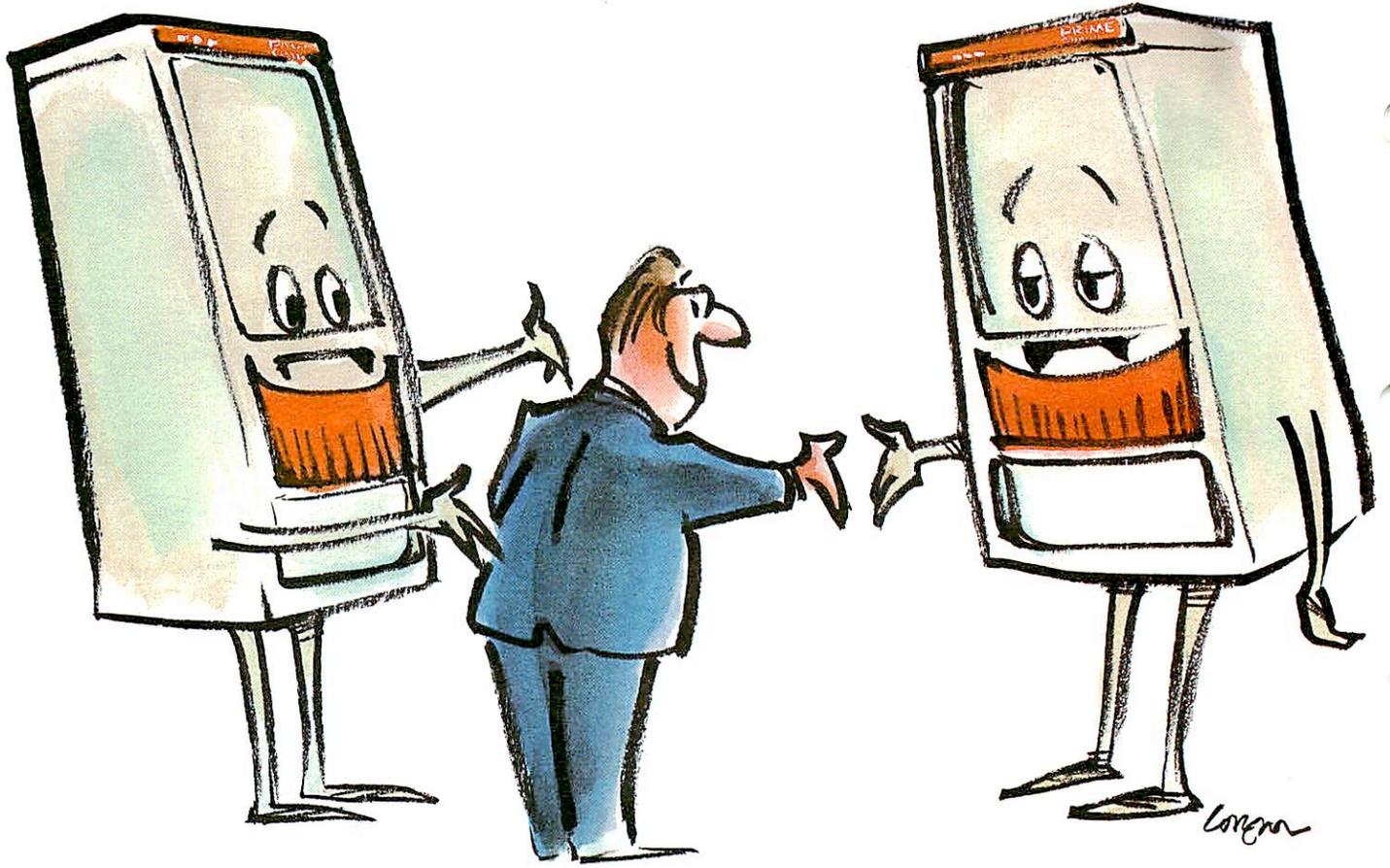
## **THE EDITOR REFERENCE SECTION**

Explanation of the Command Format	9-1
The Commands	9-1

# 10

## **THE RUNOFF REFERENCE SECTION**

Explanation of the Command Format	10-1
RUNOFF Commands	10-1





# Introduction

---

## **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 AND 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 EDITOR and RUNOFF on your computer.

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 RUNOFF commands, you can turn your typed input into pages which are neatly arranged in whatever manner you choose.

## **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:** **USING 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**. Provides 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**. Explains how to do slightly more complicated or non-standard work using RUNOFF.
- Section 7:** **RUNOFF DECIMALIZATION**. Shows how to do decimalized headers and tables of contents.
- Section 8:** **SAMPLE SESSIONS**. Demonstrates common tasks for EDITOR and RUNOFF, plus some advanced techniques.

- Section 9:**           **EDITOR REFERENCE SECTION.** Contains full information on all the EDITOR commands, in alphabetical order.
- Section 10:**       **RUNOFF REFERENCE SECTION.** Contains full information on all the RUNOFF commands, in alphabetical order.

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 unlikely that you are near your Prime computer at the moment. The computer is 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 to as many as 63 working terminals at the same time.

Your terminal may be connected directly to the computer by a length of wire, or over the telephone, using a device called an acoustical coupler. This device translates the computer's and terminal's interaction into signals which can be sent through a phone line.

## THE COMPUTER

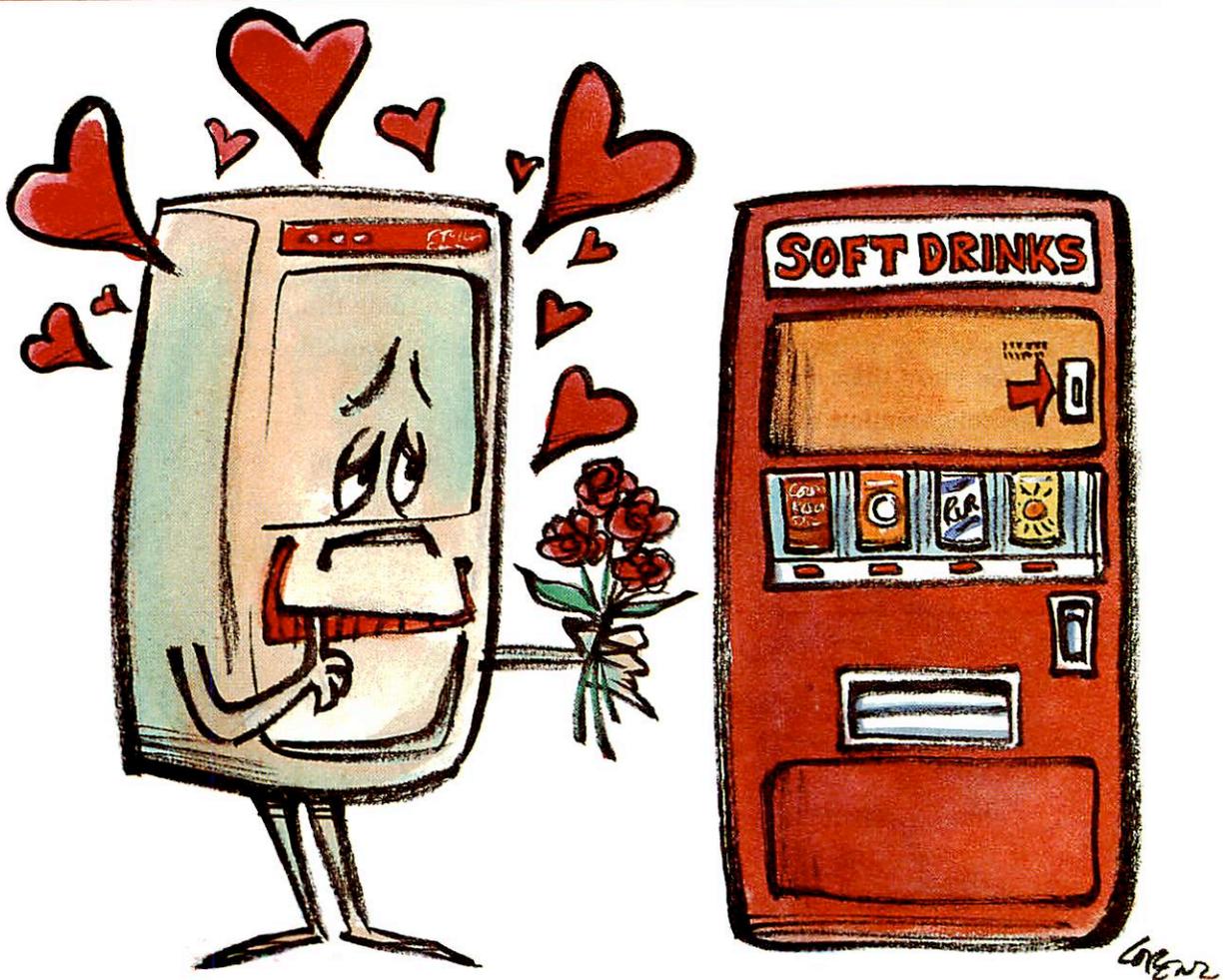
Although computers are complicated machines, you don't need to understand how they work in order to use them. The important thing to remember about a computer is that it is a machine. Like any other machine, it does not think; it merely follows orders and does exactly what you tell it to do. This means you must tell the computer precisely what you want. If you type a command containing an error that can still be understood, it will follow the command the way you typed it, regardless of what you might have meant. Computers do exactly what you want them to do if you explain what you want in words the computer can understand. Since the computer follows your orders precisely, you want to avoid making mistakes. (You do have a chance to correct typing errors, and even retype entire lines before the computer acts on them. This will be explained later.)

You identify yourself to the computer 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.

The computer checks everything you tell it to make sure you have given it reasonable instructions (commands) — so far as it can tell. It recognizes each correct command, in terms of spelling and permissible abbreviations. If for some reason the computer is not happy with what you have told it, it will display an **error message** on your terminal, explaining its interpretation of your mistake.

**Remember:** The computer cannot determine 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 computer can do it, it will.
- **Computers are always consistent.** If the circumstances are the same, the computer will always react in exactly the same way to the same command (or command error).



Computers have no common sense

- **Nothing breaks when you make mistakes.** Computers are built to be used by people. And people are known for making mistakes. It is possible to scramble your own files; however it is difficult to do something that cannot be undone, and with the file access you have, it is next-to-impossible to do anything at your terminal that will actually harm the computer.

With a little practice, you'll be able to analyze what it is that you want to do, and then determine how to make the computer do it. This manual explains how.

### COMPUTERESE — Some basic terms

As in every other field, there are many terms which have specialized meanings when talking with, or about, computers. Below is a partial glossary with brief definitions. Fuller explanations will appear throughout the text as necessary. Additional terms will crop up throughout the manual. New terms will be printed in **rust**, and will be explained as necessary.

**Terminal:** Something looking very much like a typewriter with extra keys, through which you can communicate with the computer.

**Session:** The period of time spent at the terminal between **login** and **logout**.

**Login:** Connecting yourself, at the terminal, with the computer.

**Logout:** Disconnecting yourself, at the terminal, from the computer.

**Text string:** A series of letters, numbers, and symbols ( \* ' < > etc., which have special meaning to the computer), or any combination of letters, numbers and symbols. In this manual we will frequently refer to a **string**, meaning the same thing as text string.

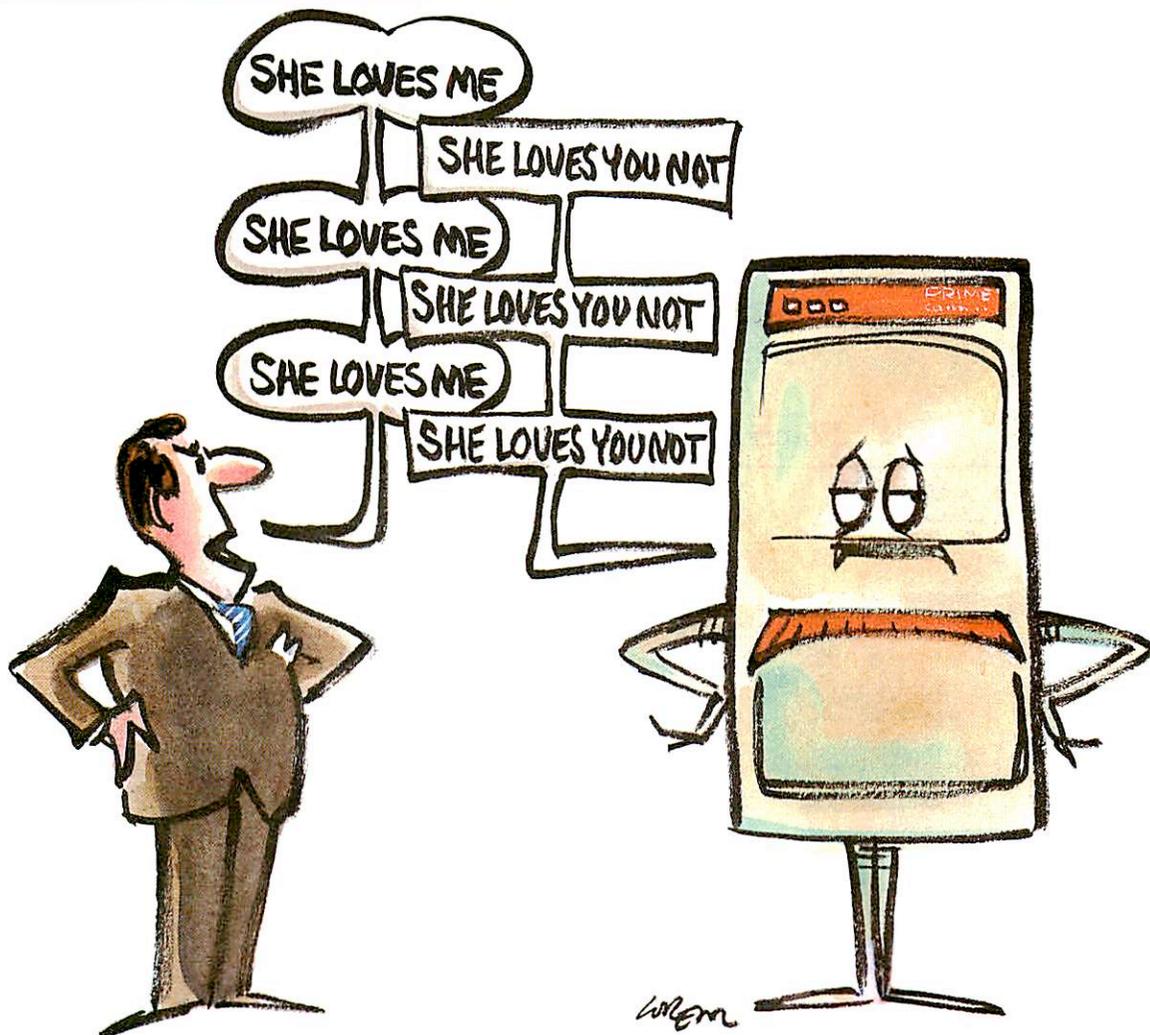
**File:** A series of text strings which are being preserved. All work done in a session is done in a file, or in more than one file.

**Dialog:** The conversation between you and the computer.

**Input:** Your portion of the dialog — what you type at the terminal.

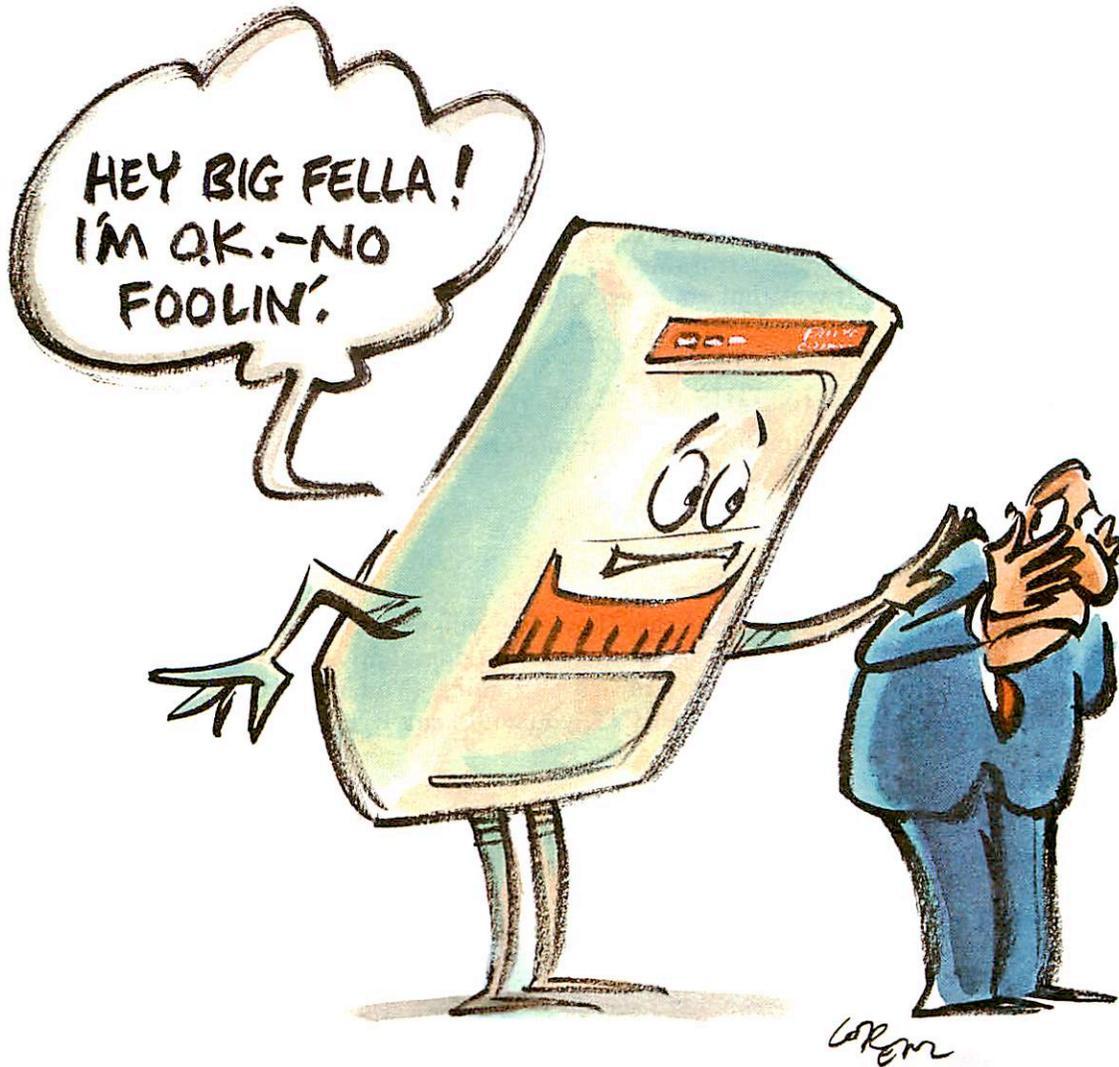
**Data:** Consists of numbers and text strings. Data is the actual information you have typed which the computer processes, as directed by your commands.

---



Computers are always consistent

---



Nothing breaks when you make a mistake

**Command:** A line of input which specifies what you want the computer to do. Commands must be terminated by a RETURN. See Chapter 3.

**Parameter:** Input which defines the command, or tells the computer something about how the command should be performed. Parameters are not always necessary. If a parameter is enclosed in square brackets [] in this manual, the parameter is optional — you may want it, but the command will operate by default if you do not use it. We'll explain more about parameters later in this section.

**Default:** If you input a command without a parameter (an optional parameter), the computer will process the data by default, which means according to its own rules for that command. (However, if you omit a *required* parameter, you'll get an error message.) Defaults for each command will be defined when each command is explained throughout this text.

**Output:** Dialog from the computer.

**Requested information:** Output that you asked to see.

**Reminders:** Prompts, queries and verifications from the computer, to help you get everything the way you want it.

**Messages:** Output from the computer which may be **error messages** or could be messages from your system's computer operator.

**Error messages:** Output from the computer telling you that you asked it to do something it does not understand.

If the computer outputs a question mark (?), it means "That makes no sense" or "What do I do with this?". If it outputs **ER!** (for error), it may also print an abbreviated explanation of what it thinks you did wrong. Errors are usually caused by a typing mistake, or by your forgetting exactly what it was that you were doing.

## 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**.

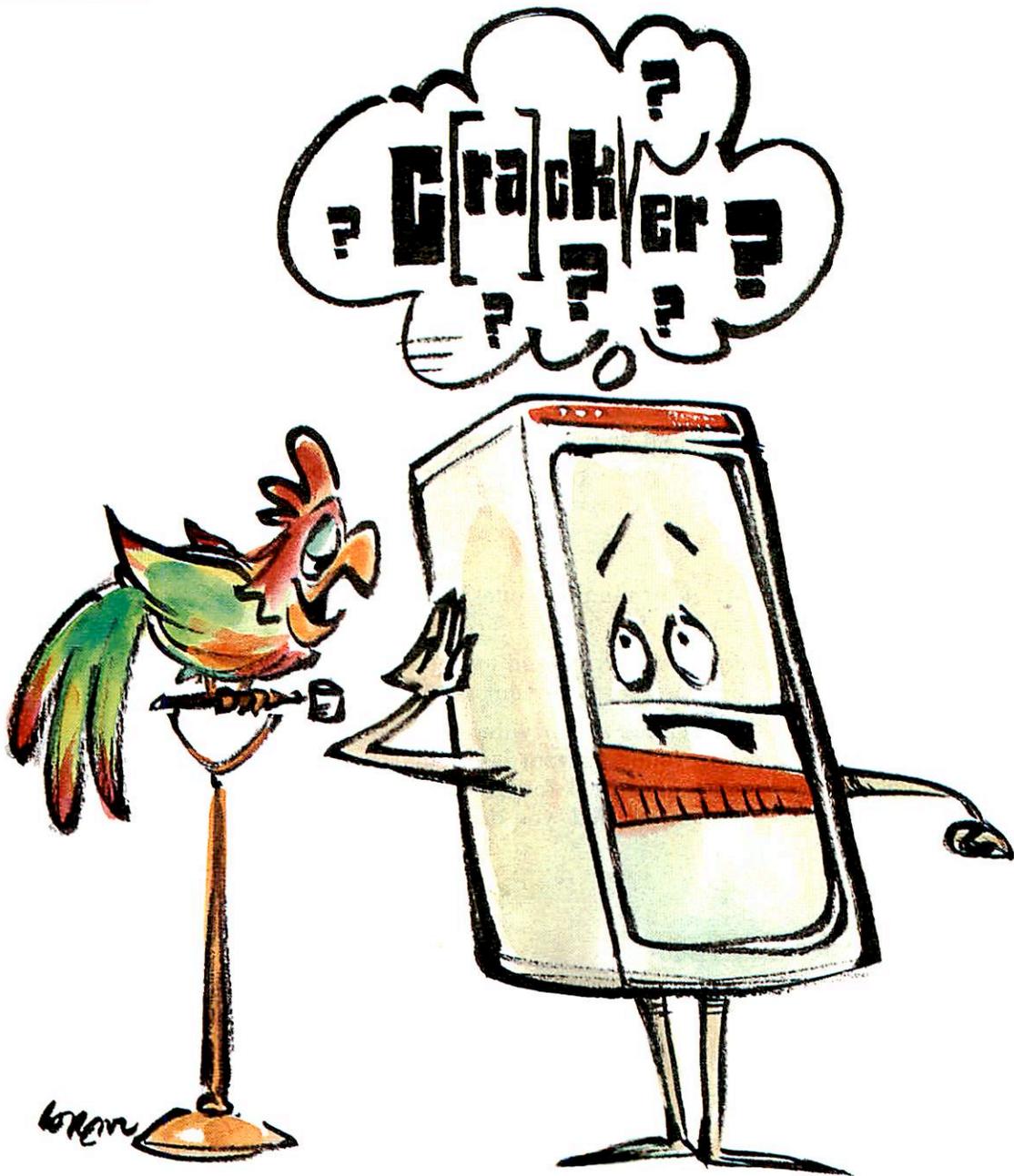
A numeric parameter is represented in a command format by a lower-case letter, usually "**n**". Numeric parameters specify things like numbers of lines, times to do an operation. As an aid, all parameters are shown in bold when they first appear in text, like this: "...**n** lines from filename." The 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)  
**FIN**(5)



Some commands make no sense at all

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 (however, you may enter them in either upper- or lower-case). 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 always have to 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 onto paper 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.

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 and switches
- Special characters

### Terminal controls and switches

Terminal controls and switches affect the special ways in which a terminal performs. Depending on what model terminal you have, these may be switches on the front, side, bottom or back of your terminal; they may be keys to the side of the keyboard. The controls and switches that you need to know about are:

**ON/OFF:** This is the power switch. Terminals sometimes 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 like a specialized typewriter. This switch is often labeled:

ON-LINE/OFFLINE

REMOTE/LOCAL

LINE (and an indicator light which is on in LINE mode and off in LOCAL mode)

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

**HALF DUPLEX/FULL DUPLEX:** This key or switch (sometimes called HALF/FULL or DUPLEX/SIMPLEX) selects whether or not your terminal should "echo" each character on the screen as you type it on the keyboard. Prime systems most often (but not always) use FULL DUPLEX.

If every character you type appears twice, try switching to FULL DUPLEX; if not at all, try HALF DUPLEX.

**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

UPPER/LOWER

UC (for upper-case - when on)

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

**The CONTROL key:** The key labeled CONTROL (or CTRL) 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.

### Special Characters

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

**The backslash:** The Backslash symbol \ has a specialized meaning in EDITOR—the default TAB character.

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

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



# Using PRIMOS

---

## WHAT IS PRIMOS?

PRIMOS is the operating system of your Prime computer.

PRIMOS's 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.

## USER FILE DIRECTORY NAMES

Before you 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, 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

Anytime you type the Erase character, you will erase the most recent character to the left. Unless your System Administrator has changed it, PRIMOS's 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 type 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's 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:

- Applies any Erase or Kill typing corrections you have made.
- 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 UFDs 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 do.

#### 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 terminal 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:

**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.

**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.

**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 070481
OK,
```

Your input is shown in rust. The Kill character (question mark), while not required, is a good practice.

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. 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 "070480" means July 4, 1980. The "OK," tells you that PRIMOS considers you logged in, and is waiting for your next command.

### PROBLEMS IN LOGGING IN

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 ER!	<ul style="list-style-type: none"><li>• You misspelled the word LOGIN. Try again.</li></ul> 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. (LOGIN) ER!	<ul style="list-style-type: none"><li>• You misspelled the name of your UFD. Try again.</li></ul> No UFD by that name exists. Check your UFD name.
Insufficient access rights. (LOGIN) ER!	<ul style="list-style-type: none"><li>• You forgot to type your password. Try again.</li></ul> You misspelled your password - or the password has been changed. See your System Administrator.

### OTHER REASONS YOU CAN'T LOG IN

1. Your terminal is not turned on. (Remember: Some terminals have the power switch located on the back, or even 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 any key plus the RETURN 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 cord of the handset is at the end of the coupler marked CORD.
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.

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 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:

```
OK, listf
UFD=STAFF 3 OWNER
FRANK  MARTHA  ROSEMARY      ALBERT
OK,
```

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

```
OK, listf

UFD=STAFF 3 OWNER

.NULL.

OK,
```

The number after the name of the UFD helps the computer find the UFD. This number is not important to you, (it identifies the "logical disk"). The word next to the number will either be OWNER or NONOWN. 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 spend 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

STAFF (15) LOGGED OUT AT 11'28 070481
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.

Your PRIMOS system may be set to log out inactive terminals after a period of time, perhaps an hour or two.



# 3

## The essentials of EDITOR

---

### EDITING

Editing a file consists of three basic operations:

- Searching
- Examining
- Changing

You search in a file to locate material you wish to change, and you examine it, ensuring that it is, indeed, what you sought. You make changes (or additions, or deletions), and then re-examine, to check that you did them correctly.

### 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:

```
Twinkle, twinkle, little bat!  
How I wonder what you're at!  
Up above the world you fly,  
Like a tea-tray in the sky.
```

EDITOR lets you correct and change it to:

```
Twinkle, twinkle, little star,  
How I wonder what you are!  
Up above the world so high,  
Like a diamond in the sky.
```

## CONVENTIONS IN EDITOR

### The erase and kill characters

EDITOR's erase and kill characters are the same as those of 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 (?).

### Entering an erase or kill character in your text

As you know, the erase and kill characters (default 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 ?"

### 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**.

**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 **text**, **string**, or **newline**, representing a portion of text (or "text string"), 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 the text string.

For example, the command:

```
find    henceforth
```

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

### Example format

In the examples 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 a 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
INPUT
```

Everything you type will then be treated as input, and inserted directly into your work file 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 sales.shoes
EDIT
```

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

1. EDITOR hunts for this file in your UFD. If the file exists EDITOR will find it. (If not, you will get:

```
Not found. FILENAME (OPENR)
ER!
```

Use the LISTF command to check the contents of your UFD.)

2. EDITOR makes a copy of the file and puts the copy in your workfile. The original file remains in storage and is not affected by anything you do (until you are finished and want to make your changes permanent). This permits you to make changes, alterations, and additions to a copy without having to lose the original file. You don't have to worry if you make mistakes, or change your mind, because you can always go back to the original. You can also make several slightly different versions of the original file, saving each variation with its own different filename.
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 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 obviously means you cannot type a semicolon as part of the file. Later we'll show you how to get around this, using the CHANGE command.

### 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 RETURNS in a row.

For example,

```
INPUT          INPUT
text;RETURN    or text RETURN
EDIT          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 place markers 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 full 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 updated when lines are added or deleted, 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

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:

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

The simplest way is RETURN followed by RETURN.

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 after 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 "memo", will be used in examples in this section.

Mr. Damian Chronos, well-known time management consultant, has been engaged to give us a series of informal workshops on the topic of effective time management. He has given these workshops at our branch offices in New York, New Jersey, New Hampshire, and Maine; and at our main office in Connecticut.

If you can answer either of the following questions with a "yes", you should attend:

"Am I often late with an assignment?"

"Do things often seem to back up on me?"

Mr. Chronos uses a technique combining insight and humor; I am sure this will be a delightful and productive experience.

3/25/82

## 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 command** - 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:

```
apen
BAD APPEN  Improper command format
nm
?          Meaningless command
c/offica
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.

```
EDIT
print 3
.NULL.
Mr. Damian Chronos, well-known time management
consultant, has been engaged to give us a
print 6
consultant, has been engaged to give us a
series of informal workshops on the topic of
effective time management. He has given these
workshops at our branch offices in New York,
New Jersey, New Hampshire, and Maine; and at
our main office in Connecticut.
print -5
series of informal workshops on the topic of
print -1
series of informal workshops on the topic of
print-2
consultant, has been engaged to give us a
print 0
consultant, has been engaged to give us a
print
consultant, has been engaged to give us a
q

OK,
```

The space between PRINT and **n** is optional. A PRINT immediately after the TOP or BOTTOM command 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
series of informal workshops on the topic of
where
LINE 3
next 5

where
LINE 8
next-12
TOP
q

```

## POINTER-MOVING COMMANDS

Pointer-moving commands reposition the pointer either 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
print
.NULL.
print2
.NULL.
Mr. Damian Chronos, well-known time management

```

### The BOTTOM command

The BOTTOM command positions the pointer at the bottom of the file, just below the last line of text. You use this command prior to entering new text at the end of an existing file. The format of the BOTTOM command is:

**BOTTOM**

For example:

```
bottom
print
.NULL.
next
BOTTOM
next-1
3/25/82
```

#### 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
Mr. Damian Chronos, well-known time management
next 2
series of informal workshops on the topic of
next4
our main office in Connecticut.
next -5
consultant, has been engaged to give us a
next0
series of informal workshops on the topic of
next
effective time management. He has given these
```

#### 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
series of informal workshops on the topic of
point 6
New Jersey, New Hampshire, and Maine; and at
where
LINE 6
```

## STRING-FINDING COMMANDS

The FIND and LOCATE commands search for the first line beyond the current line which contains a 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 searches for the first line below the current line which contains **string** anywhere in that line. It leaves the pointer positioned to the line it found. 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.

```
locate Hampshire
New Jersey, New Hampshire, and Maine; and at
```

### 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 searches for 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 series
series of informal workshops on the topic of
```

### The NFIND command

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

```
NFIND string
```

For example:

```
OK, ed sales.shoes
EDIT
print6
.NULL.
0938 sandals
0939 pumps
0940 clogs
1040 boots
1041 sneakers
top, nfind 09
1040 boots
```

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).

```
OK, ed sales.shoes
EDIT
find(7) clogs
0940 clogs
```

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

##### **NFIND(n) string**

For example:

```
t
nfind(2) 9
1040 boots
```

### 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 its abbreviation) from the **string** you wish to append. All further blanks will be treated as part of the string. Notice the two blanks after **append** in this example:

```
OK, ed memo
EDIT
point2
consultant, has been engaged to
append give us a
consultant, has been engaged to give us a
```

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.

```
point 1
Mr. Damian Chronos
append , well-known time management
Mr. Damian Chronos, well-known time management
```

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.

```
point 5
workshops at our branch offices in New
append York,
workshops at our branch offices in NewYork,
```

### 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]
```

Use a delimiter which does not occur in the text you are changing. Slash (/) is a common delimiter, but if your text to be changed contains slashes, use a different character, as in these examples:

```
find 3
32/1//82
change:32/1:3/21:
3/21//82
c#/#/#
3/21/82
```

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.

```
point 5
workshops at our branch offices in NewYork,
change/NewYork/New York/
workshops at our branch offices in New York,
next
N. Jersey, N. Hampshire, and Maine; and at
c/N./New/g
New Jersey, New Hampshire, and Maine; and at
```

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 value 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 **n**th line. If there are fewer 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 delimiter.
3. You can specify the semicolon (;) as a text character within the delimiters e.g., 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:

```
point7
main office in Connecticut.
change//our /
our main office in Connecticut.
```

#### 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, only the current line is deleted. The value of **n** may be positive or negative, indicating deletion of the current line plus **n-1** below or above the current line. Since **n** always includes the current line, the commands **d**, **d1**, and **d-1** are equivalent.

```
bottom
n-1
3/25/82
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 without deleting them altogether. Although this technique accumulates files in your UFD, it can be useful if you are afraid of accidentally DELETing 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:

```
point9
If you can answer either of the following
insert questions with a ^"yes^", you should attend:
print
questions with a "yes", you should attend:
```

### 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.

```
point 12
"Am I often late with an assignment?"
r ^"Am I often late with an assignment?^"
print
"Am I often late with an assignment?"
```

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

RETYPE followed immediately by a space and 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 command 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 RETURN) response will QUIT you; you will get back the OK prompt, meaning you're at PRIMOS command 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 auto-

matically QUIT you, having saved your work. If you did not create or modify a file, saying QUIT automatically returns you to PRIMOS.

```
OK, ed memo
EDIT
print 2
.NULL.
Mr. Damian Chronos, well-known time management
quit
OK,
```

```
OK, ed memo
EDIT
delete 3
quit
FILE MODIFIED, OK TO QUIT? yes
OK,
```

```
OK, ed memo
EDIT
delete 3
quit
FILE MODIFIED, OK TO QUIT? no
PLEASE FILE
?
file new.memo
```

#### 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 memo
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 (without any warning), 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.

You don't need to mention the filename when updating an old file:

```
ed sales.jul
l $45l
    Chicken wire          $451812
c/81/8.1/
    Chicken wire          $4518.12
file
```

Although "file sales.jul" would work, it's not good practice—you could inadvertently type "file sales. jun" and wipe out the June sales figures.

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? A YES response QUITs you back to PRIMOS; NO provokes PLEASE FILE, at which point you give the FILE command.

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

OK,
```

The rules for making filenames are:

1. Filenames can be up to 32 characters long.
2. Filenames can contain only the following characters: A through Z, 0 through 9, & - \$ . \_ / #
3. The first character may be any legal character except a digit.
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

```
NEWFILE
Todays-Prices
Highs&Lows
$Monthly.Report
R34587
A-Tale-Of-Two-Cities
```

#### Invalid filenames

```
A?
TWO@JOHN
"EUREAKA"
WHY A DUCK
```

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 memo
EDIT
file memo2

OK,
```

If you are inputting a lot of data into a file, new or old, it is a good idea to FILE it periodically, say after a page or two, to make sure you don't inadvertently lose what you've just entered.

## 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 TABSET command in the **Editor Reference section**. To edit your tabulated files, use the MODIFY or OVERLAY commands.

### Up-Arrow ( ^ or ↑ )

Because the up-arrow is used for inserting literal erase (") and kill (?) characters, you can only include a literal up-arrow by typing two of them. EDITOR will always print this literal up-arrow as two up-arrows. All other programs, such as RUNOFF and SLIST, will print it as a single up-arrow.

(You can enter any ASCII character except line feed, octal 212, as ^nnn where nnn are the 3 octal digits representing the ASCII character.)

## GENERAL INFORMATION ABOUT EDITOR

- EDITOR works on a copy of the original file, called the work file.
- The erase character (usually double-quote) erases the previous character.
- The kill character (usually 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  
OVERLAY string

### Control commands

\*[n]  
XEQ buffer  
PAUSE

### Symbol changing commands

PSYMBOL  
SYMBOL name character  
ERASE character  
KILL character

### Mode changing commands

VERIFY  
BRIEF  
MODE CKPAR  
MODE NCKPAR  
MODE PROMPT  
MODE NPROMPT  
MODE NUMBER  
MODE NNUMBER  
MODE COLUMN  
MODE NCOLUMN  
MODE COUNT  
MODE NCOUNT  
MODE PRALL  
MODE PRUPPER  
MODE PRLLOWER

### 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 the reference section and have no trouble learning additional commands.



You'll easily master your Prime computer

# 4

## More PRIMOS

---

### 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	FUTIL
CLOSE	SLIST
CNAME	SPOOL
CREATE	SORT
DATE	TERM
DELETE	

### 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 staff*

OK, *listf*

UFD=<SALES>STAFF 3 OWNER

FRANK    MARTHA    ROSEMARY            ALBERT

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:

```
NOT FOUND. ufd-name
```

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 word OWNER for owner, or NONOWN 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 may get the message:

```
OK, attach payroll dolla5
Bad password. PAYROL (df_unit_)
ER!
```

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:

```
ER! listf
No UFD attached.
ER!
```

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

```
OK, listf

UFD=<SALES>DATA 2 OWNER

NEW.REPORT REP.DISP SALES.JAN SALES.FEB
SALES.MAR SALES.APR SALES.MAY SALES.JUN SALES.JUL

OK, a STAFF
OK, listf

UFD=<SALES>STAFF 2 OWNER

FRANK MARTHA ROSEMARY ALBERT

OK, a products
OK, listf

UFD=<SALES>PRODUCTS 2 NONOWN
```

```

TOYS  GAMES  CLOTHING  SHOES  HARDWARE  APPLIANCES
POSTERS  RECORDS  PCA  COMPUTERS  PUPPIES  JUNK

```

```

OK, ed games
Insufficient access rights.  GAMES (OPENR)
ER! a products psword
OK, listf

```

```

UFD=<SALES>PRODUCTS  2  OWNER

```

```

TOYS  GAMES  CLOTHING  SHOES  HARDWARE  APPLIANCES
POSTERS  RECORDS  PCA  COMPUTERS  PUPPIES  JUNK

```

```

OK, ed games
EDIT
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 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 is:

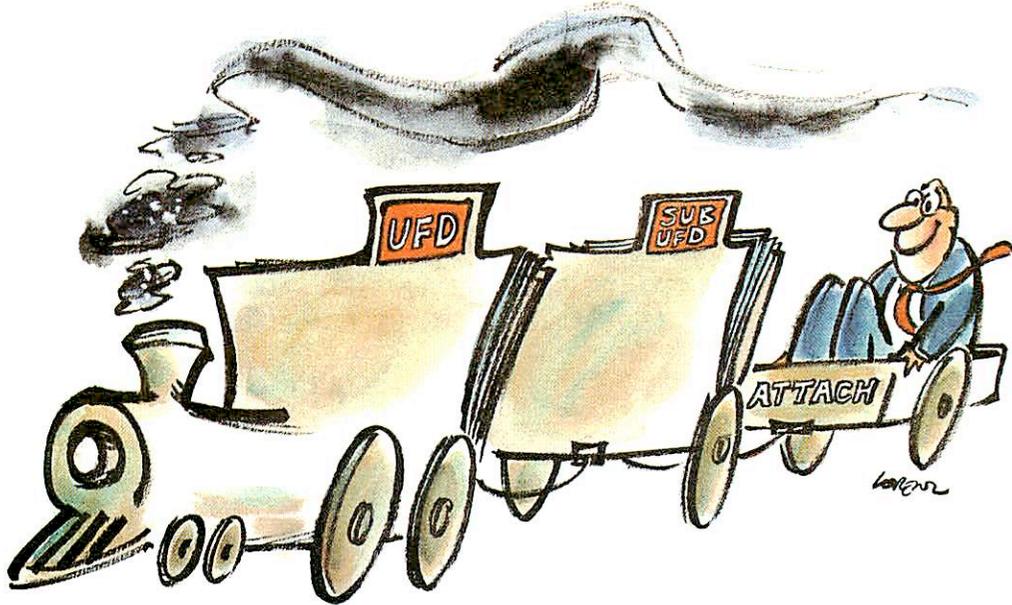
```
ATTACH *>sub-ufd-name [password]
```

You must prefix \*> to the name 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 star-greater-than, 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 two uses for the ATTACH command:

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



#### Attaching to a sub-UFD

We haven't shown all the ways to use ATTACH, but we've covered enough to get you started. More information about ATTACH, and about **pathname** (which we haven't even mentioned), may be found in the **PRIMOS Commands Reference Guide**.

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:

```
OK, login sales
PRIMOS Version 17.2.5
SALES (49) LOGGED IN AT 17'33 021380

OK, listf

UFD=<MRKT02>SALES 1 OWNER

EAST.COAST      WEST.COAST

OK, a *>east.coast
OK, create north
OK, create central
OK, create south
OK, a sales>west.coast
OK, create north
OK, create central
OK, create south
```

```

OK, a sales>east.coast>north
OK, create main
OK, create branch1
OK, create branch2
OK, a sales
OK, listf

UFD=<MRKT02>SALES 1 OWNER

EAST.COAST      WEST.COAST

OK, a *>east.coast
OK, listf

UFD=<MRKT02>SALES>EAST.COAST 1 OWNER

NORTH  CENTRAL SOUTH

OK, a *>north
OK, listf

UFD=<MRKT02>SALES>EAST.COAST>NORTH 1 OWNER

MAIN  BRANCH1 BRANCH2

OK, a sales>west.coast
OK, listf

UFD=<MRKT02>SALES>WEST.COAST 1 OWNER

NORTH  CENTRAL SOUTH

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 format of the SLIST command is:

**SLIST filename**

The SLIST command displays a file on your terminal. On a hard copy terminal, SLIST prints your file on paper. It is most likely, however, that you are using a video terminal. If your file is longer than one screenful, SLIST will flash it by faster than you can read it. To get around this difficulty you can either:

- Examine your file with the EDITOR. See **The Print Command** in Section 3, **Essentials of Editor**.
- Use CONTROL-S and CONTROL-Q to freeze the screen when using SLIST, as described below.

First give the PRIMOS command **TERM -XOFF** which permits you to stop and start terminal output on both video and hard-copy terminals via **CONTROL-S** (Stop terminal output) and **CONTROL-Q** (Resume terminal output). This permits you to “freeze” a 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.

The command **TERM -XOFF** enables the use of CONTROL-S and CONTROL-Q. The command **TERM -NOXOFF** disables them. NOXOFF is the default state.

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 PRTxxx, is n records long.

**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.

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 -list**

[SPOOL REV 17.2]

user	prt	time	name	size	opts/#	form	defer	at: CAROUSEL
FRANK	004	10:42	\$INV-I	3		WHITE		1
MARY	005	10:43	\$INV-II	3		WHITE		1
ROSEM	006	10:44	\$SALESREP	10				1
ALBERT	007	11:11	\$TERMSPEC	55		WHITE	0:12	1
FREDB	009	11:29	T\$ALL	11		WHITE		1
BART	010	11:53	STAFFOUT	8	2	WHITE		1
NELSON	011	12:15	CODE	3		WHITE		

OK,

If, after having given the SPOOL command, for some reason you decide you do not want your file to be spooled—for example, because 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**

or

**SPOOL -CANCEL xxx**

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:

```
File open on delete. It's printing. (SPOOL)
```

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

```
Not found. PRTxxx (SPOOL)
```

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

```
Bad parameter. prtfilename (SPOOL)
```

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

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 hyphen (-) preceding DEFER is required!

The value for **time** can be expressed either in 24-hour time (00:00 = Midnight) or in 12-hour time, 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.

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 message:

```
Illegal name. (SPOOL)
```

This example shows how to use the FORM option:

```
OK, spool $inv-iii -form white
[SPOOL REV 17.2]
Your spool file, PRT014, is 22 records long.
```

### 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. NEW-FILENAME  
ER!
```

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

```
Not found. OLD-FILENAME  
ER!
```

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

**DELETE** { filename }  
          { sub-ufd-name }

You cannot delete a sub-UFD which contains files. To delete such a sub-UFD, you must first delete each file 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:

```
The directory is not empty. DIRECTORY-NAME
```

### **SORT**

The **SORT** command sorts a copy of a file, line by line, in alphabetic or numeric order. Refer to the **PRIMOS Commands Reference 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 date appears in this form:

```
Wednesday, February 13, 1980 2:18PM
```

### **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 closing one file with the **CLOSE** command is:

**CLOSE** filename

The command to close all open files is:

**CLOSE ALL**

### **COPYING FILES WITH FUTIL**

FUTIL is Reference a subsystem that manipulates files. Refer to the **PRIMOS Commands Reference Guide**, for full information.

### **THE TERM COMMAND**

You can use the TERM command of PRIMOS to change your ERASE and KILL characters, and to select certain other terminal-related characteristics. See the **PRIMOS Commands Reference Guide**.



# 5

## The essentials of RUNOFF

---

### 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:

- It breaks text into paragraphs. (In the absence of other commands, RUNOFF assumes that a line beginning with a space is a new paragraph.)
- 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, paragraphing 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. Section 6, **More Runoff**, explains additional commands to aid you in setting up formats and processing files. Section 10, the **Runoff Reference Section** contains a complete alphabetized list of the RUNOFF commands, with full explanations of how to use them.

### 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. RUNOFF accepts your source filename, 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**

**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

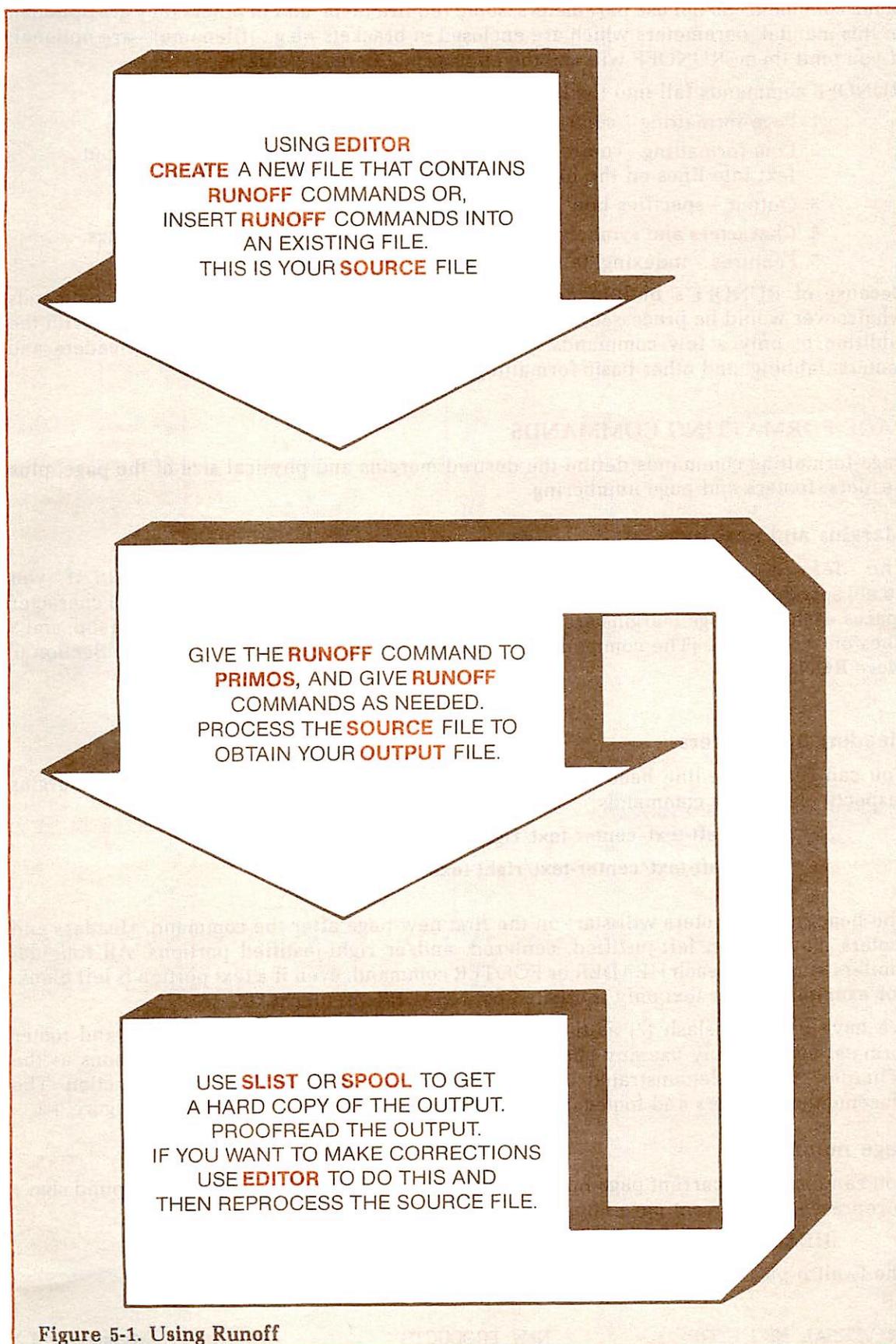


Figure 5-1. Using Runoff

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 and 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, indentation, 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**.)

#### Headers and 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. For example, center text only is created by `.HEADER//center-text//`

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 # to represent 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:

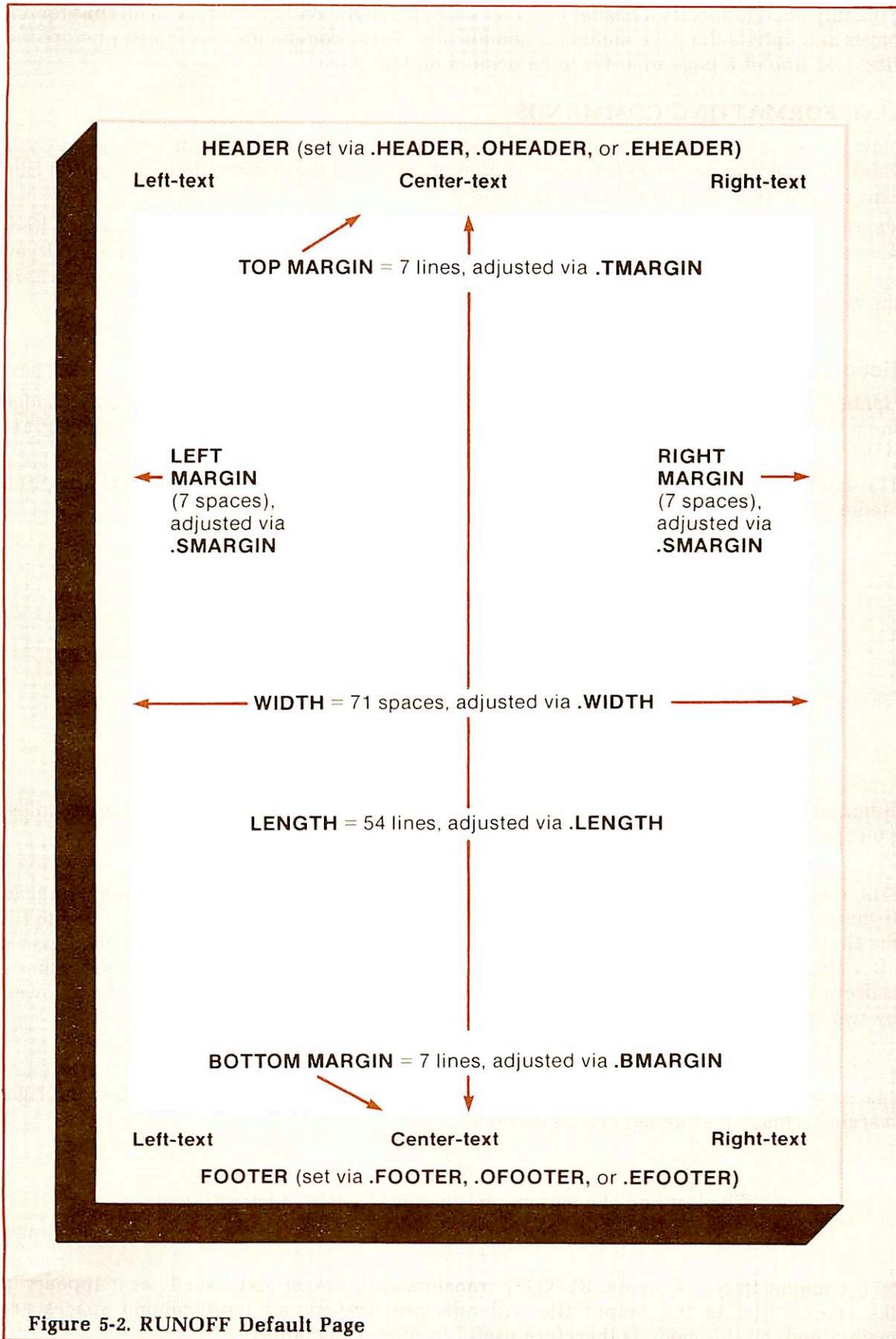


Figure 5-2. RUNOFF Default Page

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 mode commands instruct RUNOFF how to process the source file. There are three modes:

- **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, and will fill each line with text. If there are not enough words on the line, FILL will take words from the second line to fill the first, and so on. FILL puts exactly one space between each word, and exactly two spaces after a period, semicolon, exclamation point, question mark, and colon. If a right parenthesis is preceded by any of these punctuation marks except the semicolon, it will also be followed by two spaces.

**ADJUST mode:** In ADJUST mode, a line is first filled; then 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”.

#### Note

The last line of a paragraph is neither ADJUSTed nor FILLed.

**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. 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-20**

Tab stops must be in ascending order, relative to the left margin. For example,

```
.tab @ 7 47 27
```

will produce an error message when you RUNOFF your file, because 47 is greater than 27. In NFILL mode, RUNOFF recognizes all tabs (up to 20). In FILL and ADJUST modes, the tab character is recognized only in those source lines beginning with a tab character.

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 that in the last two lines of the RUNOFF input example, all @ symbols were ignored as commands and were included as part of the text. In ADJUST and FILL modes, RUNOFF only recognizes an input line's TAB characters as tabulation commands if one occurs in the first column of the line. The tab character and the tab stops can be redefined at any point in the source file. RUNOFF will use the most recent values given.

Tabs are primarily useful for formatting tables.

### Local line-formatting commands

Local line-formatting commands have an effect only for lines on which they appear, 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.

<b>. &gt; text</b>	Center the following line of text.
<b>.+text</b>	Enter exactly as written even if you are in .ADJUST or .FILL modes.
<b>./text/text/text/</b>	Apportion text in left, center and right fields (like a header or footer).
<b>.*text</b>	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 slash may not be used as a text character in this line.

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 an automatic 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 original default values for **m** and **n** are 0 spaces, 1 line. If no values are given, RUNOFF will use the default values; however, specifying **m** and **n** values makes them the new default values. The value of **m** can be positive or negative; for a full discussion, see **How to do Hanging Indents**, in 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
"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:

```
"...You were not to be found, and I was
fearful of losing a bargain."
```

```
"Amontillado!"
```

```
"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, or if a line is preceded by a blank line RUNOFF acts as if there were a PARAGRAPH command there. 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). See .EJECT in Section 10, the **RUNOFF Reference Section**.

### 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 yearly report of the
. *Magrathean Mfg. Corp.
. tty
. header//MAGRATHEAN MANUFACTURING CORPORATION//
. footer//WE WILL CONTINUE TO TAKE PLANET ORDERS//
. skip
. /Yearly Report//July, 2001/
. skip3
. +{{Summary}}
. tab @ 10 24 40
. paragraph 5 2
Although sales this year have not met with previous
expectations, our staff in Small 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
. /Portable fjords/Humanoid Androids/Wide Beam Lasers/
. /Cloned Pets/Faithful Biro's/Sub-ether Radios/
. /Electric Pencils/Photon Drive Cars/Artificial Night/
. /Ultimate Mousetraps/Ersatz Sawdust/Portable Windmills/
. skip
. paragraph
{{The schedule of release is:}}
. skip
. nfill
@Quarter 1@Quarter 2@Quarter 3
. skip
@Androids@Cloned Pets@Sawdust
@Fjords@Lasers@Windmills
@Pencils@Biro's@Mousetraps
@Radios@Cars@Night
. adjust
. paragraph
In addition, Magrathean Mfg. is introducing an entire
new line of products--Complete, tailored-to-order solar systems.
```

RUNOFF processes it into:

MAGRATHEAN MANUFACTURING CORPORATION

Yearly Report

July, 2001

Summary

Although sales this year have not met with previous expectations, our staff in Small 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.

These new products include:

Portable fjords	Humanoid Androids	Wide Beam Lasers
Cloned Pets	Faithful Biros	Sub-ether Radios
Electric Pencils	Photon Drive Cars	Artificial Night
Ultimate Mousetraps	Ersatz Sawdust	Portable Windmills

The schedule of release is:

Quarter 1	Quarter 2	Quarter 3
Androids	Cloned Pets	Sawdust
Fjords	Lasers	Windmills
Pencils	Biros	Mousetraps
Radios	Cars	Night

In addition, Magrathean Mfg. is introducing an entire new line of products--Complete, tailored-to-order solar systems.

WE WILL CONTINUE TO TAKE PLANET 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. When you have finished EDITing your source file by inserting RUNOFF commands, and 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 that contains RUNOFF commands. RUNOFF will announce itself by displaying the word "RUNOFF" plus the version number of RUNOFF which is on your system:

```
OK, runoff form.letter  
RUNOFF REV 17.2  
COMMANDS  
$
```

3. If you did not specify a filename when you gave the RUNOFF command, 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—and 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" and the Command Mode prompt, which is the dollar sign (\$). You may now give general RUNOFF commands directly from your terminal. Commands entered in command mode are not put into your source file. When entering commands in command mode, you need not use the period preceding the command. RUNOFF will continue to issue the \$ prompt until you hit a carriage return without issuing a command on that line:

```
OK, runoff form.letter  
RUNOFF REV 17.2  
COMMANDS  
$ header/Magrathean Manufacturing/New Products/page #1/  
$ 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 (see Output Options in Section 6), 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:

NEW NAME:

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 are 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 your terminal, after the \$ prompt, 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. SPOOL or SLIST your output file.
7. 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 frogs
GO
RUNOFF REV 17.2
COMMANDS
$ tty
$ space 2
$
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, and grammar.

Command errors can occur either when you create 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 few typical errors are:

- Typing a command (such as .P) and text on the same line.
- Forgetting closing delimiters in a command line
- Forgetting the period before a command.
- Unwanted implicit .PARAGRAPH caused by a blank line or by a space in column 1.

Some of these errors cannot be detected by RUNOFF. For example, RUNOFF has no way to discover that a dot is missing on a command.

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 the erroneous source line into your output file. It places the erroneous line between a pair of blank lines, at the current output line:

```
          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. (This does **not** correct your source file.)
- 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) *****
```

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 wish to make any changes:

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.

# 6

## More RUNOFF

---

### 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 allows insertion of 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 indentation.
- 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).

#### Page size commands

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

```
.LENGTH [n]  
.WIDTH [m]
```

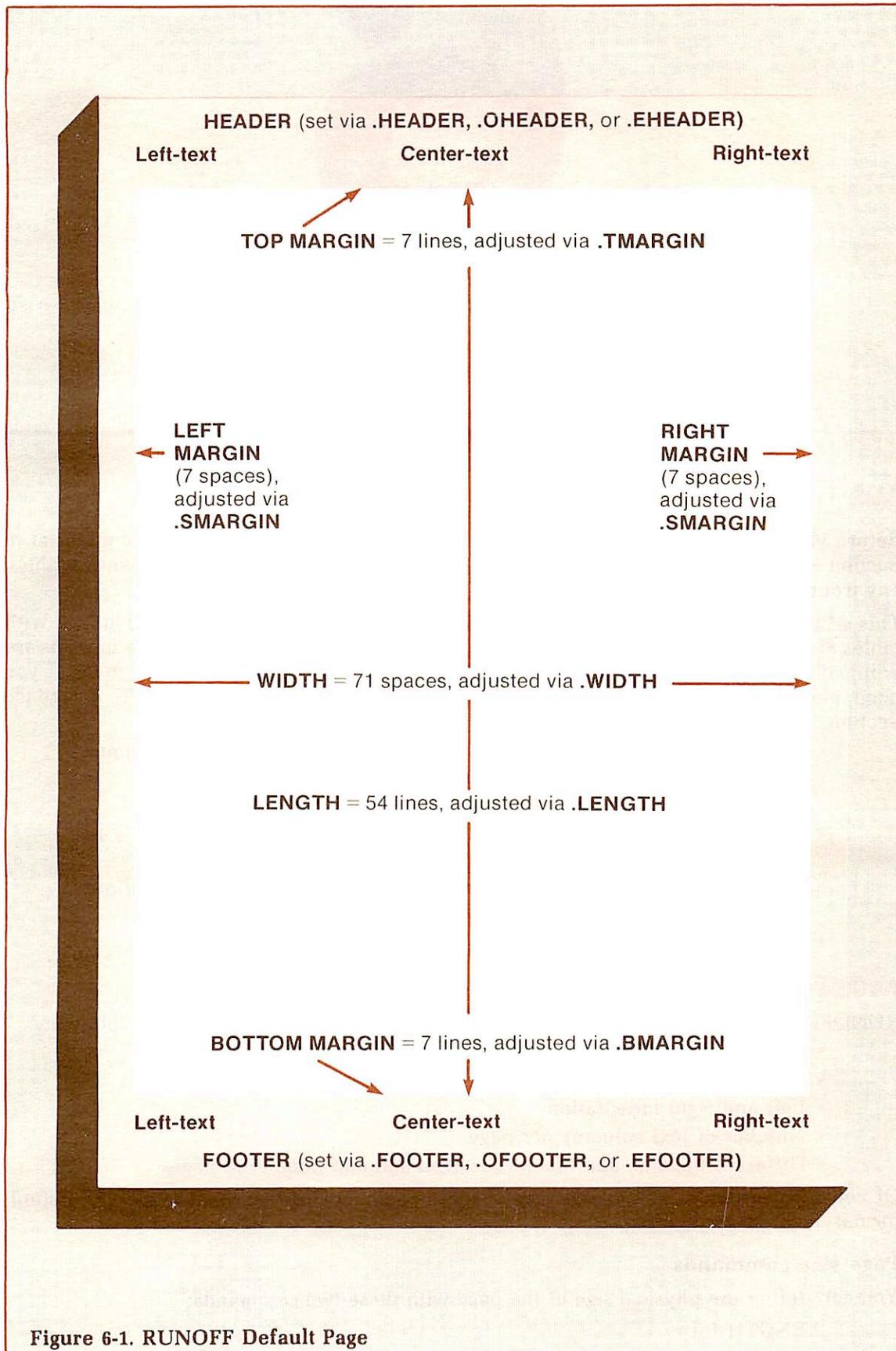


Figure 6-1. RUNOFF Default Page

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).

Remember, the WIDTH and LENGTH commands are defining the physical page size, not the number of columns or lines RUNOFF will print on that page. To create wider or longer text, add as many extra columns or lines you need per page to the default values of 85 and 66 respectively.

Both the WIDTH and LENGTH commands cause an implicit BREAK and EJECT. The default page size is 8 1/2 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)

### Indentation

The indentation command allows you to move the left and right margins individually and temporarily. Indentation is always relative to the most recent setting. (Remember that your tab settings will always be relative to the current left margin). The indentation 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 UNDEMENT 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 boasted,  
that, tho' he could not fiddle,  
he knew how to make a great city  
of a little one. The science  
that I, a modern simpleton, am  
about to communicate, is the very  
reverse.

ministers who have the management  
of extensive dominions, which  
from their very greatness are  
become troublesome to govern,  
because the multiplicity of their  
affairs leaves no time for  
fiddling.

I address myself to all

### 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, however all four delimiters must appear in a given command.
3. 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 a 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 section2
GO
RUNOFF REV 17.2
COMMANDS
$ from 2
$ to 4
$
PROCESSING...

OK,
```

### 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 continues 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 to see 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. The default value of *n* is one line.

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. 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 most of these reserved characters. RUNOFF's special characters, plus their defaults and meanings, are:

Name	Default	Meaning
BLANK	CONTROL-@	Required single 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.

In general, a symbol may be used to define only one action at a time. For example, the page number character (#) may not be defined as the BLANK character because it is already being used to define page numbers.

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 or you have changed them with PRIMOS's TERM command. 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. They have no special meaning when they occur as part of your source file. You may give the value-changing commands ERASE 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,
```

See Inserting Addresses on Form Letters in Section 8.

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. If a symbol definition is over 30 characters, RUNOFF will print a warning message and flag the error, but will continue to process text.
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: 95%% pure.

**Pre-defined Date Symbol-names:** RUNOFF has two special symbol-names, %DATE.1% and %DATE.2%, whose values are predefined. Each one is replaced by the current date, as known by PRIMOS.

%DATE.1% is replaced in the output file by the current date in the format mm dd yy. (For example, 10 25 84.)

%DATE.2% is replaced in the output file by the current date in the format Month Day, Year. (For example, September 12, 1984.)

Remember that RUNOFF distinguishes between upper and lower case letters in symbol-names. %Date.1% is not equivalent to %DATE.1%.

Also, remember that RUNOFF inserts the current date at the time the source file is processed. If you SPOOL the output file at a later date, the dates are not updated. For example, a file processed on September 12, 1984 will contain that date. If you spool that output file on November 3, 1984, the date inserted by %DATE.2% will still be September 12, 1984. To get an output file with the new date, you will have to RUNOFF the source file again.

### 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 process 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; pair of right braces ({} ) indicates the end.

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

### Note

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, footers, and apportioned text can be underlined by portion:

```
./{{text}}/text//
```

or completely - the entire line:

```
./{{text/text}}/
```

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

For example:

```
.header/{{Monthly Report/MAGRATHEAN 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.,

```
.blank &  
{{&&&&}}
```

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

```
{{        }} or {{&&&&}}
```

3. In either mode, use the underscore character (  ), not inside braces.

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.

The phantom hyphen character is set by the `.HYPHEN` command, described in Section 10. 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 does not format underlined text properly when you use RUNOFF's TTY command, then you should SLIST the output file instead. A separate line containing the underline characters will be printed below the text to be underlined.
2. On Diablo keyboards, the codes control-left-parenthesis and control-right-parenthesis generate the left and right braces, respectively.
3. If you have underlined a decimal heading (see Section 7), 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 `IXFILE` 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 encountered. 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 no specified `IXFILE`, RUNOFF asks for the name of the index file with the prompt:

```
FILENAME FOR INDEX:
```

You may give a filename, or else hit 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 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. See .PAUSE and .PERFORATE in Section 10 for further details.

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**

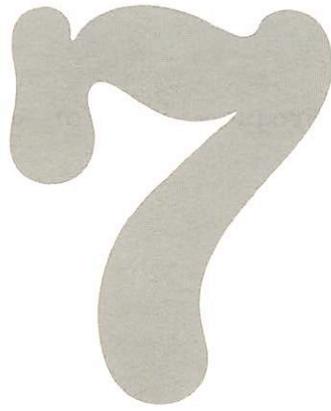
tells RUNOFF to suppress all error messages as well as the associated actions, and instead process the entire file. The output file will include the results of any errors that occur.

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.



# RUNOFF decimalization

## 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. Schedule
  - 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.

2 is a label for a first-level block of text  
2.1 is a label for a second-level block of text  
2.1.6 is a label for a third-level block of text  
2.1.6.2 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 on both 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 indentation 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.

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.

## 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 cooperation 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

## FORMAT PARAMETERS

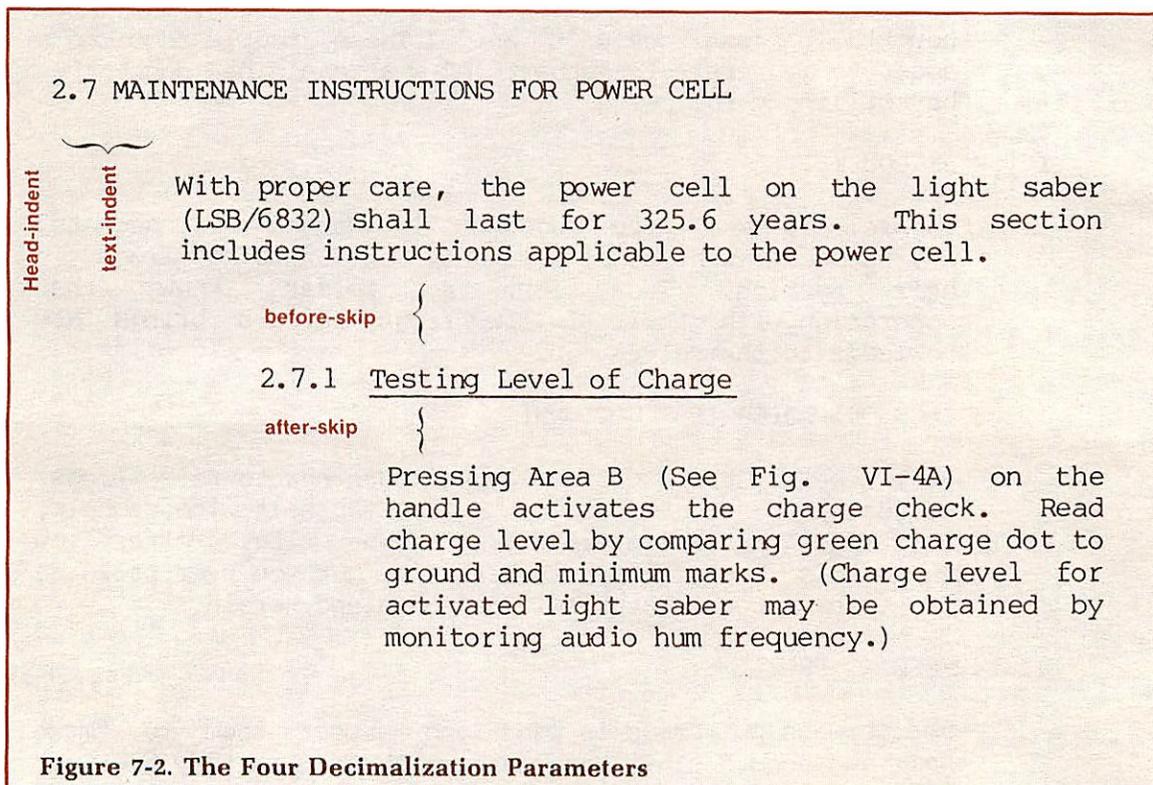
Format parameters are the values you can set for each level of decimal headings, controlling the indentations and line-skipping. Figure 7-2 illustrates the four parameters you can set, which are:

- **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)

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.

Figures 7-3 and 7-4 illustrate a page of text with default decimalization settings.



## LEVELS

In this section, we talk about "levels". Usually (but not always) a level appears as an amount of indenting. Figure 7-2 shows two levels: 2.7 (second level) and 2.7.1 (third level). We will talk about going "down" from second to third, and "up" from third to second level. Think of going "down" into the indentation, and coming back "up" to the left margin.

Notice that you go down from 2 to 3 and up from 3 to 2. This may not be the way you would expect things to work, but that's how RUNOFF's decimalization levels are defined.

```

.*This document contents a decimalization example using FROGS
.ds 0 1 1
.ds 4 1 0
.di 3 3 5
.h//FROG RAISING FOR PLEASURE AND PROFIT//
.sk
.dn HISTORY OF THE FROG INDUSTRY
.dd {{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:
.dd SKEPTICISM AND JEALOUSY
.dd {{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.
.dn {{Jealousy:}}
Frog raisers who were successfull guarded their secrets.
They seemed to want to be the only one raising frogs in their section.
Today most frog raisers know that cooperation with
their fellow frogs raisers brings NEW knowledge to
themselves too.
.du
.dn 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.
.dn 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?
.dn 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.
.du
.dn {{Discovery of Frog Meat As A Delicious Food}}
The frogs industry began when man discovered the delicious taste
of tender, white frogs legs. This happened centuries ago in
Europe. The story has it that a monk was walking in a
monastery garden when he espied a
greenfrog in the clutches of the unmerciful jaws
of a snake. The monk couldn't stand such a sight, so he dispatched a q

```

Figure 7-3. Runoff source 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 cooperation 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?

1.1.4 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 7-4. Default Decimalization Output

Notice also that “down” and “up” do not mean “down and up the page” but in and out levels of indentation.

## 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 { level-number } 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:

```
.dskip 1 5 2      .dskip 1 5 2
.dskip 2 5 2      .dskip * 5 2
.dskip 3 5 2      .dskip * 5 2      .dskip 0 5 2
.dskip 4 5 2      .dskip * 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 DINDENT command is:

```
.DINDENT level-number level-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 indentations 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 indentation 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 `.dnxt Mating Season` produces the label and heading:

2.3 Mating Season

The DDOWN command generates an 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
text...
.dup
.dnext
```

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 level 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:

$$\text{value} - 1 + 1 = \text{value}$$

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. For 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
.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:

```
.dnsup PORPOISE-HUMAN VOCODERS  
.ddsup 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 (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 UNDEXT or UNDEXT 0 will undent to the left margin setting for the current level instead of the page margin defined by SMARGIN.

Be sure to UNDEXT 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}}
```

becomes

```
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 files 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 a 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. For example:

```

1 INTRODUCTION.....1
  1.1 Overview.....1
  1.2 Terminology.....3
    1.2.1 Reserved Words.....4
2 GETTING STARTED PROCEDURES.....7
  2.1 System Commands.....13
3 EXPLANATION OF A HEADING WHICH IS TOO LONG
  LONG TO FIT ON ONE LINE.....17
  
```

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:

<b>.TMARGIN current-value</b>	Specify top margin
<b>.BMARGIN current-value</b>	Specify bottom margin
<b>.SMARGIN current-value</b>	Specify side margins
<b>.WIDTH current-value</b>	Specify paper width
<b>.LENGTH current-value</b>	Specify paper length
<b>.BLANK current-value</b>	Specify Special Blank Character
<b>.NFILL</b>	Enter NFILL mode
<b>.COLUMN 1</b>	Single Column
<b>.DSKIP 0 0 0</b>	Skip no lines before or after entry, except,
<b>.DSKIP 1 1 0</b>	Skip one line before level 1 entries.
<b>.DINDENT 0 0 3</b>	Indent 3 after all levels, except, level 1
<b>.DINDENT 1 0 2</b>	Indent 2 after level 1 entry
<b>.DINDENT 2 0 4</b>	Indent 4 after level 2 entry

<code>.DINDENT 3 0 6</code>	Indent 6 after level 3 entry
<code>.DLEVEL 1</code>	Go to decimal heading level 1
<code>.DRESET 1</code>	Reset to heading value 1
<code>.EJECT</code>	Eject to new page
<code>.SKIP 2</code>	Skip 2 lines
<code>. &gt; {{Table of Contents}}</code>	
<code>.SKIP 2</code>	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
.length 66
.
.
.
```

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; 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:

<b>.DDOWN [heading]</b>	Go down one level and generate a label.
<b>.DDSUPPRESS [heading]</b>	Go down one level but do not print a label.
<b>.DINDENT level label-indent [text-indent]</b>	Set indent increment for <b>level</b> to <b>label-indent</b> and <b>text-indent</b> spaces.
<b>.DLEVEL level</b>	Define the <b>level</b> at which numbering continues.
<b>.DLIMIT [limit]</b>	Reset maximum level of label to be entered in TOFC file.
<b>.DNEXT [heading]</b>	Generate a label (and <b>heading</b> ) on the current decimal level.
<b>.DNSUPPRESS [heading]</b>	Generate a <b>heading</b> on the current decimal level, but do not generate a label.
<b>.DRESET level value</b>	Change the current level number to <b>level</b> and set the next number on this level to <b>value</b> .
<b>.DSKIP level before-skip after-skip</b>	For each label on specified <b>level</b> , skip <b>before-skip</b> lines before the label and <b>after-skip</b> lines after it.
<b>.DUP [n-levels]</b>	Decrement the level number by <b>n-levels</b> .
<b>.TOFC filename [limit]</b>	Generate a table of contents in <b>filename</b> . If the <b>limit</b> 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.

**.TTOFC string**

Enter **string** in TOFC file.

# 8

## Sample sessions

---

### INTRODUCTION

This section explains and demonstrates the more advanced features of EDITOR and RUNOFF. The features covered are:

In RUNOFF

- Document formatting
- How to do hanging indents
- Form letters

In EDITOR

- Using GMODIFY
- Using MOVE
- Using XEQ

### 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 selected:

<b>SUBJCT</b>	For the name of the document
<b>DOCNO</b>	For the number of the document
<b>VERSNO</b>	For the number of the version or revision
<b>PUBDAT</b>	For the date of publication
<b>TITLE</b>	For the section title
<b>SECNO</b>	For the number of the section

The file GLOBAL.INFO contains the information pertinent to all sections of a given document. Use EDITOR to change the letters W, X, Y and Z to the appropriate values. You can also define one or more groups of standard tab stops here.

Here is one possible example of a GLOBAL.INFO file:

```
. *GLOBAL.INFO--contains general formatting information
.define SUBJCT FROGS
.define DOCNO 2001
.define VERSNO REV 3
```

```
.define PUBDAT 10/12/80
.*FIVES, TENS, and TABLE1 are for convenience in setting tabstops.
.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 INSERT it.

```
.*SECTION.INFO--sets up section format
.insert GLOBAL.INFO
.eheader/SECTION %SECNO%/SUBJECT//
.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 indentation at the beginning of paragraphs. Positive indentation is the more traditional — the first word of text is indented **m** spaces to the right of the current left margin, like this:

```
    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.
```

DOCU 2001

Introduction

## SECTION 1

## Introduction

## 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 cooperation 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?

## 1.1.4 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.

Page 1 - 1

Oct. 12, 1980

Figure 8-1. Page Formatted by GLOBAL.INFO

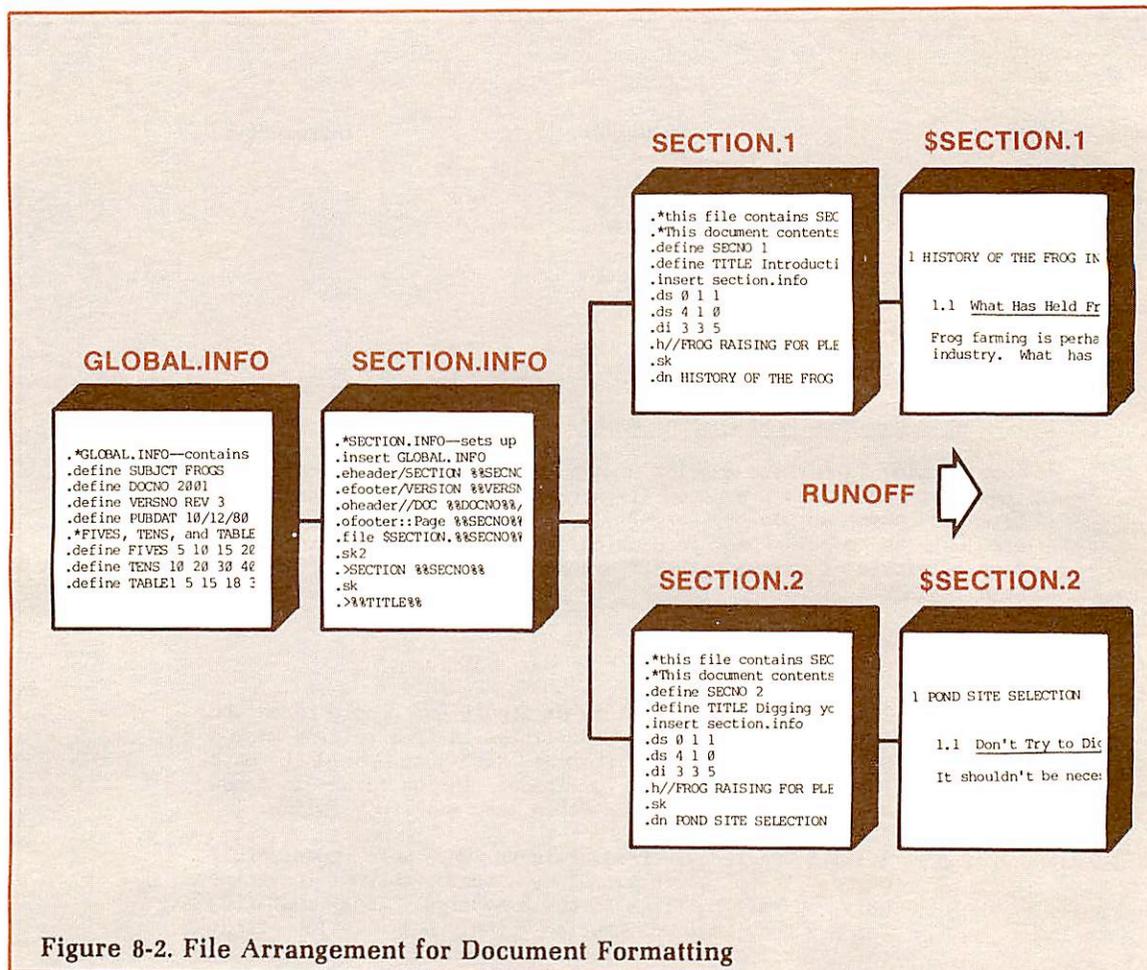


Figure 8-2. File Arrangement for Document Formatting

Negative, or hanging, indentation 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 indentation, 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.)
2. Set the negative indentation 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:

```
. *This file demonstrates a list with hanging indents
.wi 55
.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 indentation 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 TEMPERATURE CHANGES	Cold water retards the development of and growth 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. 01914  
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 away 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
. footer// "I never Ate A Frog I Didn't Like"//
@%DATE.2%
. sk
%SYMB-1%
%SYMB-2%
%SYMB-3%
. sk2
Dear %SYMB-1%,
. fill
. p 5 l
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
. 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:

```

.*This file shows how to change addresses, one at a time
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
n
5507 Burney Blvd.
c//.def SYMB-2 /
.def SYMB-2 5507 Burney Blvd.
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 .inser 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). See **Asterisk Constructions** below.

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 Hari Seldon
.def SYMB-2 2 Foundation Place
.def SYMB-3 Trantor, 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 address, 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 RUN-OFF's PAUSE command at the beginning so you can insert paper for each letter.

Figure 8-3 shows what your output will look like. Figure 8-4 shows the arrangement of files for doing form letters.

### ASTERISK CONSTRUCTIONS

Use the asterisk (\*) to repeat an EDITOR command line. If you include the FIND or LOCATE commands in your command line, you can cause a command to be applied to selected lines in a file. For example, if you had a file containing names and addresses, and several of the zip codes were missing for New York, NY, you could:

```
f New York;a xx;c/NYxx/NY 10000/;c/xx//;*
```

This finds lines beginning with "New York". On those lines, it appends "xx". If the line has no ZIP code after "NY", then "NYxx" (at the end of the line) is changed to "NY 10000". If the line already has a ZIP, the "xx" is removed. The \* repeats the command for all lines in the file. For lines not beginning with "New York", the command makes no changes.

If you want a command repeated for only the next several lines, but not to the end of the file, follow the asterisk with a number parameter. For example,

```
point 37
locate Vienna;c/Australia/Austria/;*88
```



## The Raniburger Corporation

April 9, 1980

Arthur Trent  
13 Diablo Drive  
Wallingford, Pa.19380

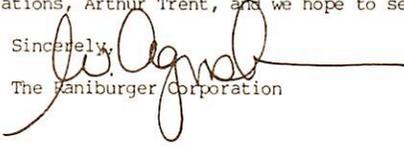
Dear Arthur Trent,

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"

666 Ouroboros Drive, Arkham, MA 02546

Figure 8-3. Form Letter Output

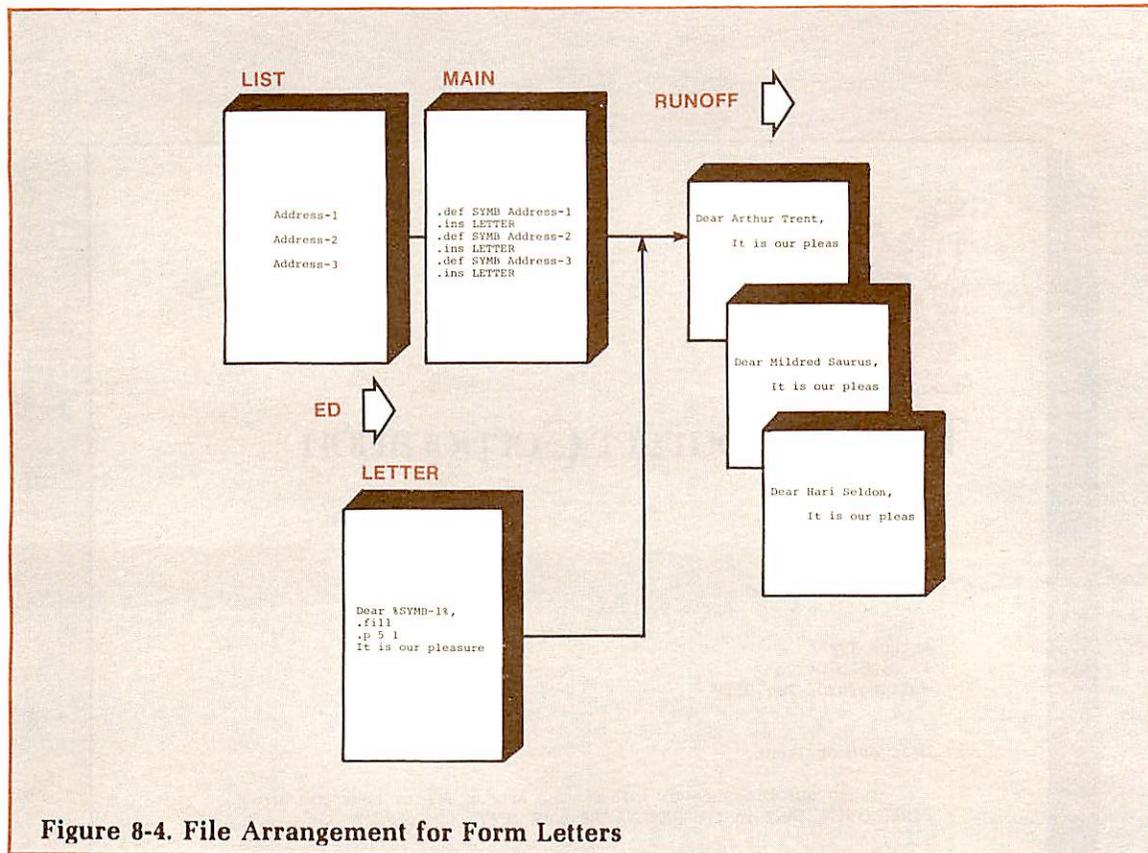


Figure 8-4. File Arrangement for Form Letters

changes "Australia" to "Austria" in line 37 and any of the 88 following lines in which Vienna appears.

Without a FIND or LOCATE, your command will affect every line. For example, this command encloses all lines in parentheses:

```
c//(;/a );n;*
```

### THE GMODIFY COMMAND

The GMODIFY command edits a line of text on a character-by-character (column-by-column) basis. It can perform complicated editing operations which would be difficult or impossible with the other EDITOR commands. For example, you can change

```

Ackerwyth, F.J.: 213-555-4244
Garvin, W.: 200-555-9950
Mason, Perry: 955-3438
Poirot, Hercule: 411-9922

```

into

```
213-555-4244: Ackerwyth, F.J.
200-555-9950: Garvin, W.
955-3438: Mason, Perry
411-9922: Poirot, Hercule
```

with the single command

```
g d:e2fsi/: /c:/;n;*
```

Just as EDITOR assigns line numbers to each line of your file starting at the first line, GMODIFY assigns column numbers to each character on a line, starting at the left. GMODIFY uses a column pointer to move across a line and find the character or characters you request. Think of it as pointing between characters, not at them.

GMODIFY works by putting the current line into a temporary storage area and then building a new current line by moving a column pointer across the stored line as directed by your subcommands, and moving your requested characters into the new current line. It discards the stored line when it is done. Using subcomands, GMODIFY will do the following tasks:

- Take a specified number of characters from the stored line and copy them into the new current line.
- Copy all characters from the stored line into the new current line, starting from the current column position, and continuing until it finds a specified character.
- Move the column pointer along the stored line in either direction, without copying characters into the new current line, either to a specified column or until a specified character is found.
- Insert new text strings into the new current line.

EDITOR cannot perform the GMODIFY command on a .NULL. line. If you try to GMODIFY a .NULL. line, you will get the message

```
.NULL.
BAD GMODIF
```

Each subcommand consists of a single letter which 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 new current line as specified.

One GMODIFY command may contain several subcommands. Subcommands may be separated by any number of spaces (or no spaces), but may not be separated by commas. Subcommands must be separated from the GMODIFY command by at least one space.

In the following explanation of GMODIFY's subcommands, the **c** represents a character in the stored line, **n** stands for a number of columns. The slash character (/) represents a delimiter for a string — any character not contained in the string may be used as the delimiter except for an erase or kill character, or a carriage return.

GMODIFY's subcommands are:

Subcommand	Meaning
Cc	Copies into the new line from the stored line, starting at the current column, all characters up to (but not including) the <b>c</b> parameter. The column pointer moves to the immediate left of

	the column containing <b>c</b> . If <b>c</b> is not found, all characters to the end of the stored line are copied.
<b>Mn</b>	Copies <b>n</b> characters from the stored line into the new current line. The column pointer moves <b>n</b> characters to the right of where it began.
<b>Dc</b>	Moves pointer, without copying, across the stored line up to, but not including, the <b>c</b> parameter. If <b>c</b> is not found, the pointer moves to the end of the stored line; this has the effect of deleting all characters to the right of where the pointer began.
<b>En</b>	Moves pointer <b>n</b> positions to the right in the stored line, without copying those <b>n</b> characters into the new current line.
<b>Bn</b>	Starting at the current column, this moves the pointer <b>n</b> spaces to the left in the stored line (but not past the beginning of the line).
<b>S</b>	Repositions pointer to the beginning of the stored line.
<b>F</b>	Starting at the current column of the stored line, copy all remaining characters into the new current line. If you do not use the <b>F</b> subcommand, all remaining parts of the stored line which were not copied into the new line will be deleted.
<b>I/string/</b>	Insert <b>string</b> into the new current line.
<b>A/string/</b>	Starting from the current column, copy all remaining characters from the stored line into the new current line, then append <b>string</b> . Equivalent to <b>FI/string/</b> .
<b>R/string/</b>	<b>String</b> is copied into the current line. The pointer is moved ahead on position in the stored line for every character in <b>string</b> , thus "replacing" that portion of the stored line with <b>string</b> .
<b>O/string/</b>	The same as <b>R/string/</b> , except that a space character in <b>string</b> causes the corresponding character in the stored line to be copied into the new current line. The wild symbol (!) will create an actual space in the line. The effect is to copy a portion of the stored string into the new current line and then to "overlay" <b>string</b> onto it.
<b>N</b>	Reverses the sense of the <b>c</b> parameter in the next <b>C</b> or <b>D</b> command, to mean "... up to next character not matching <b>c</b> ". <b>N</b> may be separated from the <b>C</b> or <b>D</b> by other subcommands, but affects only the <b>C</b> or <b>D</b> , not the others.

Here are a few examples using GMODIFY:

```

OK, ed bells
GO
EDIT
print 2
.NULL.
bells bells bells bells
g m18 i/gongs/f
bells bells bells gongsbells
g m23 o!/bells/
bells bells bells gongs bells
g e18 cb b18 f
gongs bells bells gongs bells
g db cb r/gongs / e6 m6 s m6 i/gongs/
gongs gongs gongs gongs
file

```

```

OK, ed verb.list
EDIT
p23
.NULL.
reading
writing
editing
reeling
writhing
BOTTOM
t, n
reading
g fi/ /sfi/;/n;*
reading reading:
writing
writing writing:
editing
editing editing:
reeling
reeling reeling:
writhing
writhing writhing:
BOTTOM
t
c/ing:/er/99
reading reader
writing writer
editing editor
reeling reeler
writhing writher
BOTTOM
file

OK,

```

## THE MOVE AND XEQ COMMANDS

These commands are of limited value to most users. They can be useful to those developing complicated EDITOR macros. Most repetitive editing actions are best done with **Asterisk Constructions**, described above.

### String Buffers

The EDITOR has several special storage areas, called string buffers, each of which can hold one line of text. They are:

- STR.1, STR.2, STR.3, STR.4, STR.5, STR.6, STR.7, STR.8, STR.9, and STR.10 are available for you to use as storage areas. (STR.1, STR.2, and STR.3 are also known as STRA, STRB, and STRC.)
- EDLIN is the command line currently being executed.
- INLIN is the current text line in your workfile.

Upper and lower case letters are equivalent in string buffer names.

The primary use for the string buffers is in building complicated commands which can be executed repeatedly, using the XEQ command. The MOVE and XEQ commands, described below, are the only EDITOR commands which use the string buffers.

**Using MOVE**

Use the MOVE command to put a line of text or commands into a string buffer. The command format is:

```
MOVE buffer-2 buffer-1
```

or

```
MOVE buffer-2 /string/
```

A line of text (or commands) is moved into `buffer-2` from `buffer-1` or from `/string/`. The delimiters on string (shown here as slashes) may be any non-alphabetic character which does not appear in string, including comma and semicolon.

Examples:

<code>MOVE INLIN STR.1</code>	Replaces the current line with the contents of STR.1
<code>MOVE STR.1 :c/iron/gold/:</code>	Puts "c/iron/gold/" into STR.1 (for use by XEQ, below).

Note that there are two sets of delimiters in the string in the second example above. The colons enclose the entire string, the slashes are used as delimiters for the change command. The MOVE command is necessary to the XEQ command described below.

**XEQ command**

Once you have MOVED a command string into a string buffer, you can execute it via EDITOR's XEQ command. For example:

```
OK, ed fair
EDIT
p2
.NULL.
fair is foul and foul is fair
move str.1 /c:foul:fxxx:g;c:fair:foul:g;c:fxxx:fair:g/
xeq str.1
fair is fxxx and fxxx is fair
foul is fxxx and fxxx is foul
foul is fair and fair is foul
xeq str.1
fxxx is fair and fair is fxxx
fxxx is foul and foul is fxxx
fair is foul and foul is fair
file

OK,
```

# 9

## The EDITOR reference section

---

### 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 **PPEND: 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, appEND**.

#### Parameters

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

- The word **filename**, representing a filename or a file's pathname.
- 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... henceforth
find... henceforth
```

would be using the string `SS` 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 only if the semicolon character has been freed for use as a text character via the SYMBOL command. Semicolons must otherwise 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, NFIND.

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**.

It distinguishes between upper-case and lower-case letters. If string-1 is not found, no change is made.

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. A space between CHANGE and the delimiter is optional, unless the delimiter is a letter or space character, in which case the space is required.

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 **n**th 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 — thus, if you used "@" every place in your file where you wanted to use ";", 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 newline; 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 or D -1 will all delete only the current line.

**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** anywhere within it. 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, including proper case of letters, 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, containing the current line.

DUNLOAD TO leaves the pointer positioned at the line containing string, which was not deleted. It prints out this line.

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 accidentally 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 deletes 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 the escape character (default is uparrow (^)).

To check the current value of the erase and escape characters 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. If you do specify a filename, EDITOR writes into a file being edited, and prints its name. 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, but instead want to retain your original version, unchanged, type QUIT instead of FILE. If you made any 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.
4. Rules for making filenames
  - A. Filenames can be up to thirty-two characters long.
  - B. Filenames can contain only the following characters: A through Z, 0 through 9, - & # \$ \* \_ . /. Characters not permitted in filenames include: imbedded blanks (spaces), special characters such as: , ? ! @ ;
  - C. The first character may be any legal character except a digit.
  - 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. You may use the wild symbol (!) and match-n-spaces (#) symbols 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. There must not be a space between FIND and {.

**▶ 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 comas.

See Section 8, SAMPLE SESSIONS, for complete information on the GMODIFY command.

**▶ IB newline**

The IB (insert before) command is used exactly like the INSERT command, except that **newline** is inserted before the current line, rather than following it.

**▶ INPUT**  $\left\{ \begin{array}{l} \text{(ASR)} \\ \text{(PTR)} \\ \text{(TTY)} \end{array} \right\}$ 

The INPUT command reads text from the specified input devices:

- ASR — Teletypewriter 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. PTR must be ASSIGNED before use.

**▶ 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 a semicolon in newline unless the semicolon is preceded by the escape character (normally ^). A semicolon or RETURN will signal the end of the line being inserted. Use the PSYMBOL command to inspect the current ESCAPE and SEMICO characters, and the SYMBOL command if you wish to change them.

**▶ 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 PSYMBOL 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) or less than 10, but within those limits it can be set or reset as much as desired. The parameter **n** is the number of characters in the line. If EDLIN (the command line), INLIN (the input line), or STR.1 through STR.10 (the string buffers) contain more characters than the new maximum, they are 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. The remainder of the typed line 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 1024 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 spaces will be considered part of string. You may use the wild symbol (!) and match-n-spaces (#) symbols in string.

### ▶ **MODE {CKPAR } {NCKPAR}**

The command MODE CKPAR causes EDITOR to print as ^nnn any characters which have a parity bit of 0. (Prime's standard is that the parity bit is 1.) In MODE NCKPAR (the default) characters are printed normally, regardless of the parity bit.

### ▶ **MODE COUNT [start] [increment] [width] {PRINT BLANK SUPPRESS}** **MODE NCOUNT**

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 ≠ 0 (default = 1)
- **width** — field width (number of digits) 1 < width < 10 (default = 5)
- **PRINT** — Print leading zeroes (default)
- **SUPPRESS** — Do **not** print leading zeroes
- **BLANK** — Replace leading zeroes with blanks

The counter symbol 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 {NUMBER } {NNUMBER}**

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** {**COLUMN** }  
          {**NCOLUMN**}

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** {**PROMPT** }  
          {**NPROMPT**}

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** {**PRALL** }  
          {**PRUPPER** }  
          {**PRLOWER**}

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. For example, the text:

```
  This line begins with UPPER case,
  while this one does not.
```

under **MODE PRUPPER** appears (or may be input) as:

```
  T^LTHIS LINE BEGINS WITH ^UUPPER ^LCASE,
  ^LWHILE THIS ONE DOES NOT.
```

and under **MODE PRLOWER** as:

```
  ^UT^LTHIS LINE BEGINS WITH ^UUPPER ^LCASE,
  WHILE THIS ONE DOES NOT.
```

▶ **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-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 its 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 **n**th line. If there are fewer than **n** lines in the file below the current line, the pointer will be left below the last line.

1. Remember to go to the **TOP** before **MODIFY**ing 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 with the delimiters — e.g., if you used "@" every place in your file where you wanted to use ";", then the command sequence

```
MODIFY/@/;/G 99999
```

would change all the @'s to ;'s. (Make sure **n** is greater than the number of lines in your file.)

► **MOVE** **buffer-2** { **buffer-1** }  
                          { **/string/** }

The **MOVE** command moves one line of text into **buffer-2** from **buffer-1** 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**)
- **STR.4** through **STR.10** (no synonyms)

The delimiters around **string** can be any non-alphabetic character including comma (,) or semicolon (;) which does not appear in **string**. If **MOVE** is the last command on the line, the closing delimiter may be omitted.

The command: **MOVE** **buffer** // clears **buffer**. See Section 8 for more information.

► **NEXT** [**n**]

The **NEXT** command moves the pointer **n** lines. Positive values of **n** move the pointer toward the bottom, negative values toward 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**, **TOP** or **BOTTOM** will be printed 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.

**NFIND** is the exact opposite of **FIND** — it skips over lines which begin with **string**, and stops at the first line which doesn't.

If **NFIND** can't find a line which doesn't begin with **string**, the pointer is left at **BOTTOM**.

Put exactly one space between the **NFIND** command and **string**. For example, **NFIND S S S AUTOMOBILES** looks for the first line not beginning with "S S AUTOMOBILES".

Like **FIND**, you can **NFIND** beginning on a column other than column 1:

```
NFIND(n) string
```

Remember: There must be no spaces between NFOUND and (n), and exactly one space between (n) and string. The parentheses ( ) are required.

### ▶ **NLOCATE string**

The NLOCATE command finds the first line below the current line which does not contain **string**, and makes that line the new current line.

NLOCATE is the exact opposite of LOCATE—it skips over lines containing **string**, and stops at the first line which doesn't.

If NLOCATE can't find a line which doesn't contain **string**, the pointer is left at BOTTOM.

The first space after NLOCATE separates string from the command word. All other spaces will be considered part of **string**.

### ▶ **OOPS**

The OOPS command restores the most recently changed line to its condition just before changing. If your last command changed several lines, OOPS will restore only the last line to have been modified. The most recently changed line must be the current line for OOPS to work. A second OOPS will not undo a previous OOPS.

### ▶ **OUTPUT**      $\left\{ \begin{array}{l} \text{(DISPLAY)} \\ \text{(TTY)} \end{array} \right\}$

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 those described below will replace the character in the corresponding column of the current line. (Note: ^, and ^? are each considered to be a single text character.)
2. A space ( ) in string leaves the original character unchanged.
3. An exclamation (!) in string forces a space in the corresponding column.
4. The tab (/) causes tabbing to the next tab stop.
5. ERASE, KILL and SEMICO (normally",?, 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 issuing a FILE command. 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 greater than 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.

### ▶ PPRINT [*first*] [*last*]

The PPRINT command prints a range of lines relative to the current line without changing the position of the current line. For example,

```
PPRINT -3 12
```

prints from three lines before the current line to 12 lines past the current line.

PPRINT with no numbers is the same as

```
PPRINT -5 5
```

printing from five lines above to five lines below the current line.

If you only specify one number, and it is negative, PPRINT sets **first** to that number, and **last** to zero. For example,

```
PPRINT -8
```

prints from eight lines back, up to the current line.

If you only specify one number, and it is positive, PPRINT sets **last** to that number, and **first** to zero. For Example.

```
PPRINT 7
```

prints from the current line up to seven lines forward.

### ▶ 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.

### ▶ PSYMBOL

The PSYMBOL command displays the current value of EDITOR's symbols. See the SYMBOL command for a list of symbols, their meanings, and their default values.

### ▶ PTABSET *ptab-1* *ptab-2* ...

The PTABSET command tells EDITOR to use your terminal's physical tab stops when printing output. For example, if your terminal has built-in tabs every 8 spaces as many do,

```
PTABSET 8 16 24 32 40 48 56 64 72
```

will enable EDITOR to reduce the number of spaces it needs to send to your terminal when printing long lines with many spaces.

▶ **PUNCH**       $\left\{ \begin{array}{l} \text{(ASR)} \\ \text{(PTP)} \end{array} \right\} [n]$

The PUNCH command punches *n* lines on the specified paper tape punch:

- **ASR** — Teletypewriter punch
- **PTP** — High speed punch

The opening parenthesis is required. A PUNCH command with no argument assumes the PTP device. The device PUNCH must be ASSIGNED before use.

Punching on the ASR must be done in BRIEF mode.

▶ **QF**

The QF command (quit final) is the same as the QUIT command, except that it quits without question even if the file is modified.

▶ **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, typing 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.

▶ **SAVE filename**

The SAVE command allows a file to be written out without leaving the EDITOR. If **filename** is not specified, EDITOR saves into the file being edited and prints its name.

▶ **SYMBOL name character**

The SYMBOL command changes the character value of a special symbol name to the new character. The special symbols, their initial (system default) values, and their purposes are:

<b>Special Symbol Name</b>	<b>Character</b>	<b>Purpose</b>
<b>KILL</b>	?	Delete current line
<b>ERASE</b>	"	Delete previous character
<b>WILD</b>	!	Match any character
<b>BLANKS</b>	#	Match-n-spaces
<b>TAB</b>	/	Spaces to next tab stop
<b>ESCAPE</b>	^	Remove special meaning from next character
<b>CPROMP</b>	\$	MODE PROMPT's EDIT prompt
		FIND, NFIND, LOCATE

DPROMP	&	MODE PROMPT's INPUT prompt
COUNTE	@	MODE COUNT's counter symbol
SEMICO	;	End-of-line separator

You cannot use any of the following as a special symbol character.

- Multiple characters (e.g., ABC)
- comma (,)
- space ( )
- asterisk (\*)
- any character currently in use as a Special Symbol except the CPROMP or DPROMP

You can check the current value of your symbols with the PSYMBOL command.

```
OK, ed
INPUT

EDIT
psymbol
KILL ?
ERASE "
BLANKS #
WILD !
TAB \
ESCAPE ^
CPROMP $
DPROMP &
COUNTE @
SEMICO ;
symbol kill {
symbol tab *
BAD SYMBOL
symbol escape %
symbol wild ~
psymbol
KILL {
ERASE "
BLANKS #
WILD ~
TAB \
ESCAPE %
CPROMP $
DPROMP &
COUNTE @
SEMICO ;
q
```

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.

### ▶ **TABSET tab-1 tab-2 tab-3 ... tab-8**

The TABSET command allows you to specify tab settings for use with the backslash symbol (\). Default tab settings are 6, 12, 30.

```
tab 5 10 15 20
```

```
INPUT
0\5\10\15\20
```

will process to:

```
0 5 10 15 20
```

Use the MODIFY or OVERLAY commands to edit already existing tabulated text:

```
print 2
.NULL.
637-8687      637-12344
o\637-1234
637-8687      637-1234
```

You must set your tab values correctly each time you edit the file to input additional data.

### ▶ **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 being EDITed into a new file named **filename**. If **n** is 0 or omitted, it is assumed 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)
- **STR.4** through **STR.10**

Use the MOVE command to set up commands in string buffers.

XEQ with no buffer name re-executes the previous command line. It is possible to nest XEQ commands so that commands in string buffers can be executed as subroutines.

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. See **Asterisk Constructions** in Section 8 for examples.

# 10

## The RUNOFF reference section

---

### 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 possibly one or more **parameters**.

#### 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 should 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(s) shown in rust in a command word indicate the minimum acceptable abbreviation.

#### 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. It also has a special use with the **.FLOAT** command.

### ▶ **.+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 “greater-than” symbol (>) 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/**

Apportions the text as **left**-justified, **centered**, 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.

### ▶ **.ADJUST**

The ADJUST command 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. Blanks may be used to indicate words which must not be separated by extra spaces. These blanks will be neither suppressed nor padded during FILLing and ADJUSTing.

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.

If you want a combination of single and multi-column material on a given page, you must:

1. RUNOFF the different section as a distinct 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.)

▶ **.DDSUP [heading]**

Go down one level but suppress (do not print) label. (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 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.

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

The DINDENT command specifies the amount to indent labels and text from the current left margin. (Decimalization commands.)

▶ **.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 level. (Decimalization command.)

▶ **.DNSUP [heading]**

Generate a **heading** on the current decimal level, but suppress (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.)

▶ **.EEVEN**

The EEVEN command (Eject Even) will cause an EJECT to the next even numbered page. If the next page is odd, the odd page remains blank except for headers and footers (if any). The page number used will be that displayed by # in the header or footer. If PAGEN (#) is not specified, the physical page number will be used.

▶ **.EFOOTER/left-text/center-text/right-text/**

The EFOOTER command sets up the footer on even-numbered pages. Any character which is not used in the text 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 which is not used in the text 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***right-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 implicit EJECT when given; this prevents pages from having contrasting text format (even if you want it). See the COLUMNS command for a way around this limitation.

Two EJECT commands will not be interpreted as a request for a blank page in your output. To create a blank page, use a blank character

▶ **.EODD**

The EODD command (Eject Odd) will cause an EJECT to the next odd numbered page. If the next page is even, the even page remains blank except for headers and footers (if any). The page number will be that displayed by # in the header or footer. If PAGEN (#) is not specified, the actual page number will be used.

▶ **.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 prompts and error messages 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 specifies 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. Each FILE command is effective on all text which follows it, up to the next FILE command or (if there is none) to the end of the source file.

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

#### Note

If RUNOFF asks you to "enter output file treename" even though you have supplied a FILE command, it means that text appears before the FILE command. Even a single blank line anywhere before the first FILE command will cause this error.

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.

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). (This is the one instance when `.*` means more than just a comment.) 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^377lishment
```

would run over the right margin, RUNOFF would process the word as follows.

```
text 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 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 UNDEMENT 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%, ..., %9, in corresponding order. Unassigned symbol-names have. NULL. values. In other words, the command:

```
.insert MEMO (Jan05, 1984, Sales, 503, 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
.define 9
.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,noisy)
.insert MEMO (Mar15,1984,Acctg,666,Stock,Report)
```

There is an additional format for the INSERT command:

```
.INSERT [unit] [(symbol-1, symbol-2, ... symbol-n)]
```

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 from the source file 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 INDEX 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 = 1 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.

▶ **.NFILE**

The NFILE command tells RUNOFF that no output file is to be written.

▶ **.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 for 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 commands 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, then indents the first line of a new paragraph **m** from the current left margin after skipping **n** lines.

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, or by including a blank line. This is known as implicit paragraphing.

The command PARAGRAPH 0 0 is equivalent to BREAK.

For information on "hanging indents" (negative indentation), see Section 8. **Sample Sessions**

▶ **.PAUSE { [1] }  
          { 0 }**

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:

- On printing terminals, to print each page of RUNOFF output on a separate piece of paper.
- On CRT terminals, to 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 for the character.

### ▶ **.PERFORATE [n]**

The PERFORATE command prints a line of hyphens as perforation marks between pages, on the terminal and in the output file.

The command .PERFORATE 1 causes the perforation line to consist of only a hyphen at each end of the line. A subsequent .PERFORATE (with no number) restores regular perforation.

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 command 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 CONTROL-P (or Break) key on your terminal has exactly the same command as the QUIT command in command mode. (In command mode it does not do an EJECT.) 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, or with

name — e.g., RBAR OFF, RBAR XXX, RBAR. In ADJUST mode, if the revision bar is turned off in the middle of a line, the line is marked by a bar. RBAR ON does not put a revision bar next to any blank line.

The command `.RBAR ALL` is the same as RBAR ON, except that it puts revision bars on blank lines in the revised section.

### ▶ **.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., times line spacing. (If double-spacing, skip 2n lines, etc.) SKIP causes an implicit BREAK. If **n** is omitted, it defaults to 1.

RUNOFF will disregard a skip command as the first line of a new page since it has already “skipped” a number of lines to get to the top of a page. To skip line as the first line of text on a new page, use:

```
.blank &  
&  
.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 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, by 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 line 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 file line numbering this returns to the sequence in use before 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 is 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 (%).

### ▶ **.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.

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** is 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]**

Generates 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, turns 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 UNDEMENT 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.

# Symbols, etc.

\ EDITOR tab character 1-9,  
3-18  
" entering into EDITOR text 3-2  
# RUNOFF page number 5-4,  
10-10  
% in RUNOFF 6-7  
\* comment in RUNOFF 5-8  
\* repeat EDITOR command 8-10  
+ verbatim in RUNOFF 10-2  
•NULL• 3-5  
/// apportion in RUNOFF 5-8  
; entering into EDITOR text 3-14  
> center text in RUNOFF 5-8,  
10-2  
? entering with EDITOR 3-2  
? error 3-7

## A

Acoustic coupler 2-3  
Adding to the end of a line 3-12  
ADJUST 10-2  
ADJUST mode 5-6  
APPEND 3-12, 3-6, 9-2  
Apportioning text 5-8, 10-2  
Artwork 6-6, 10-11  
Asterisk constructions in  
EDITOR 8-10, 9-14  
Asterisk in RUNOFF 10-2  
ATTACH 4-1, 4-3

## B

Backslash \ 1-9  
Backslash used for tab 3-18  
BAD error 3-7  
BLANK 10-2  
BLANK character (RUNOFF)  
6-7  
Blank lines 5-9  
Blocks of text 6-6  
BMARGIN 6-3, 10-2  
BOTTOM 3-9, 9-2  
Bottom margin 6-3  
Braces on Diablo printer 6-11  
Braces { } 6-11  
BREAK 5-9, 10-2  
Break, implicit 5-9  
BRIEF 9-2

## C

Cancelling spooled files 4-6  
Caret (^) 1-9

Centering text 5-8, 10-2  
CHANGE 3-13, 9-3  
Changing directories 4-1  
Changing names of files and  
sub-UFDs 4-8  
Changing text 3-12  
Character 1-7  
Character parameters 3-3  
CLOSE 4-8  
Closing open files 4-8  
CMARGIN 6-3, 10-2  
CNAME 4-8  
COLUMNS 6-4, 10-3  
Columns of text 6-4  
Comma 3-6  
Command 1-5  
Command format in EDITOR  
3-2  
Command parameters 3-2  
Command words 1-6, 3-2  
Commands in EDITOR 3-6  
Comments in RUNOFF source  
5-8, 10-2  
CONTROL key 1-9  
CONTROL-Q 4-5  
CONTROL-S 4-5  
Conventions in EDITOR 3-2  
Conventions in examples 1-8  
Copying files 4-9  
CREATE 4-3  
CTRL key 1-9  
Curly-braces { } 6-11

## D

Data 1-4  
DATE 4-8  
Date symbols 6-9  
DDOWN 7-8, 10-3  
DDSUP 10-3  
DDSUPPRESS 7-10  
Decimalization 7-1  
Decimalization command  
summary 7-13  
Decimalization levels 7-4  
Default 1-5  
Default values in EDITOR 3-2  
Default values of parameters 1-6  
DEFINE 6-8, 10-3  
DEL key 1-9  
DELETE 3-14, 4-8, 9-3  
DELETE TO 9-3  
Deleting files and sub-UFDs 4-8  
Dialog 1-4  
DINDENT 7-7, 10-3  
Directories 2-5  
Displaying files on terminal 4-5

Displaying RUNOFF output on  
terminal 5-11  
DLEVEL 7-9, 10-3  
DLIMIT 7-13, 10-3  
DNEXT 7-8, 10-3  
DNSUP 10-3  
DNSUPPRESS 7-10  
Document formatting 8-1  
Double quote (") 3-2  
Double-spacing of lines 5-7  
DRESET 7-9, 10-3  
DSKIP 7-7, 10-4  
DUNLOAD 9-4  
DUNLOAD TO 9-4  
DUP 7-8, 10-4

## E

EDIT mode of EDITOR 3-3, 3-5,  
3-6  
Editing a new file 3-3  
Editing an existing file 3-4  
EDITOR commands, repetition  
of 8-10  
EDITOR:  
basic commands 3-7  
command format 3-2, 9-1  
commands 3-6  
conventions 3-2  
error messages 3-7  
line changing commands 3-12  
tab character 3-18  
tab stops 3-18  
work file 3-3  
EDLIN 8-15  
EEVEN 10-4  
EFOOTER 6-4, 10-4  
EHEADER 6-4, 10-4  
EJECT 5-11, 10-4  
Eject, implicit 5-11  
End of page 5-11  
Entering text into EDITOR 3-3,  
3-4  
EODD 10-4  
ER! 1-6  
ERASE 9-4, 10-4  
Erase character 2-1, 3-2  
ERASE character (RUNOFF) 6-7  
ERRGO 6-12, 10-5  
Error messages 1-6  
Error messages from EDITOR  
3-7  
Error messages, suppressing  
6-12  
Errors in RUNOFF 5-15  
Even and odd pages 6-4, 10-4,  
10-9  
Examples 8-1  
Examples, conventions in 1-8  
Exiting EDITOR 3-15, 3-16

## F

Figure insertion 6-6, 10-5, 10-11  
File 1-4  
FILE (in EDITOR) 3-16, 9-5  
FILE (in RUNOFF) 6-5, 10-5  
File directories 2-5  
Filename 1-7  
Filename parameters 3-3  
Filenames, rules for 3-17  
Files 2-5  
Files:  
  deleting 4-8  
  renaming 4-8  
FILL 10-5  
FILL mode 5-6  
FIND 3-11, 9-5  
FIND(n) 9-5  
FLOAT 10-5  
FOOTER 10-6  
Footers 5-4, 6-4  
Form letters 8-7  
FROM 6-5, 10-6  
FUTIL 4-9

## G

GMODIFY 8-12, 9-5  
Greater-than > 10-2

## H

HALF-DUPLEX/FULL-DUPLEX  
  switch 1-9  
Hanging indents 8-2  
Hard-copy terminals 1-8, 4-5  
HEADER 10-6  
Headers 5-4, 6-4  
HYPHEN 10-6  
HYPHEN character (RUNOFF)  
  6-7

## I

Illegal commands 5-15  
Implicit break 5-9  
Implicit eject 5-11  
INDENT 6-3, 10-7  
Indenting 8-2  
Indenting for paragraphs 5-10  
Indenting text 6-3  
INDEX 6-11, 10-7  
Indexing 6-11  
Initial text period 6-9  
INLIN 8-15  
Input 1-4  
INPUT 9-6  
INPUT mode of EDITOR 3-3,  
  3-5, 3-6

INSERT 3-14, 10-7, 3-6, 8-2, 9-6  
Inserting semicolons into text  
  3-14  
Inserting text in RUNOFF 6-6  
Insufficient access rights  
  (LOGIN) 2-4  
Inter-column margin 6-3  
IXFILE 6-11, 10-8

## J, K

Justification of text 5-6  
Keyboards 1-8  
Keywords 1-8  
KILL (in EDITOR) 9-6  
KILL (in RUNOFF) 10-8  
Kill character 2-2, 3-2  
KILL character (RUNOFF) 6-7

## L

Leaving EDITOR 3-15, 3-16  
LENGTH 6-1, 10-8  
Line numbers in EDITOR 3-6,  
  3-9  
Line pointer 3-5  
Line-formatting 5-6  
LINE/LOCAL switch 1-9  
Line/Local switch 2-2  
Lineprinter 4-6  
Lines in EDITOR 3-5  
LINESZ 9-6  
List files 2-5  
LISTF 2-5, 4-2  
LOAD 9-6  
LOCATE 3-11, 9-7  
Logging in 2-2, 2-3  
Logging in, problems 2-4  
Logging out 2-6  
Login 1-4  
LOGIN 2-2, 2-3  
LOGIN PLEASE 2-4  
Logout 1-4  
LOGOUT 2-6  
Looking at files 4-5

## M

Making directories 4-3  
Making sub-UFDs 4-3  
Margins 5-4, 6-3  
Messages 1-6  
MODE:  
  CKPAR 9-7  
  COLUMN 9-7  
  COUNT 9-7  
  NCKPAR 9-7  
  NCOLUMN 9-7  
  NCOUNT 9-7  
  NNUMBER 9-7

NPROMPT 9-8  
NUMBER 9-7  
PRALL 9-8  
PRLLOWER 9-8  
PROMPT 9-8  
PRUPPER 9-8  
Modem 2-3  
MODIFY 9-8  
MOVE 8-16, 9-9  
Moving EDITOR's pointer 3-9  
Multi-section documents 8-1  
Multiple columns 6-4

## N

NADJUST 10-9  
NEED 6-6, 10-9  
Negative indentation 8-2  
NERRGO 6-12  
New page 5-11  
NEXT 3-10, 9-9  
NFILE 6-5, 10-9  
NFILL 10-9  
NFILL mode 5-6  
NFIND 3-11, 9-9  
NFIND(n) 9-9  
NIXFILE 6-11, 10-9  
Not found (LOGIN) 2-4  
NPARAGRAPH 10-9  
NPAUSE 6-12, 10-9  
NPERFORATE 6-12, 10-9  
NTTY 10-9  
Null lines 3-5  
Null string 1-7  
Numbered lines in RUNOFF  
  output 6-6  
Numeric parameters 1-6

## O

Odd and even pages 6-4, 10-4,  
  10-9  
OFOOTER 6-4, 10-9  
OHEADER 6-4, 10-10  
ON/OFF switch 1-8  
Open files, closing of 4-8  
Output 1-5  
OUTPUT 9-9  
Output file 6-5  
OVELAY 3-6, 9-10  
Overwriting old files 3-17

## P

Page:  
  end 5-11  
  formatting 5-4, 6-1  
  headers and footers 6-4  
  length 6-1  
  numbers 5-4

- size 5-4
- width 6-1
- PAGEN 10-10
- PAGEN character (RUNOFF) 6-7
- Pages, range of output 6-5
- Paper tape:
  - punching 9-11
  - reading 9-6
- PARAGRAPH 5-10, 10-10
- Paragraphing 5-9
- Parameter 1-5
- Parameters 1-6
- Parameters:
  - default values 1-6
  - in EDITOR commands 3-2
  - numeric 1-6
  - text 1-6
- Passwords 2-1, 4-2
- Pathnames 4-4
- PAUSE 6-11, 10-10, 9-10
- Percent sign 6-7
- PERFORATE 6-11, 10-11
- Perforation marks 6-11
- Period at beginning of line 6-9
- PICTURE 6-6, 10-11
- Plus sign + 10-2
- POINT 3-10, 9-10
- Pointer to current line 3-5
- Pound sign # 5-4
- PRIMOS 2-1
- PRIMOS commands:
  - ATTACH 4-1, 4-3
  - CLOSE 4-8
  - CNAME 4-8
  - CREATE 4-3
  - DATE 4-8
  - DELETE 4-8
  - FUTIL 4-9
  - LISTF 2-5, 4-2
  - SLIST 4-5
  - SORT 4-8
  - SPOOL 4-5
  - TERM 4-9
  - TERM-NOXOFF 4-6
  - TERM-XOFF 4-5
- PRINT 3-8, 9-10
- Printing files 4-5
- Printing files on lineprinter 4-6
- Processing selected pages only 6-5
- PSYMBOL 9-10
- PTABSET 9-11
- PUNCH 9-11
- PURGE 10-11

## Q, R

- Question mark 1-6, 3-2
- Question-mark error 3-7

- QUIT 3-15, 10-11, 9-11
- Ragged right text 5-6
- RBAR 10-11
- Reminders 1-6
- Removing files 4-8
- Renaming files and sub-UFDs 4-8
- Repeat 9-14
- Repetition of EDITOR commands 8-10
- Requested information 1-5
- RETURN (in RUNOFF) 10-12
- RETURN key 2-2
- RETURN key in EDITOR 3-5
- RETYPE 3-15, 3-6, 9-11
- Revision bars 10-11
- Right-justified text 5-6
- RINDENT 6-3, 10-12
- Rubout key 1-9
- Rules for filenames 3-17
- RUNDENT 6-3, 10-12
- RUNOFF 5-1
- RUNOFF command format 5-2
- RUNOFF:
  - conventions 6-7
  - date symbols 6-9
  - erase character 6-7
  - errors 5-15
  - footers 5-4
  - headers 5-4
  - illegal commands 5-15
  - indentation commands 6-3
  - indexing commands 6-11
  - kill character 6-7
  - line-formatting 5-6
  - margins 5-4, 6-3
  - numbered lines in output 6-6
  - output file 6-5
  - output options 6-4
  - output to terminal 5-11
  - page numbers 5-4
  - page size 5-4
  - page-formatting 5-4, 6-1
  - spacing between lines 5-7
  - special characters 6-7
  - summary of use 5-3, 5-14
  - symbols 6-7
  - underlining 6-9
  - unrecognized commands 5-15

## S

- Saving edited text 3-16
- Searching for particular text 3-11
- Semicolon 3-4, 3-5, 3-6
- Semicolons, inserting into text 3-14
- Session 1-4
- Side margins 6-3
- Single sheet RUNOFF output 6-11

- Single-spacing of lines 5-7
- SKIP 5-9, 10-12
- Skipping lines 5-9
- Slash / 10-2
- SLIST 4-5
- SMARGIN 6-3, 10-12
- SORT 4-8
- Sorting files 4-8
- SOURCE 6-6, 10-12
- SPACE 5-7, 10-13
- Space for artwork 6-6
- Spacing between lines 5-7
- Special characters, entering 3-18
- SPOOL 4-5
- Spool queue, cancelling files from 4-6
- Spool queue, inspecting 4-6
- STOP 10-13
- String 1-4, 1-7
- String buffers 8-15
- String, null 1-7
- Sub-UFD:
  - changing to another 4-3
  - deleting 4-8
  - making 4-3
  - renaming 4-8
- Suppressing RUNOFF error messages 6-12
- Switches:
  - HALF-DUPLEX/FULL-DUPLEX 1-9
  - LINE/LOCAL 1-9
  - ON/OFF 1-8
  - UPPER-CASE/LOWER-CASE 1-9
- Switching between EDIT and INPUT modes 3-5, 3-6
- SYCHAR 10-13
- SYCHAR character (RUNOFF) 6-7
- SYMBOL (EDITOR) 9-11
- Symbols 8-1
- Symbols in RUNOFF 6-8
- System Administrator 2-1

## T

- TAB 10-13
- TAB character (RUNOFF) 6-7
- Table of contents 7-11
- Tabs in EDITOR 3-18
- Tabs in RUNOFF 5-7
- TABSET 9-13
- Telephone line 2-3
- TERM 4-9
- TERM-NOXOFF 4-6
- TERM-XOFF 4-5
- Terminal 1-4

Terminal controls 1-8  
Terminal switches 1-8  
Terminal, use of 2-2  
Terminals 1-8  
Terminals:  
  hard-copy 1-8  
  keyboards 1-8  
  upper-case only 9-8  
  video 1-8  
Text parameters 1-6, 1-7  
Text string 1-4  
Text string parameters 3-3  
TMARGIN 6-3, 10-14  
TO 6-5, 10-14  
TOFC 7-11, 10-14  
TOFC 0 10-14  
TOP 3-9, 9-13  
Top margin 6-3  
Triple-spacing of lines 5-7  
TTOFC 7-13, 10-14  
TTY 5-11, 10-14  
Two-column text 6-4  
Typographical conventions 1-8

## U

UFD 2-1, 2-5  
UFD, changing to another 4-1  
UNDEFINE 10-14  
UNDENT 6-3, 10-14  
Underlining 6-9  
UNLOAD 9-13  
UNLOAD TO 9-13  
Unrecognized commands 5-15  
Up-arrow (^) 1-9  
Up-arrow character 3-18  
UPPER-CASE/LOWER-CASE  
  switch 1-9  
User File Directory (UFD) 2-1,  
  2-5

## V

Variables, see symbols  
Verbatim text 10-2  
VERIFY 9-13  
Vertical lines 10-11  
Video terminals 1-8  
Viewing files 4-5

## W, X, Y, Z

WHERE 3-9, 9-14  
WIDOW 6-6, 10-14  
Widow prevention 6-6  
WIDTH 6-1, 10-15  
Working directory 4-1  
XEQ 8-16, 9-14  
XOFF 4-5  
XON 4-5

# Prime Technical Publications:

Unique services,  
and an invitation for  
you to participate.

## PRIME Computer



# AIDUS

## **Automatic updates and the fastest method of ordering documentation from Prime**

AIDUS, the Automatic Individual Update Service, is a unique and valuable service for Prime users. Through automatic updates for Final Documentation Release (FDR) manuals, it assures you of timely information on all Prime product enhancements and software revisions. It also provides the fastest method of ordering documentation from Prime.

When Prime revises its software, AIDUS documents these revisions in the form of change sheets that let you easily replace non-current information with new pages. Change sheets encompass corrections, enhancements, and differences between a new release of Prime software and its predecessor. To highlight changes, they have notations in the margins that indicate any applicable software revision numbers. If we revise a book instead of producing change sheets you're covered—we'll send you the new book. In this way, your FDRs remain current and complete.

AIDUS also covers Programmer's Companions™, the condensed technical summaries produced in pocket-size format for easy reference. These are updated by complete reprints, not by change sheets. Each time Programmer's Companion is revised and reprinted, you will be sent a fresh copy as part of the AIDUS service.

The price for this three-year service is only \$15.00 for each FDR and \$5.00 for each Programmer's Companion. (Please note these prices apply only to the United States. Outside the United States consult your local sales office.) Quantity discounts are available, based on the total number of AIDUS packages ordered. This means you

can have the convenience of individual service and the savings of having a coordinator distribute updates to several users.

You can also use the AIDUS order form to get the fastest service when ordering documentation from Prime, even if you don't order AIDUS. You'll benefit from our aggressive quantity discount schedule which affords up to a 30% discount when 101 or more documents are ordered at the same time. Qualified non-profit educational institutions are eligible for a straight 50% discount on all orders for AIDUS and currently listed documentation.

For more information and an AIDUS order form, check the box at the bottom of the User Survey.

# PRIME SOFTWARE DOCUMENTATION SUMMARY

	<b>Title</b>	<b>Doc. Number</b>	<b>Rev.</b>
<b>FORTRAN</b>	The FORTRAN Reference Guide Perfect-bound edition	FDR3057-101A	17
	Loose-leaf edition	FDR3057-101B	17
	The FORTRAN Programmer's Companion	FDR3338	17
<b>FORTRAN 77</b>	The FORTRAN 77 Reference Guide	IDR4029	17
<b>COBOL</b>	The COBOL Reference Guide Perfect-bound edition	FDR3056-101A	17
	Loose-leaf edition	FDR3056-101B	17
<b>PL/I</b>	The PL/I Subset G Reference Guide	IDR4031	17
<b>RPGII</b>	The RPGII Programmer's Guide	PDR3031	16
	Technical update	PTU2600-066	17
	The RPGII Debugging Template	FDR3275	16
<b>BASIC/VM</b>	The BASIC/VM Programmer's Guide Perfect-bound edition	FDR3058-101A	17
	Loose-leaf edition	FDR3058-101B	17
	The BASIC/VM Programmer's Companion	FDR3341	16
<b>BASIC INTERPRETIVE</b>	The Interpretive BASIC Programmer's Guide	IDR1813	16
<b>ASSEMBLY LANGUAGE</b>	The Assembly Language Programmer's Guide Perfect-bound edition	FDR3059-101A	16
	Loose-leaf edition	FDR3059-101B	16
	Correction sheet updates	COR3059-001	17
	The Assembly Language Programmer's Companion	FDR3340	16
	The System Architecture Reference Guide	PDR3060	N/A
<b>PRIMOS OPERATING SYSTEM/UTILITIES</b>	The PRIMOS Commands Reference Guide Perfect-bound edition	FDR3108-101A	17
	Loose-leaf edition	FDR3108-101B	17
	The PRIMOS Programmer's Companion	FDR3250	16
	The Prime User's Guide	IDR4130	17
	The System Administrator's Guide	PDR3109	17
	The System Administrator's Programmer's Companion	FDR3622	16
	PRIMOS Subroutines Reference Guide	PDR3621	17
	LOAD and SEG Reference Guide	IDR3524	16
	Technical update	PTU2600-064	17
The Source Level Debugger Guide	IDR4033	17	
<b>TEXT PROCESSING/EDITOR</b>	The New User's Guide to EDITOR and RUNOFF Perfect-bound edition	FDR3104-101A	17
	Loose-leaf edition	FDR3104-101B	17
<b>DATA MANAGEMENT</b>	DBMS Administrator's Guide	PDR3276	16
	DBMS Schema Reference Guide	PDR3044	16
	DBMS FORTRAN Reference Guide	PDR3045	16
	DBMS COBOL Reference Guide	PDR3046	16
	The PRIME/POWER Guide	IDR3709	16
	The MIDAS Reference Guide	IDR3061	16
	Technical update	PTU2600-062	17
	The FORMS Programmer's Guide Technical update	PDR3040 PTU2600-061	16 17
<b>COMMUNICATIONS</b>	The PRIMENET Guide Technical update	IDR3710 PTU2600-065	16 17
	The Distributed Processing Terminal Executive Guide	IDR4035	17
	The Remote Job Entry Guide	IDR4036	17
<b>STATISTICS</b>	The SPSS Programmer's Guide	PDR3173	16
<b>SYSTEM INSTALLATION</b>	The System Installer's Guide	PDR3105	N/A

## **Help us find out how we're doing and we'll send you a free Programmer's Companion**

Because we're a user-oriented organization we need your ideas and comments. To obtain this information we developed the Technical Publications User Survey. It's an easy-to-fill-out questionnaire that deals with all phases of our technical publications effort. Your answers will help us plan better and more effective publications to meet your needs.

The first part of the survey deals with our publications in general. We need to know a little bit about you, what you do and how much experience with Prime you've had so that we can interpret your comments better. The second part asks questions about a specific book. You may choose to review any Prime document on our Current Documentation Summary.

As a token of our thanks for completing this survey, we'd like to send you a complementary copy of the Programmer's Companion of your choice. Check the box indicating your choice at the bottom of the survey form.

# The Technical Publications User Survey

## Part One

Your name \_\_\_\_\_

Company or School \_\_\_\_\_

Address \_\_\_\_\_

City, State, Zip \_\_\_\_\_

1. What is your job title or function? \_\_\_\_\_

2. What specific task describes what you do?  
(i.e., Systems Programmer, Data Entry Clerk, etc.)  
\_\_\_\_\_

3. Does your company/school/organization now own a Prime Computer?  Yes  No

If YES, what model? \_\_\_\_\_

a. How long have you been using it? \_\_\_\_\_

b. Is it networked with other Prime Computers?  Yes  No

c. Is it networked with other non-Prime computers?  Yes  No

Which ones? \_\_\_\_\_

d. Which of the following software packages do you use?

- |   |                                 |                                   |
|---|---------------------------------|-----------------------------------|
| <input type="checkbox"/> FORTRAN                  | <input type="checkbox"/> COBOL  | <input type="checkbox"/> BASIC/VM |
| <input type="checkbox"/> FORTRAN 77               | <input type="checkbox"/> PL/I-G | <input type="checkbox"/> POWER    |
| <input type="checkbox"/> MIDAS                    | <input type="checkbox"/> DBMS   | <input type="checkbox"/> SPSS     |
| <input type="checkbox"/> RPG II                   | <input type="checkbox"/> FORMS  |                                   |
| <input type="checkbox"/> PRIMENET                 |                                 |                                   |
| <input type="checkbox"/> Remote Job entry (which) |                                 |                                   |

e. Have you previously read any of Prime's documentation?  Yes  No

If YES, what title(s)? \_\_\_\_\_

4. Are you presently evaluating Prime Computers?  Yes  No

If YES, are you using the documentation as part of the evaluation process?

Yes  No

## Part Two

What book are you reviewing? FDR 3104

1. My initial reaction to this book was:  
 Excellent    Very Good    Good    Fair    Poor
2. After reading it, my reaction was:    Better    The Same    Worse  
If either BETTER or WORSE, why? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
3. How much have you used this book?    Just Got It  
 A Little    Fairly Often    Very Often    Every Day
4. Did the book have the content you expected it to have?    Yes    No  
If NO, what would you add or delete? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
5. Did the organization of the material aid you in locating topics of interest?    Yes    No  
If NO, what about the organization was a problem? \_\_\_\_\_  
\_\_\_\_\_
6. Were there too many or too few examples?  
 Too Many    Too Few    About Right
7. Were there too many or too few illustrations?  
 Too Many    Too Few    About Right
8. Could you locate the information you needed?    Yes    No
9. Did you use the index?    Yes    No
10. Was the index adequate?    Yes    No
11. Please give us some feedback on the writing style and editorial quality:
  - a. Clarity    Hard To Understand    Average    Very Clear
  - b. Tone    Stilted    Neutral    Friendly    Patronizing
  - c. Technical Level    Oversimplified    About Right    Too Technical
  - d. General Writing Quality    Poor    Average    Good    Excellent
  - e. Editorial Quality (Typos, misspellings, etc.)  
 Poor    Average    Good    Excellent

12. Have you used documentation from other computer manufacturers?  Yes  No

Which manufacturer? \_\_\_\_\_

How good is Primes compared to theirs?

Much Worse  Worse  Same  Little Better  Much Better

13. If the book you are reviewing is either an FDR or a Programmer's Companion, please answer the following:

a. Did you like the general presentation?  Yes  No

b. Do you like the color paper the book is printed on?  Yes  No

If NO, what would you change? \_\_\_\_\_

c. Do you like the way we used the color and graphics throughout?  Yes  No

If NO, what would you change? \_\_\_\_\_

d. Do you think that showing abbreviations and user input in a second color is effective?  Yes  No

e. Did the screens over the tables and charts make it EASIER or HARDER to read?  Easier  Harder

f. Do you like the Programmer's Companion concept?  Yes  No

If NO, what would you change? \_\_\_\_\_

g. Which form of bindery do you find most useful?  Loose-Leaf  Bound

h. Do you know anything about the AIDUS update program?  Yes  No

Any additional comments or suggestions you might have: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Thank you for filling out our User Survey. Check off which Programmer's Companion you'd like to receive as a token of our appreciation.

PRIMOS  FORTRAN  BASIC/VM  Assembly Language

System Administrator  COBOL  Power

Send me information on the AIDUS program.

© 1980 by Prime Computer, Inc., 500 Old  
Technical Path, Framingham, MA, 01901  
in USA. Specifications subject to  
change without notice. Designed by  
Nangle Associates, Printer  
JRK-Burton



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

FIRST CLASS PERMIT NO. 531 WELLESLEY HILLS, MA 02181

**BUSINESS REPLY MAIL**

Postage will be paid by:

**PRIME** Computer  
Attention: Technical Publications Bldg. 10B  
40 Walnut St., Wellesley Hills, MA 02181

